# Combining Extended UML Models and Formal Methods to Analyze Real-Time Systems

Nawal Addouche, Christian Antoine, Jacky Montmain

# Combining Extended UML Models and Formal Methods to Analyze Real-Time Systems

Nawal Addouche[1], Christian Antoine[2] and Jacky Montmain[2]

[1] Ecole des mines d'Alès, Parc Scientifique Georges Besse,
30035 Nîmes, France
nawal.addouche@ema.fr
[2] URC CEA-EMA, Parc Scientifique Georges Besse,
30035 Nîmes, France
{christian.antoine, jacky.montmain}@ema.fr

**Abstract.** In the paper, we present a methodology developed in order to verify probabilistic temporal properties related to dependability of real-time systems. The methodology is made of three essential steps. The first one is a UML profile called DAMRTS (Dependability Analysis Models for Real-Time Systems) designed using GME tool. The aim is to model a real-time system with qualitative and quantitative information related to its quality of service. In this profile, UML statecharts are used to represent the system behavior. An extension is introduced with probabilities, real-time requirements and nondeterministic choices. The second one proposes a translation from the extended UML statecharts to probabilistic timed automata (PTAs). In this step, global clocks are used to represent synchronization of concurrent UML statecharts in probabilistic timed automata. The last one concerns a probabilistic model checking with PRISM tool. This requires specification of dependability properties with a suitable temporal logic.

## 1 Introduction

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 [7]. 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. Other approaches are also proposed to formalize UML models which are extended with quantitative information. Dynamic UML models are formalised with stochastic Petri nets in [15], with stochastic process algebra in [5] or with continuous time Markov Chains such as we proposed it in [2]. This one is adequate for the performance evaluation and the verification of some dependability properties. However, the formal model contains only rates. Then, it is not suitable for modeling real-time systems. Different models exist to describe real-time systems such as timed automata [3] which have a clear semantics and for which a tool support for automatic verification (Uppaal, Kronos) is available.

The Unified Modeling Language (UML) [13] which becomes an official standard of the Object Management Group (OMG) is widely adopted in industry. This semi-formal language, easy-understood and well-established design notation in the software engineering community, is extended to support the aspect-oriented design for a system. UML support many application domains and provides a common notation independent of the kind of systems that are developed.

To combine the advantages of intuitive modeling by UML with formal verification, we chose the approach which consists to transform UML models into the input language of an existing model checker. Input of a model checker is a formal description of a system. For formal analysis, it is necessary to define what kind of semantic requirements are implied by the domain and what kind of semantics that easily allows translation into the formal model to be analyzed by the model checker. Our contribution includes:

- Definition of the profile DAMRTS [1] for modeling and analyzing real-time systems: a class diagram is proposed to represent static model with quality of service of system components; the conventional UML statecharts are extended with probabilities and real-time requirements,
- Specification of the nondeterminism in extended statecharts and synchronization of concurrent statecharts,
- Métamodeling with the Generic modeling Environment (GME) to construct the proposed profile,
- Translation of extended UML statecharts to probabilistic timed automata: global clocks are defined to represent synchronization of UML statecharts.

In section 2, the methodology of real-time systems analysis is described. The dynamic view of UML models is presented in sections 3 and 4. The nondeterminism and synchronization of extended UML statecharts is given in section 5. We present in section 6, real-time constraints of an assembly chain as well as their behavioral UML models as defined in the profile DAMRTS. The behavioral UML models are nondeterministic, with probabilistic transitions and real-time aspects. That make possible to translate them into probabilistic timed automata as given in section 7. The UML profile is designed using GME tool as presented in section 8. The translation process of resulting models is described in section 9. We conclude with section 10.

## 2 General Methodology

In order to have UML accepted by the real-time development community, the OMG group has proposed a profile called “Schedulability, Performance and Time” [18] for real-time systems. In this profile, some supports are introduced in UML to capture a maximum of real-time requirements and to perform the real-time development tasks

directly on UML models. Beside the usual analysis and design stages, scheduling analysis, performance evaluation and formal verification of critical properties are included. However, the two last activities are partially covered because "quality of service" requirements are introduced without a clear indication about the formal verification of this type of properties. Adapted tools to formal verification or performance evaluation on these UML models are not yet available.

