Comparing two UML Profiles for Non-functional Requirement Annotations: the SPT and QoS Profiles[⋆]

Simona Bernardi[1] and Dorina C. Petriu[2]

[1] Dipartimento di Informatica
Università di Torino, Torino, Italy
`bernardi@di.unito.it`

[2] Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
`petriu@sce.carleton.ca`

Abstract. The paper compares two UML Profiles adopted by OMG for annotating non-functional requirements of software systems: the *UML Profile for Schedulability, Performance and Time* (SPT) formally adopted in 2003 and the recently emerging *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* (QoS). The SPT Profile was the first attempt to extend UML with basic timing and concurrency concepts, and to express requirements and properties needed for conducting schedulability and performance analysis. While the SPT Profile is focused on these two types of analysis, the more recent QoS Profile has a broader scope, aiming to allow the user to define a wider variety of QoS requirements and properties. In order to compare the two profiles, we will focus on performability and timing aspects of software systems, by exemplifying the concepts through an example of embedded automation system. The comparative analysis shows that new concepts are needed in both profiles to express time intervals between two arbitrary events. Also, the two profiles will need to reach a common agreement on the specification of complex timing values, especially of those with stochastic characteristics. Another open problem is the parameterization of models, as in many cases fixed values for model parameters are not enough. The SPT Profile goes a step further by supporting symbolic variables and expressions, but the QoS Profile does not have such a capability yet. In general, both Profiles struggle with the balance between flexibility (i.e., allow the user to introduce its own definitions) and simplicity/convenience of expression. The challenge when defining a UML profile is to find convenient yet powerful mechanisms of expression for complex concepts, yet to remain within the limits of the UML standard extension mechanisms, which is necessary to insure that the annotated models could be understood by standard UML tools.

1 Introduction

The paper compares two UML Profiles adopted by OMG for annotating non-functional requirements of software systems: the *UML Profile for Schedulability, Performance*

[⋆] This work was done while S.Bernardi was a visiting researcher at Carleton University, Ottawa, with the financial support of the FIRB-Perf project funded by the Italian Ministry of Education, University and Research, and of the PUMA project funded by the Natural Sciences and Engineering Research Council of Canada.

and Time (SPT) [12] formally adopted in 2003 and the recently emerging *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* (QoS)[13].

The SPT Profile was the first attempt to extend UML with basic timing and concurrency concepts, and to express requirements and properties needed for conducting schedulability and performance analysis. The OMG timeline for the SPT Profile started with a Request for Proposals (RFP) in 1999, followed by a first Response to RFP in August 2000, a revised submission in June 2001, the OMG adoption in March 2002 and the formal OMG adoption in September 2003. Now OMG is in the process of issuing a new RFP for SPT-Version2, to bring it in line with UML 2.0 and to harmonize it with the emerging QoS Profile. The SPT Profile was implemented in at least two UML tools [6, 7] and was used for schedulability and performance analysis by different groups (e.g., [1, 3, 14, 15]). While the SPT Profile is focused on two types of analysis, schedulability and performance, the more recent QoS Profile has a broader scope, aiming to allow the user to define a wider variety of QoS requirements and properties [5]. The QoS profile was adopted by OMG in June 2004, and now is being improved by the Finalization Task Force, which precedes the formal adoption.

In order to compare the two profiles, we will focus in this paper on performability and timing aspects of software systems, by exemplifying the concepts through an example of embedded automation system. We chose the performability domain due to the fact that it is close enough to the performance domain, and thus it can be covered by the STP Profile (although the profile's limits are stretched). At the same time, the expression of performability aspects with the QoS Profile exercises its flexibility and power of expression.

The motivation for the paper is to provide a useful comparison of the two Profiles at an appropriate moment in time, when one is in the process of being formally adopted and the other will be upgraded. The two Profiles were defined separately, yet they are supposed to complement each other. While the SPT Profile has been applied by a number of research groups, we are not aware of any complete published examples for the application of the QoS Profile. The paper contributions are as follows: a) show how to use the new QoS Profile through an example of automation system characterized by different type of non-functional requirements, b) emphasize the main advantages/drawbacks of the two annotation approaches, and c) identify new concepts that should be introduced in the two Profiles.

