Probabilistic Model Checking for Dependability Properties of Stochastic Systems

Nawal Addouche, Christian Antoine, Jacky Montmain

Probabilistic Model Checking for Dependability Properties of Stochastic Systems

N. Addouche, C. Antoine & J. Montmain
URC-CEA /LGI2P Research Center of Ales School of Mines, France

ABSTRACT: The paper proposes a dependability analysis method based on probabilistic model checking. Using the profile DAMRTS (Dependability Analysis Models for Real-Time Systems), stochastic real-time systems are modelled with stochastic and probabilistic information. In this profile, static model of the system includes depen-dability information excerpted from fault trees. Behavioural UML models are given with combined collaboration statecharts diagrams. A method of translating these models to continuous time Markov chains (CTMCs) is proposed. The CTMCs are widely used in the context of performance and reliability evaluation of various systems. A formal approach is proposed to verify temporal probabilistic properties related to dependability of stochastic systems. It consists to use a probabilistic model checker which supports the CTMC models and then to specify properties with the suitable temporal logic.

1 INTRODUCTION

Recently, the analysis of functional system requirements in combination with quantitative aspects of system behaviour has come into focus. Several approaches have already been explored to introduce 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 & Pooley 1999, Merseguer & 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. The profile DAMRTS is presented with UML statecharts extended with stochastic information. 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 verifying formally properties related to system dependability.

In our works, we focus on UML statechart diagrams, which allow describing dynamic aspects of system behaviour. In section 2, a part of the profile DAMRTS is presented with extended UML statecharts. Dependability information excerpted from faults trees analysis (failures with their causes) are included in these statecharts. A collaboration diagram and a set of extended UML statecharts describing the behaviour of an assembly chain are 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 THE PROFILE DAMRTS

A profile is composed by a set of UML diagrams selected in the UML metamodel. The extension mechanisms (stereotypes, tagged values, constraints) make it possible to add new elements specific to the application domain (OMG. 1999). The UML profile SPT specialises the UML reference metamodel in a specific metamodel dedicated to domain of real-time systems analysis. The DAMRTS profile is a specific profile designed for dependability analysis of a real-time system (see Fig.1). It is based on concepts defined in the profile SPT with new stereotypes. Those are added to the metamodel in order to introduce particular dependability information. The malfunctions considered as undesirable events and their possible causes are modelled with stereotypes.

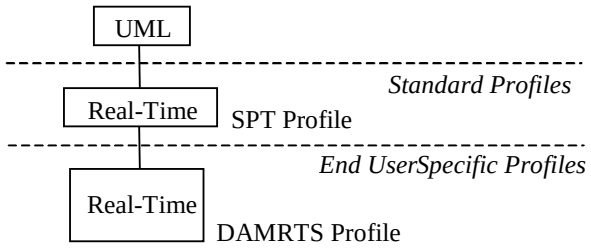

Figure 1. Standard and specific profiles

2.1 *Indicator and Cause stereotypes*

The static aspect of the system is described by a class diagram. It is based on general resources model proposed in the profile SPT. Each component is modelled with a *resource* class associated to *indicato*r and *Cause* stereotypes.

Indicator Class. Each *resource* class is associated to a *Indicator* class. One or several variables (booleans or integers) are defined as attributes of *Indicator* class. These variables are related to dynamic aspect of the *resource* class objects and represent their degraded or failure states. When evaluation of variable became true, it means that the object is in a malfunction state.

Cause Class. Attributes of *Cause class* is defined in terms of the corresponding *Indicator* class attributes. They represent one or several logical expressions composed by an elementary logical conditions linked by conjunctive and disjunctive connectors. Variables used in the elementary conditions are those defined in Indicator classes. Actions of Cause stereotype consist to evaluate the logical expressions. When one of them is true, it indicates that associated failure became true.

For more information about the static model proposed in the profile DAMRTS, see (Addouche et al.2004)

2.2 *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 systems 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).

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 particular qualitative information which represents 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.

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. Figure 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.

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 representing 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 figure 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.