For the reasons indicated above, a new profile called DAMRTS is proposed to analyze and verify dependability properties of real-time systems [1]. It represents an extension to the reference metamodels of the OMG profile "Schedulability, Performance and Time" [18]. The first aim is to be compliant with the standard OMG profile. The second one is to provide concepts that enable to specify a real-time system with its real-time constraints and probabilistic information. A behavioral UML models are proposed with a formal semantics served to probabilistic model checking [10]. It is developed with probabilities and real-time aspects, resulting in probabilistic timed automata as semantics models. These models are used to verify probabilistic temporal properties related to the dependability of real-time systems.

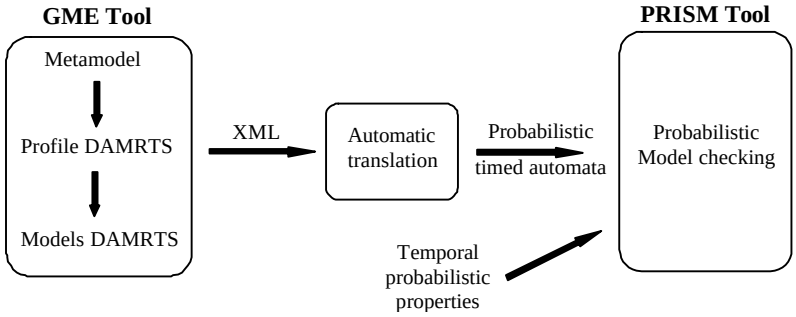


**Fig. 1. Global methodology**

As depicted in fig.1, the proposed approach is presented with the essential steps that allow associating a formal method to an oriented-object concept. The tools GME and PRISM are used for formal verification of real-time system properties. The GME tool is used to construct metamodels specifying the modeling paradigm (modeling language) of our application domain. The modeling paradigm contains all syntactic, semantic and presentation information regarding the domain of real-time systems dependability. This is developed in section 8.

Once the profile DAMRTS is built, we model the real-time system. The output of GME tool is a file having an extended XML format. The DAMRTS models are then exported in XML format for which an automatic translation is applied to transform UML behavioral models into probabilistic timed automata as it is detailed in section 9.

To verify dependability properties, the model checker Prism is adopted [10]. 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. 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 [11] depending on the model type. It then determines which states of the system satisfy each specification.

## 3 Collaboration diagram

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.

A collaboration diagram consists of objects and associations that describe how the objects communicate. It represents the structural organisation of objects which exchanges messages. In the DAMRTS profile, the signals and the orders are the two types of messages taken into account. The first ones are sent from objects of Sensor classes to objects of Controller class. The second ones are sent from instances of Controller class to instances of Effector classes (see fig 3). The proposed statecharts allow expressing events with probabilities and actions with real-time constraints.

## 4 Extended UML Statecharts

In UML, each class has an optional statechart which describes the behavior of its instances (the objects). This statechart receives events from other statecharts and reacts to them. The reactions include sending of the new events to other objects and executing of internal methods on the object. The communications between components of the system are modeled as events. Exchanged signals and orders as well as random events (e.g. undesired and lost signals) are represented as events associated to a discrete probability distribution.

The syntax of UML statecharts, defined in the standard UML [16] is extended as presented below. The operational semantics of UML statechart is inspired from [9] and extended with real-time and probabilistic aspects as presented in [1]. The informal interpretation of extended UML statecharts 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 (Probability)
Guard: = Boolean Expression
Action: = Operation name (Arguments) [Duration, deadline]

Event represents received signals, sent orders or random events with their associated probability. Guard is a boolean expression which represents either *AND-composition* or *OR-composition* states related to degraded or failure states of other objects. The compositions are excerpted from a faults tree analysis of the system [1]. Action expresses operation execution or sending messages to other objects. They are not instantaneous but have duration or deadline. Transitions between states are probabilistic. When two transitions are enabled, the choice is nondeterministic.