The paper is organized as follows: in Section 2 the example of an embedded automation system is presented, Section 3 and Section 4 apply the QoS annotation and the SPT annotation concepts, respectively, on running example. Section 5 is devoted to the comparison between the approaches of the two Profiles. Conclusions are given in Section 6.

2 Example of an embedded automation system

In this section we introduce a revised version of the embedded automation system example presented in [2]. The logical view of the system is represented by the Class Diagram of Figure 1. The left part of the diagram represents the basic structure of the automation system residing on a given automation site: the automation system controls through its components the components of the plant.

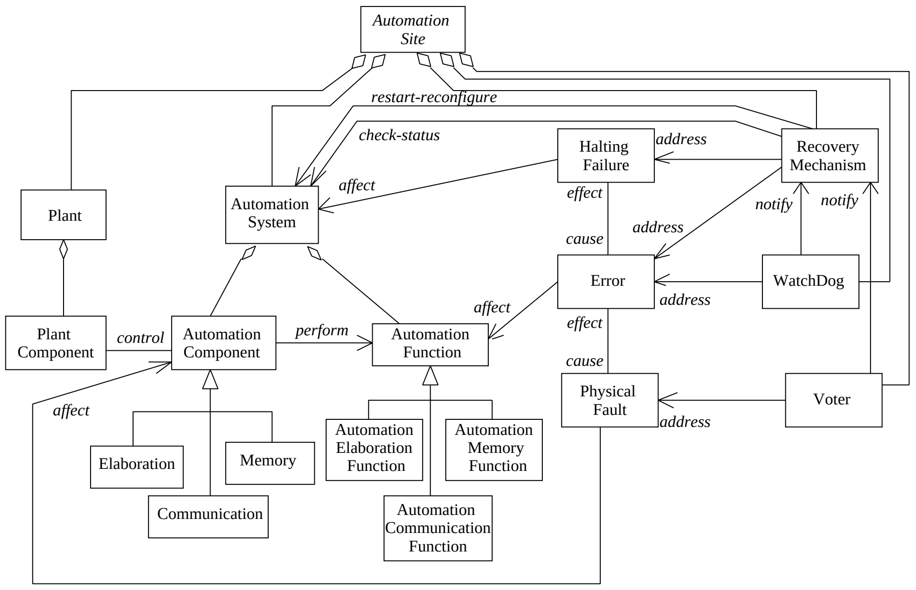

Fig. 1. Logical view of the automation site

There are three types of automation components dealing with communication (*Communication*), elaboration (*Elaboration*) and memory (*Memory*) and three corresponding types of automation functions (*Automation Communication Function*, *Automation Elaboration Function*, and *Automation Memory Function*).

From the behavioral point of view, the system is a distributed cyclic application that activates two concurrent processes: each process reads a sample input from a set of plant components, elaborates the future state, saves the new state in memory and produces the new output for the plant components.

As emphasized in the right part of Figure 1, the automation components can be affected by physical faults that may cause errors in the automation functions: unrecovered errors lead to system failure. To increase the dependability of the automation system a fault-tolerance strategy has been devised consisting of fault masking, error detection, diagnosis and recovery, and reconfiguration from system failure. Fault masking is carried out during the elaboration of the future state by means of spatial redundancy of the computation execution and voting on the results coming from the replicas. Depending on the voting technique adopted and on the spatial redundancy, a limited number of faults may be masked; in this case, the software mechanism responsible for the voting (represented by the *Voter* class in Figure 1) implements a majority voting algorithm and a minimum of 3 computations are concurrently executed, so that at least one fault can be made transparent during the elaboration of the future state of the system. If majority is not reached within a given deadline, then the voter notifies an exception to a recovery mechanism that provides to remove the faulty elaboration units and to reconfigure the system.