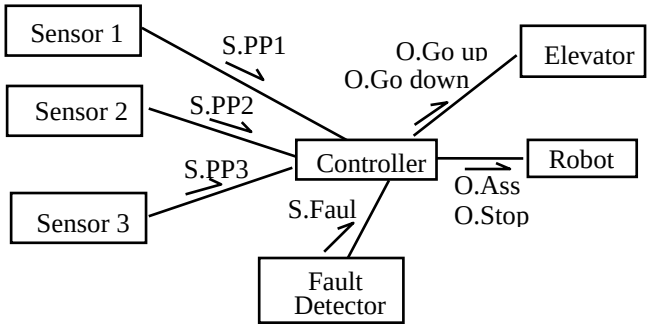

Figure 2. Collaboration diagram

3.2 *Statechart diagrams*

*Robo*t and *Controller* objects behaviour are respectively modelled as given in figure 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 or
ders are modelled in *Controller* statechart as a fault which arrives randomly.

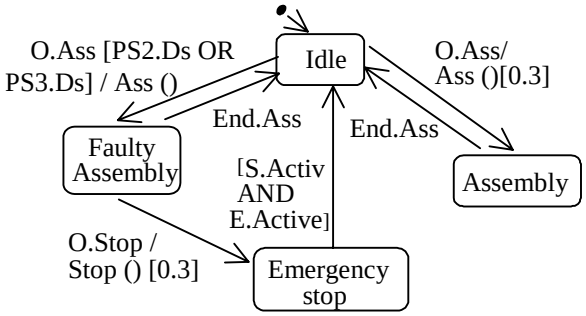

Figure 3. Robot statechart

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.

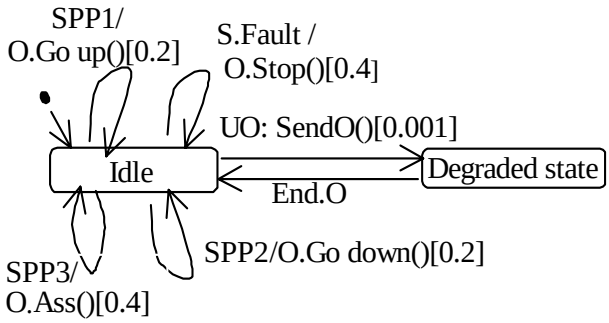

Figure 4. Controller statechart

4 TRANSLATION OF UML MODELS WITH CTMCS

4.1 *Principle of translation*

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 figure 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.

4.2 *Application*

The figure 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 figure 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.

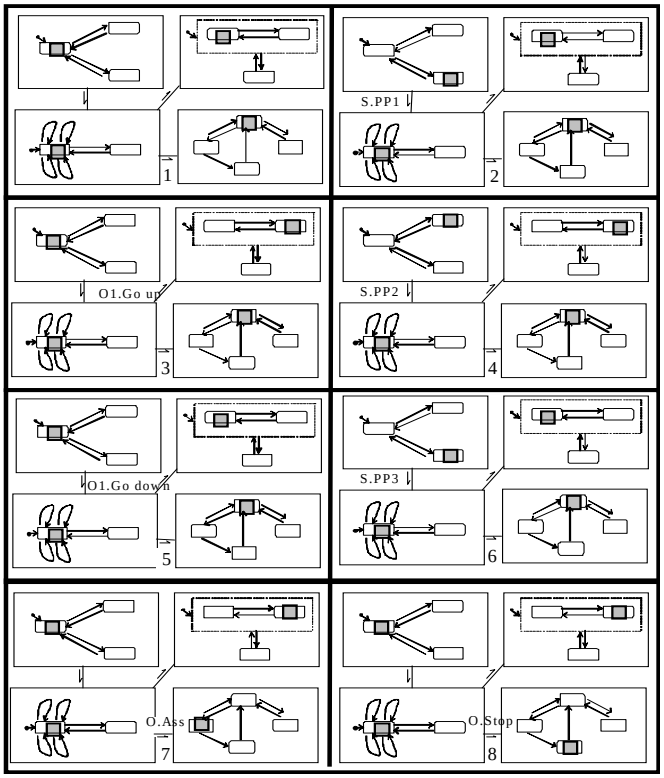

Figure 5. Direct marking of combined collaboration statechart diagrams

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 does 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. The figure 6 shows the continuous time Markov chain derived from model of figure 5. The numbers on reachable states are related to numbers on markings of figure 5.