## 5 Nondeterminism and Synchronization in UML Statecharts

**Nondeterminism.** Nondeterministic choices can be specified in transition systems by having several transitions leaving from the same state. It is used when we wish to incorporate several potential system behaviors in a model. Nondeterminism is used for several purposes. As it is specified in [6] and [17], it is used to represent phenomena such as:
*Unknown scheduling in concurrent systems.* When a system consists of several components running in parallel, we often do not make any assumptions on the relative speeds of the components, because we want the application to work no matter what these relative speeds are. Therefore nondeterminism is essential to define the parallel composition operator, where we model the choice of which system take the next step as a nondeterministic choice.
*External environment.* A system interacts with its environment via its external actions. When modeling a system, we can not predicate how the environment will behave (failures, abnormal functioning). Therefore the possible interactions with the environment are modeled by nondeterministic choices.
*Uncertainty in probabilities and the expected times.* Sometimes it is not possible to obtain exact information about the system to be modeled. When the exact duration of an action or the exact probability of an event is not known exactly but only with a lower and upper bound. In this case, all possible values are incorporated by nondeterministic choices.

**Synchronisation.** The extended UML statecharts are allowed to communicate with each other in well-defined manners. The communication and synchronization method are presented as follow:

- One UML statechart may create an event as a result of a transition that is consumed by another UML statechart.
- A guard may be used to test if another UML statechart is in a certain state before allowing a transition to occur to the guarded state.

## 6 Example of an Assembly Chain

This example presents an automated chain assembly of electrical micro-motors. It is excerpted from a European project named PABADIS (Plant Automation BAsed on DIstributed Systems) [14]. This one deals with a flexible and a reconfigurable system designed for production of different types of micro-motors. Fig 2 represents the controlled system.

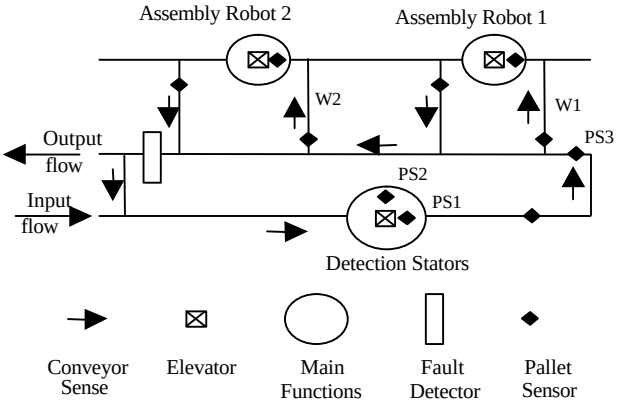


**Fig.2. Assembly chain of micro-motors**

Micro-motors consist to stators and rotors. The first 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.

The assembly robots work in parallel. Let us consider a stators pallet arrives at the level of assembly robots and detected by PS3. If the two robots are both idle, the pallet is arbitrary send to one of the waiting areas w1 or w2 showed in fig 2. If the robots are both busy, the controller send information request to robots. These send the information about the assembly state. The pallet is then leaded to the robot which will be the idle first. This behavior is modeled as given in the statechart of fig 4 and 5.

### 6.1 Collaboration Diagram

In the collaboration diagram of fig 3, interactions between objects are presented. 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 1*, *Robot 2*and *Elevator* objects. Our example presents a distributed system such that several controllers (PLC) interact to control the system functioning. To simplify, we represent in the collaboration diagram one *controller* object.

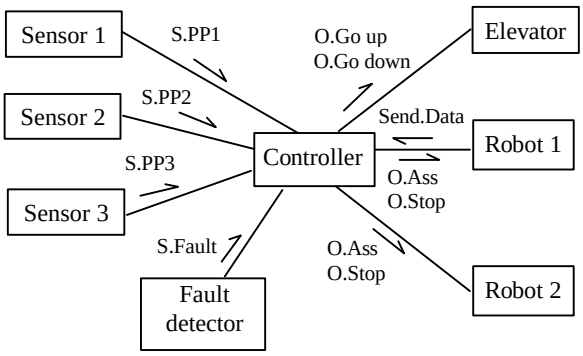


**Fig.3. Collaboration diagram**

### 6.2 Extended UML Statecharts

*Robot* and *Controller* objects behavior are respectively modeled in fig 4 and fig 5. In Robot statechart (describes robot 1 or robot 2); assembly tasks as well as communication with controller are executed in parallel.