The error detection is performed by a standard watchdog mechanism (*WatchDog* class) while error diagnosis and error recovery are implemented by the recovery mechanism. The watchdog is initialized by the automation system, its timer is enabled during the reading and saving operations and it is paused during the elaboration of the future state of the system. At the end of each cycle, the automation system sends an "Iamalive" message to the watchdog in order to reset its timer. In case of time-out expiration, the watchdog notifies an exception to the recovery mechanism, that provides to terminate the watchdog and to check the status of the automation system introducing a delay in the current activity of the system: if no error is present then it is a false alarm, and the watchdog is simply reinitialized. If instead an error is present then a backward recovery action is carried out.

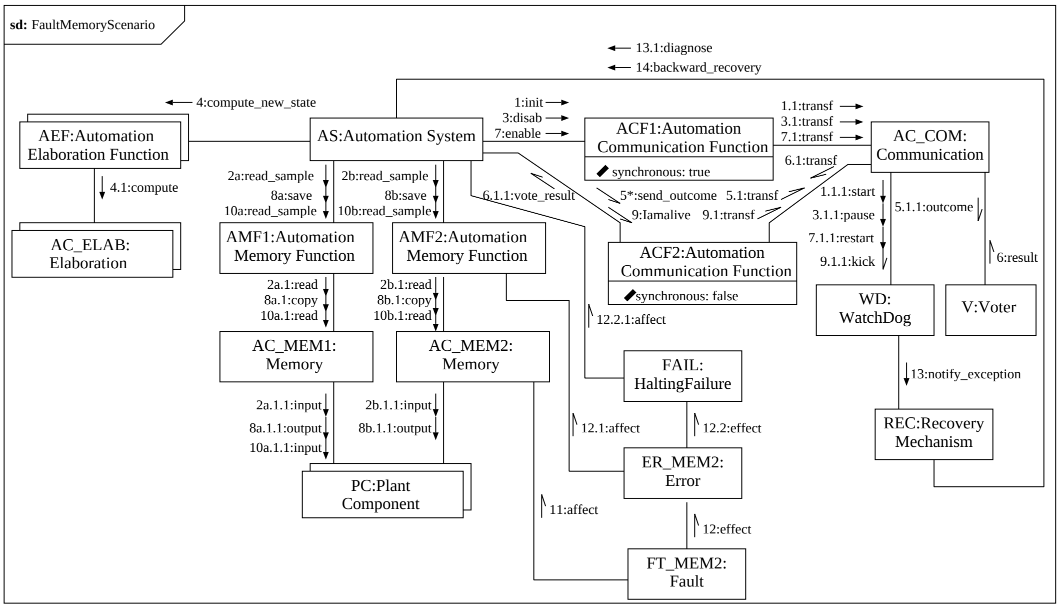

Fig. 2. A fault scenario.

A possible fault scenario is represented by the UML2.0 Communication Diagram of Figure 2: a physical fault in a memory component, *AC_MEM2*, causes an error in the execution of the memory function *AMF2* (reading operation) during the second automation cycle. The error is not recovered in due time and causes an halting failure in the system. The exception is notified by the watchdog to the recovery mechanism that, after the error diagnosis, recovers the system to the previous safe state corresponding to the beginning of the second cycle.