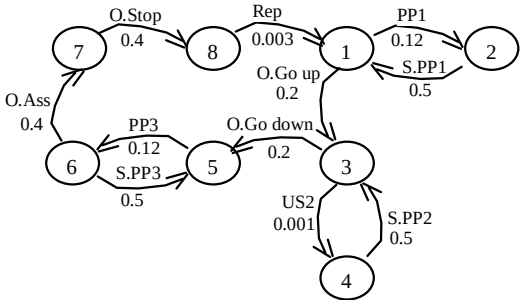

Figure 6. Continuous time Markov chain

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 figure 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 neces-

sary 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

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 & 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 figure 7.

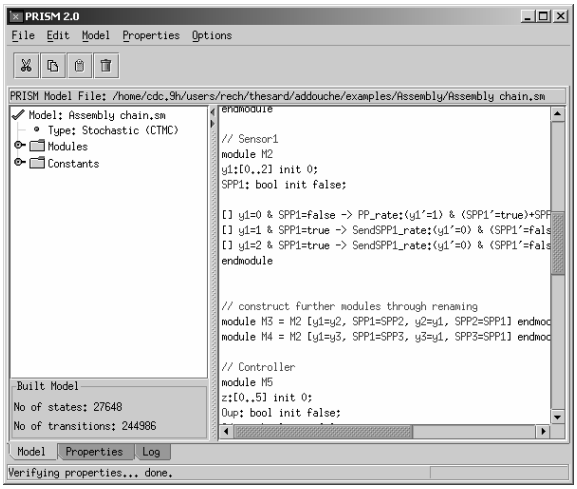

Figure 7. Prism interface: Editing a model

5.2 *Properties in Prism specification language*

Some probabilistic properties related to our example are presented. Their informal specifications are 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"$\Rightarrow$ 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 figure 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.

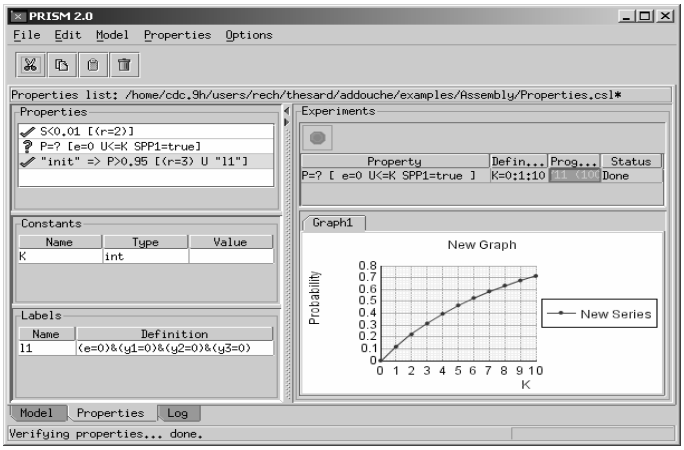

Figure 8. Prism interface: Properties specification

6 CONCLUSION

The paper presents UML models to analyse the system dependability. The aim is to verify formally depen-dability properties of stochastic systems. Extended UML statecharts are proposed to introduce stochastic information related to dependability of real-time systems. Duration of activities is expressed with rate. 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.

7 REFERENCES

Addouche, N. Antoine, C. & Montmain, J. 2004. UML Models for Dependability Analysis of Real-Time Systems. In *Proc of SMC'04*, The Hague, The Netherlands.

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

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

Canevet, C. Gilmore, S. Hillston, J. & Stevens, P. 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. Katoen, J-P. & Brinksma, E. 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. Latella, D. & Massink, M. 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. Hermanns, H. & Kaoten, J-P 2002. A Probabilistic Extension of UML Statecharts: Specification and Verification. In Proc of FTRTFT'02, pp. 355-374. Oldenburg, Germany.

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

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

Kwiatkowska, M. Norman, G. & Parker, D. 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. Majzik, I. & Massink, M. 1999, Towards a formal operational semantics of UML statechart diagrams. In Proc of FMOODS'99, pp. 331-347, Florence, Italy.

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

Merseguer, J & Campos, J. 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 1999. White Paper on the Profile mechanism v.1.0, Analysis and Design Platform Task Force, OMG Document ad/99-04-07.

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.