In substate A, the controller orders are modeled as events. In transition: "O.Ass (0.10) [PS1.Ds OR PS2.Ds]/ Assembly()[10s]", guard expresses that the edge is enabled if one of sensors PS2 or PS3 is in degraded state Ds. The probability of sending an assembly order when one of sensors is in degraded state is evaluated to 0.10; the execution of the assembly operation, Ass () lasts 10s. Otherwise, the robot performs a correct assembly with probability 0.90. 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. When probability is not represented, it means it is equal to 1.

Nondeterminism is modeled at the level of the state, *Idle*. A probabilistic choice is used to represent the possibility of performing a correct or a faulty assembly. It is also possible that robot remains idle when there is absence of pallets (AP). Substate B, describes the controller requests and sending of data from robot to controller.

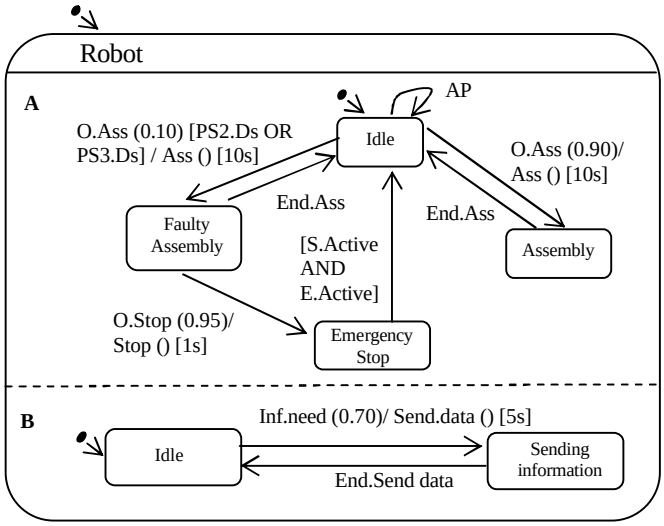


**Fig.4 Robot Statechart**

Fig 5 describes controller behavior. When signal SPP3 becomes true (sensor PS3 detects a pallet), the order Info.need is send to the robots. After receiving information, the controller send the order of assembly for one of the two robots (which will be the idle first).

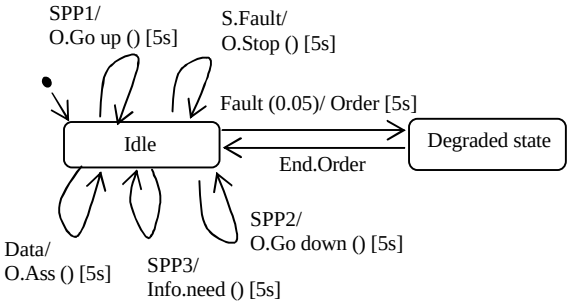


**Fig.5. Controller Statechart**

Among the malfunctions of controller, sending of undesired orders or lost orders are modeled in *Controller* statechart as random events, e.g. "Fault (0.05)/ Order [5s]". The received sensor signals are presented as events and the sending of orders as actions with their associated deadlines. The edge "SPP1/ O.Go up()[5s]", expresses that when PS1 detects a pallet, order to go up from controller to elevator is send.

## 7 Translating Extended UML Statecharts to PTAs

Timed automata are automata extended with clocks, positive real valued variables which increase uniformly with time, and whose nodes and edges are labeled with clocks constraints, respectively called invariants and guards. The invariant dictate when the automaton may remain in a node, letting time pass, and guards when the corresponding edge can be taken [3]. Probabilistic timed automata are a variant of timed automata extended with discrete probability distributions [11]. This type of automata has been chosen for formalizing extended UML statecharts because it takes into account dense time, nondeterminism and probabilistic choice as defined in the extended UML statecharts. They are also amenable to model check probabilistic temporal properties.

### 7.1 Principle of translation

To translate extended UML statecharts to probabilistic timed automata, real-time constraints of actions are represented with clocks. Events are described with their probabilities on edges. The guards defined in proposed UML statecharts describe the active state of other objects.