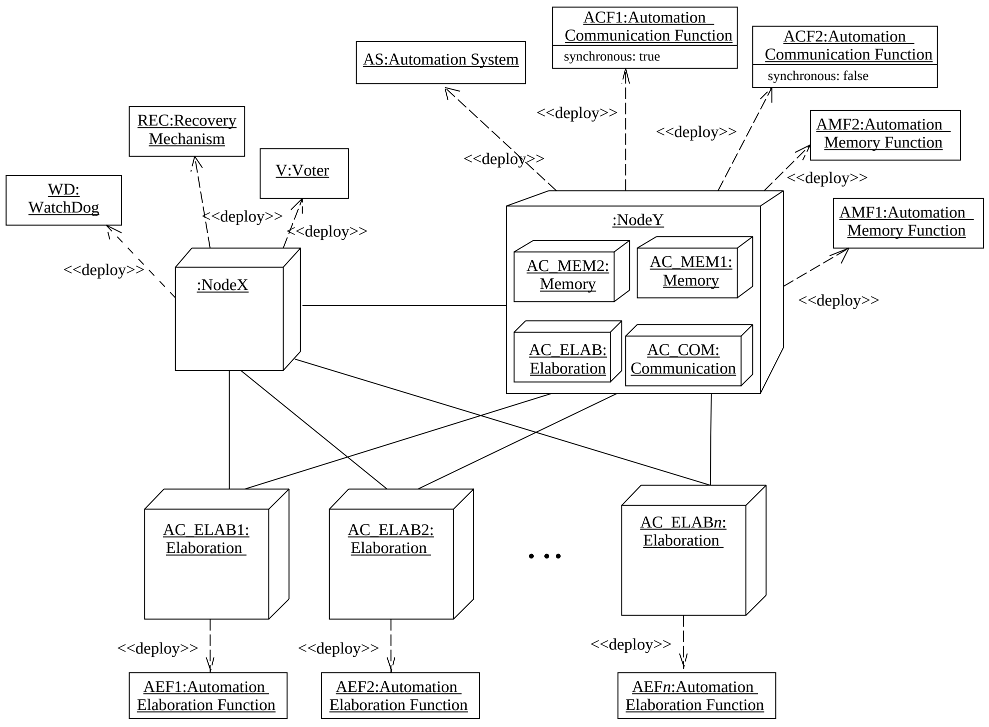

Fig. 3. Physical allocation of the system hw/sw components

The deployment Diagram of Figure 3 shows the physical allocation of the hardware and software components involved in the fault scenario represented in Figure 2. Times required for elaboration and for read/copy operations range uniformly in the interval $[0.5, 1.4]$ (milliseconds) while communication time is assumed exponentially distributed with mean 1 (millisecond). The rate of occurrence of the physical fault affecting a memory component and its duration are assumed exponentially distributed with different means (1 hour and 1 second, respectively). The time-out for the voter to wait for the reception of the outcomes from the automation system is set to 1 second from the reception of the first outcome. The voting operation on the available outcomes requires a negligible amount of time. The time spent by the recovery mechanism to diagnose the status of the automation system after the reception of an exception is at least of 2 milliseconds while the time required to carry out a backward recovery has to be at most of 3 minutes.

The automation system is characterized by the following dependability and timeliness requirements:

1. the level of availability of the automation system has to reach at least 98% and
2. the cycle time of the automation system, i.e., the time required for reading the input samples, elaborating the future state and producing the output, has to be at most of 15 milliseconds.

Performability issue. We are interested in minimizing the frequency of false alarms raised by the watchdog (i.e., time-out expiration without the presence of an error) and the time overhead introduced by the fault masking actions such that the timing constraints and the dependability requirements are satisfied.

3 QoS Profile annotations

The annotation process proposed by the Quality of Service (QoS) Profile consists in three main steps: the first step consists in defining the QoS characteristics of interest for the analysis to be carried out on a specific system domain. In the second step a "Quality Model" is defined for the specific embedded automation system illustrated previously in which the parameters of the QoS characteristic template classes specified in the first step are all resolved. Finally, in the third step, the UML models of the embedded automation system are annotated with QoS constraints and QoS values according to the QoS characteristics defined in the "Quality Model" and to the QoS characteristics of interest.

Definition of QoS characteristics We have reused some QoS characteristics of the QoS Profile catalog and defined new ones (see Figure 4) that will be used for the QoS annotation of the embedded automation system described previously. It is worth to note that the set of QoS characteristics of interest should be defined by a group of experts of the application domain in order to provide the analysts with a QoS catalog that can be used, as a pattern, for a *specific* type of analysis of a *specific* application domain.

In our context, we do not claim to provide an exhaustive QoS catalog for the performability analysis of embedded automation systems but, rather, we aim to illustrate an example of "customization" of the QoS characteristics defined in the general catalog. We have defined the following new QoS characteristics:

- **resource-service-time**: a new QoS characteristic introduced in the *Efficiency* QoS category. It is a template class that allows to specify the service time (attribute *S*) of the basic resources, in our example memory, elaboration and communication units, in a stochastic manner.
- **alarm-latency**: as a specialization of the *latency* QoS characteristic defined in the *Latency* QoS category. It is a template class that allows to specify the timer duration (attribute *timer-duration*) of some type of alarm-based software mechanisms, such as the watchdog and the voter mechanisms of the example.
- **physical-fault**: as a specialization of the *fault* QoS characteristic defined in the *Dependability* QoS category. It is a template class that allows to specify the quantitative aspects of a physical fault, i.e., its rate of occurrence and its duration (attributes *fault_rate* and *duration*, respectively), in stochastic terms.
- **overhead**: as a specialization of the *fault-tolerance* QoS characteristic defined in the *Dependability* QoS category and of the *turn-around* QoS characteristic defined in the *Latency* QoS category. It is a template class that allows to specify the overhead introduced in the system by the fault-tolerance strategy adopted in a quantitative manner, i.e., in terms of frequency of false alarms (method *false-alarm-frequency-value*) and of time (method *time-overhead-value*).

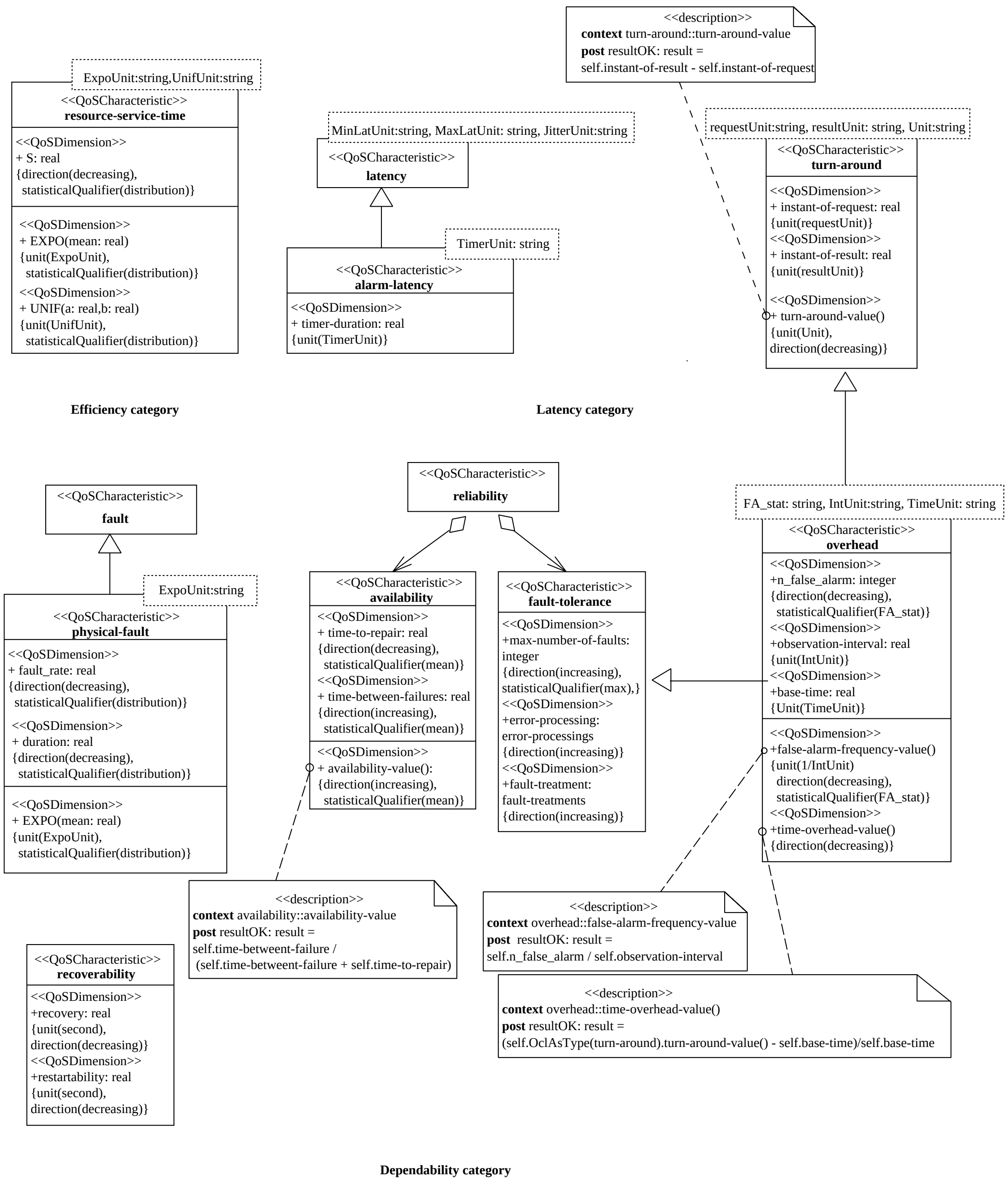

Fig. 4. QoS characteristics for performability analysis of embedded automation systems.

Figure 4 includes also several QoS characteristics of the general QoS catalog (e.g., **availability**, **turn-around**) that will be explicitly re-used for the specification of the non-functional requirements of the embedded automation system under study.

A natural way to customize the QoS characteristics of the general catalog is through specialization, as carried out in the example for all the new QoS characteristics introduced but **resource-service-time**. In particular, note that we have used multiple inheritance to define the **overhead** QoS characteristic since it includes both performance-timeliness properties and dependability- fault tolerance aspects.

Observe that most of the QoS characteristics of Figure 4 are template classes: the QoS Profile does not give any restriction on what to parameterize although it shows only examples of metric unit parameters. In our example, we have defined also a parameter for the type of statistical value (i.e., the *FA_stat* parameter defined in the QoS characteristic **overhead** for the statistical qualifier property of the attribute *n_false_alarm* and of the method *false-alarm-frequency-value*).