In probabilistic timed automata, it is not really possible to observe the location of another component directly as the principle defined in our extended UML statecharts. However, a probabilistic timed automaton component A can check the location of another component B in the following way: component B is equipped with self-loop edges in all of its locations (or some of its locations), where the events of the self-loop edges would be different for each location. Therefore, in location L1, the probabilistic timed automaton B would have enabled a self-loop edge with an event which is unique to L1: “in-L1”, for example. Then component A, when it want to know whether B is in L1 or not, would try to synchronize on event “in-L1”. If synchronization is possible, then A knows that B is in L1 and can act accordingly; otherwise, it can do something else, knowing that B is not in L1.

### 7.2 Synchronization with Global Clocks

Synchronization between probabilistic timed automata components is done using edge-labeling events, as defined in [12]. One manner to synchronize probabilistic timed automata is to create a probabilistic timed automaton component which has a single clock which is never reset during the execution of the system. Then this clock could be regarded as a “global clock”. This component could then synchronize with the other components when the value of the global clock reaches certain values.

In fig 6, we give probabilistic timed automaton describing a sub-system of the assembly chain example: the robot behavior reacting to controller orders. The probabilities used in the example should in practice be obtained from statistical analysis of observed behavior.

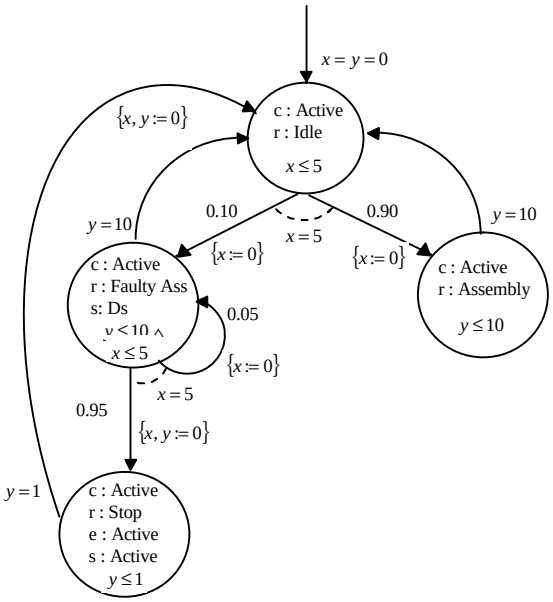


**Fig.6. Probabilistic timed automaton**

In the probabilistic timed automaton, the sub-system consists to robot, controller and two clocks *x* and *y*. Atomic propositions, related to probabilistic timed automata of elevator and sensor, are included in nodes. "s:Ds, e:Active and s:Active" express guards of UML statecharts in fig 4. In initial state, both clocks x and y set to 0. The controller sends assembly order to robot in 5 time units. Then, the robot performs the assembly tasks with probability 0.90. After assembly takes 10 time units, the robot becomes idle. When one sensor is in degraded state, then robot performs a fault assembly with probability 0.10. When an order stop is send by the controller, the robot stop. It becomes idle when sensors and elevator are in active state.

## 8 Metamodeling using GME Tool

The Generic Modeling Environment (GME) developed at the institute for Software Integrated Systems at Vanderbilt University is a configurable toolkit for creating domain-specific modeling and program synthesis environments [13].

There is a metamodeling paradigm defined that configures GME for creating metamodels. These models are then automatically translated into GME configuration information through model interpretation. Once the metamodeling interpreter is operational, a meta-metamodel is created and the metamodeling paradigm is regenerated automatically [13]. The metamodeling paradigm is based on UML notation. The syntactic definitions are modeled using UML class diagrams and the static semantics are specified with constraints using the Object Constraint Language (OCL).

### 8.1 Modeling Concepts

The vocabulary of the domain-specific languages implemented by different GME configurations is based on a set of generic concepts built into GME itself. This one supports various concepts for building complex models.

Folders, FCOs (Models, Atoms, Sets, References, and Connections), Roles, Constraints and Aspects are the main concepts that are used to define a modeling paradigm. The First Class Objects (FCOs) used to represent entities and relations, form the core of the GME concepts. These generic concepts are not generally used at the same time. However, the choice is rather an important design decision.