The system models used for performability analysis are usually stochastic. According to the QoS Profile, to each attribute/method of a QoS characteristic is associated the *StatisticalQualifier* property that allows to specify the type of statistical qualifier when the value of the attribute/method represents a statistical value such as a probability distribution. However, this property does not allow to declare the type of distribution and the QoS Profile does not provide support to describe stochastic timing concepts.

The solution we have proposed in this paper is a tradeoff between simplicity and flexibility, and is similar to that adopted in the SPT Profile. A stochastic aspect of the example concerns the service times of the basic resources that are random variables distributed according to different distributions depending on the type of resource. The service time is then defined as a QoS characteristics (i.e., **resource-service-time**) and it is characterized by an attribute *S* representing the service time (real) variable, and by two methods *EXPO(mean: real)* and *UNIF(a: real, b: real)* that return, each one, a value in the state space of a random variable distributed according to a specific distribution (i.e., an exponential distribution with mean equal to *mean* and a uniform distribution over the interval $[a,b]$, respectively).

As we will illustrate in the third step of the QoS annotation process, the specification of the service time of a given resource is carried out by assigning to the attribute *S* the return value of the proper method in which actual values are set for the input parameters of the method. A similar approach is followed for the annotation of the other stochastic timing requirements of the embedded automation system.

Definition of the Quality Model To use the QoS characteristics defined in the first step for the annotation of the system UML models, it is necessary to assign actual values to the parameters of the QoS characteristic template classes: this is carried out through the definition of QoS characteristic bound classes and of template bindings. The UML model containing the binding information and the bound classes is called *Quality Model* that is specifically designed for the system under study. The Quality Model for the running example is depicted in Figure 5.

Note that we have created two different bound classes from the **alarm-latency** QoS characteristics, **PA4ASalarm-latency-voter** and **PA4ASalarm-latency-wd**, through the

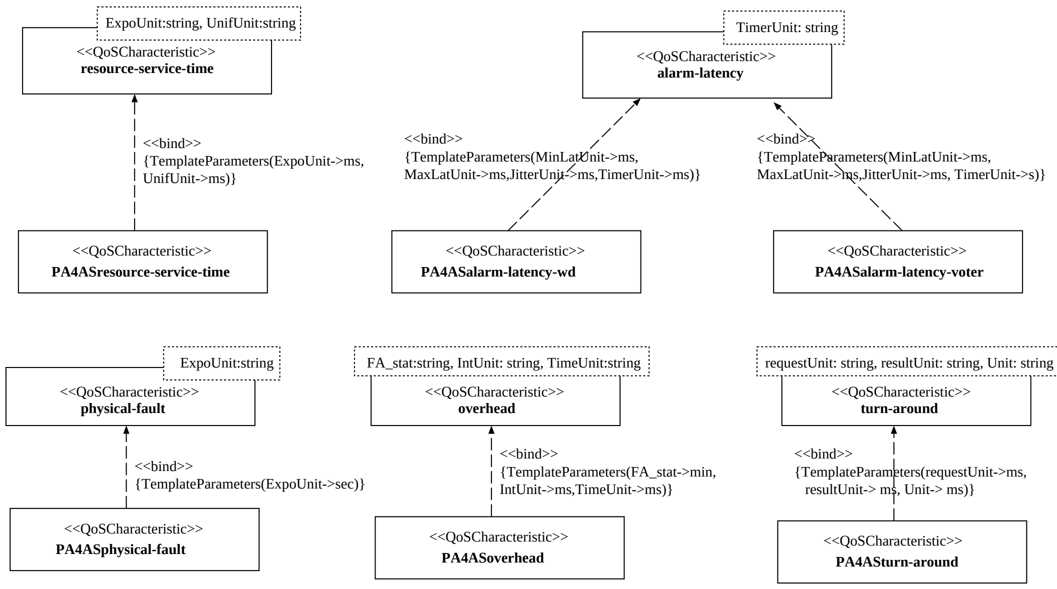
ExpoUnit:string, UnifUnit:string
<<QoSCharacteristic>>
resource-service-time
<<bind>>
{TemplateParameters(ExpoUnit->ms,
UnifUnit->ms)}
<<QoSCharacteristic>>
PA4ASresource-service-time
TimerUnit: string
<<QoSCharacteristic>>
alarm-latency
<<bind>>
{TemplateParameters(MinLatUnit->ms,
MaxLatUnit->ms,JitterUnit->ms,TimerUnit->ms)}
<<bind>>
{TemplateParameters(MinLatUnit->ms,
MaxLatUnit->ms,JitterUnit->ms, TimerUnit->s)}
<<QoSCharacteristic>>
PA4ASalarm-latency-wd
<<QoSCharacteristic>>
PA4ASalarm-latency-voter
ExpoUnit:string
<<QoSCharacteristic>>
physical-fault
<<bind>>
{TemplateParameters(ExpoUnit->sec)}
<<QoSCharacteristic>>
PA4ASphysical-fault
FA_stat:string, IntUnit: string, TimeUnit:string
<<QoSCharacteristic>>
overhead
<<bind>>
{TemplateParameters(FA_stat->min,
IntUnit->ms,TimeUnit->ms)}
<<QoSCharacteristic>>
PA4ASoverhead
requestUnit: string, resultUnit: string, Unit: string
<<QoSCharacteristic>>
turn-around
<<bind>>
{TemplateParameters(requestUnit->ms,
resultUnit-> ms, Unit-> ms)}
<<QoSCharacteristic>>
PA4ASturn-around

Fig. 5. Definition of the Quality Model for the embedded automation system.

binding of the *TimerUnit* parameter to two different metric units (seconds and milliseconds, respectively). Indeed, such classes will be used to specify, in the third step, the duration of the timer associated to the voter (of order of seconds) and the duration of the timer associated to the watchdog (of order of milliseconds). The other template parameters that appear in the two binding expressions are parameters originally defined in the super-class of **alarm-latency** template class (i.e., the **latency** QoS characteristic): although those parameters are associated to properties that will not be actually used in the annotation of the system model of the running example, we have to associate to them specific metric units.

Note that, due to this step, it is not possible to annotate the system models with input parameters and, hence, we cannot derive parametric performance models from UML system models annotated according to the QoS Profile. This is an important issue that will have to be taken care of in the future.

QoS annotation of the system model Figure 6 shows the Communication diagram of the embedded automation system of Figure 2 enriched with the QoS annotation.

The timing and performability specifications of the embedded automation system are annotated using QoS constraints. The QoS annotation has been carried out by adopting all the three possible approaches described in the QoS Profile, that is:

1. through a note symbol attached to the constrained model element and including the constraint(s) written in OCL, as the note symbol stating the constraints (*QoSrequired*) on the availability and on the cycle time and connected to the automation system.