The concepts used in our metamodel are: Aspects, Models, Atoms and connections. These latter are defined below. The other quoted concepts are defined in [8] and [13].

*Aspects* represent different "views" of the structure of a model. It is not always beneficial to present every object contained in a model all at once; aspects allow choosing what we want to see. A model with several aspects will display different subsets of its contained entities depending on the aspect selected,

*Atoms* are a basic, limited type of entity which has no internal structure (i.e. contained objects). Every feature of an atom that can be represented in a model is contained in the atom's name, attributes, and the relations it participates in,

*Models*, the second generic type of entity, are very similar to atoms. The main difference lies in the ability of models to *contain* atoms, other models, and other types of objects. Thus, models have internal structure. When viewed together, they form tree-like *containment hierarchies* of entities. Models can be *opened*, showing a diagram of their internal structure,

*Connections* are the primary concepts that represent relationships. Connections normally describe a relation between two objects, and this relation is represented as a line in a particular color and style connecting the two objects. Connections can also have their own attributes.

### 8.2 Overview on the Metamodels of DAMRTS

The DAMRTS profile is a specific profile designed for dependability analysis of a real-time system. It is based on concepts defined in the profile SPT [18] 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. The QoS is represented as attributes when it is about actions of resource classes (e.g. duration of actions, response time for a call action, etc.). It is also represented as a tagged value when it is about general QoS, like reliability and maintainability of resources [1].

To build the profile DAMRTS, the metamodeling paradigm based on UML is used. Three UML metamodels are created to represent class diagram, collaboration diagram and extended UML statecharts of a real-time system. Such as presented in fig 7, the metamodel of the class diagram contains the concept Atoms: sensor, effec-

tor (all components in contact with raw material such robots, conveyor belt, etc.) and controller.

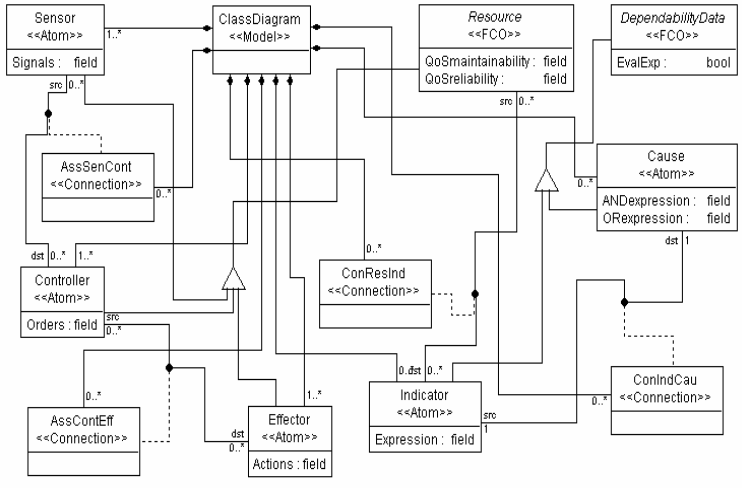


**Fig.7. Class diagram metamodel of profile DAMRTS**

The FCOs *Resource* and *Dependability* are defined to be hidden in the class diagram. We use the concept Attributes (which does not necessarily represent attributes in the class diagram) to define the QoS related to dependability of *Resource* as well as the methods send signals, send orders and actions related to the Atoms sensor, controller and effector (see fig 7). The Attributes of GME tool can have a set of specifications such as the data-type [8]. Then we specify the defined Attributes with integer or double according to whether it is of real-time data (deadline and duration of methods) or of probabilities assigned to undesired events.

Each entity *Resource* is associated the Atom *Indicator* which represent the undesired events such failures. The Atom *Cause* has as attribute one or several logical expressions composed by elementary logical conditions linked by conjunctive and disjunctive connectors. This attribute is specified to be boolean. When one of the expressions is true, it indicates that associated failure became true.

## 9 Automatic Transformation Process

Once the UML models are built with the profile DAMRTS, the different view of the real-time system are presented including dependability information. The following step consists to analyze the models. For lack of UML's models analysis tools, we chose a tool which allows analysis of models containing real-time and probabilistic

information such as in the extended UML statecharts. The probabilistic model checker Prism is then used [10].

To allow verification of probabilistic temporal properties, it is necessary to translate behavioral UML models from the GME tool to input model of Prism tool. For this, an automatic translation is performed using the parser XERCES.

The statecharts UML models are exported to XML format. Syntactic analysis is applied on the XML files using the parser. Transformation rules are then defined to rewrite the XML nodes to Prism language based on the Reactive Modules formalism [4]. 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.

## 10 Conclusion

The approach used in our proposition is to enrich the UML model with the local quality of services parameters relevant to a specific analysis objective (for instance, failure/repair rates are associated with elements of UML model) and to automatically transform the relevant parts of the enriched UML models to probabilistic timed automata.

The advantage of the approach is that it is relatively easy to experienced UML users to create extended UML models and automatic translation made it possible to apply Prism. The formal model is correct with respect to requirements of UML model. Writing properties with probabilistic temporal logic such as PCTL is not easy. In Prism, syntax is proposed to express properties. This one is easier than that of probabilistic temporal logic.

Due to the denseness of time, the underlying semantic model of a probabilistic timed automaton is infinite, and hence effective decision procedure rely on building a finite quotient of the state space. In future works, the verification technique used, will be based on the generation of the forward reachability graph with Kronos, and model checking the obtained graph encoded as a Markov decision process with Prism.

## References


1. Addouche, N., Antoine, C., Montmain, J.: UML Models for Dependability Analysis of Real-Time Systems. In: Proc of SMC'04, The Hague, The Netherlands (2004)
2. Addouche, N., Antoine, C., Montmain, J.:"Formalisation of Quantitative UML models Using Continuous Time Markov Chains", Third Conference on Management and Control of Production and Logistics, Santiago, Chile (2004)
3. Alur, R., Dill, D.L.: A Theory of Timed Automata, Theoretical Computer Science, 126(2):183-235 (1994)

4. Alur, R., Henzinger, T.: Reactive Modules, In: Proc of LICS'96, IEEE Computer Society Press, New Jersey (1996), 207-218
5. Canevet, C., Gimore, S., Hillston, J., Stevens, P.: Performance Modelling with UML and Stochastic Process Algebra, In Proc of the Eighteenth Annual UK Performance Engineering Workshop (2002)
6. De Alfaro, L.: Formal Verification of Probabilistic Systems, PhD thesis, Standford University (1997)
7. Gnesi, S., Latella, D., Massink, M.: A Stochastic Extension of a Behavioural Subset of UML Statechart Diagrams, In: Proc of HASE'00, Albuquerque, New Mexico (2000)
8. ISIS.: GME 4 User's Manual, version 4.0, Institute for Software Integrated Systems, Vandertbilt University (2004), http://www.isis.vanderbilt.edu/Projects/gme/
9. Jansen, D.N., Hermanns, H., Katoen, J-P.: A Probabilistic Extension of UML Statecharts: Specification and Verification, FTRTFT 02, Oldenburg, Germany (2002) 355-374
10. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic Model Checker, In Proc.TOOLS 2002, volume 2324 of LNCS (2002), 200-204
11. Kwiatkowska, M.:Model Checking for Probability and Time: From Theory to Practice, In LICS 03, IEEE Computer Society Press (2003) 351-360
12. Kwiatkowska, M., Norman, G., Sproston, J.: Model Checking of Deadline Properties in the IEEE 1394 Fire Wire Root Contention Protocol, Formal Aspects of Computing (2003), 14(3), 295-318
13. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The Generic Modeling Environment, Workshop on Intelligent Signal Processing, Budapest, Hungary (2001)
14. Lüder, A., Peschke, J., Sauter, T., Deter, S., Diep, D.: Distributed Intelligence for Plant Automation on Multi-agent Systems: the PABADIS approach, Production Planning and Control (2004), 15(2), 201-212
15. 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.
16. OMG.: Unified Modeling Language Specification v.1.5, OMG Document Formal / 03-03-01, (2003)
17. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems, PhD thesis, Massachussetts Institute of Technology (1995)
18. Selic, B., Moore, A.: Response to the OMG RFP for Schedulability, Performance and Time: Revised submission, OMG document ad/2001-06-14