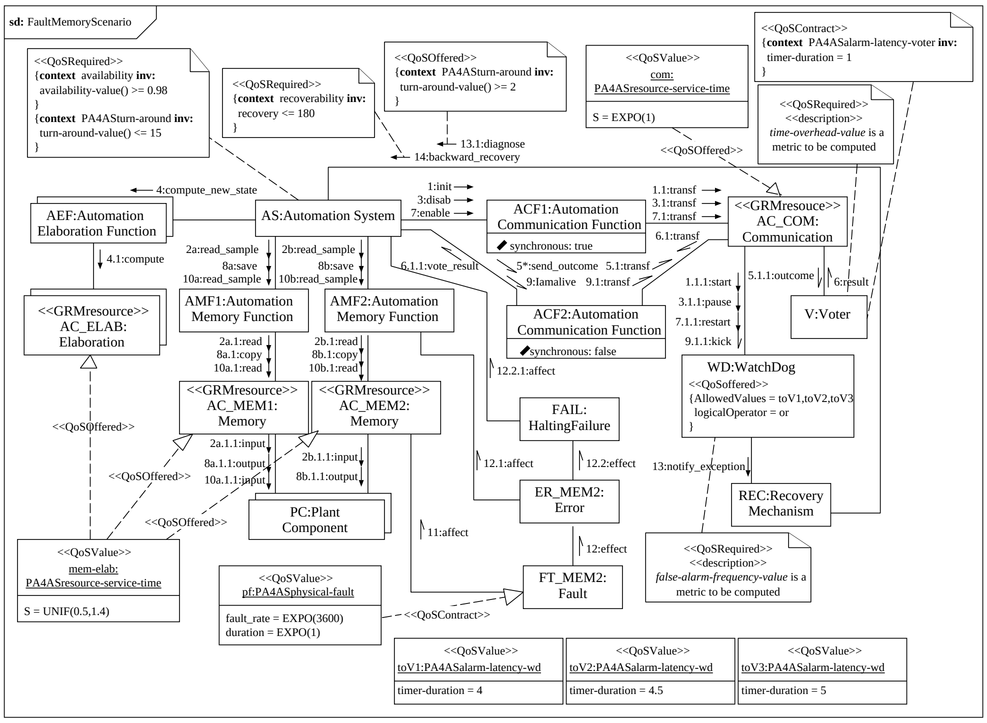

Fig. 6. QoS annotation of the Communication diagram.

2. By connecting the constrained element with a class instance stereotyped as *QoS-Value*. The timing assumptions for the physical fault affecting the memory unit have been specified following this approach: the type of dependency is a constraint of type *QoSContract* stating that the fault occurrence rate and its duration are assumed values.
3. By stereotyping the constrained element with a QoS constraint and by using the *AllowedValue* and the *logicalOperator* properties to reference a set of QoS values and to specify the logical relationship between the referenced QoS values. This approach has been followed for the specification of the timer duration of the watchdog mechanism that is an input parameter of the model (so that the watchdog instance is stereotyped as *QoSOffered*) and that can assume one of the referenced QoS values (i.e., either 4 or 4.5 or 5 milliseconds).

The frequency of false alarms raised by the watchdog and the time overhead caused by the fault masking solution are two metrics to be computed. Unfortunately, as already mentioned in the previous step, the QoS Profile does not support means to specify them as (output) parameters in the system model and, since we have not found a *simple* OCL expression to specify them, the required metrics have been annotated as QoS constraint in a descriptive manner.

The *GRMresource* stereotype allows to identify resources and supports the description of QoS provided by or required from resources. In Figure 6 the memory, elaboration and communication units have been stereotyped as *GRMresource*, being general resources of the embedded automation system, and annotated with the *QoSOffered* constraint in order to specify the service time they can guarantee. The *GRMresource* stereotype is a concrete concept introduced by the QoS Profile as an extension of the original concept described in the General Resource Model of the SPT Profile, where no UML extension are defined for general resources independently from the type of analysis to be carried out.

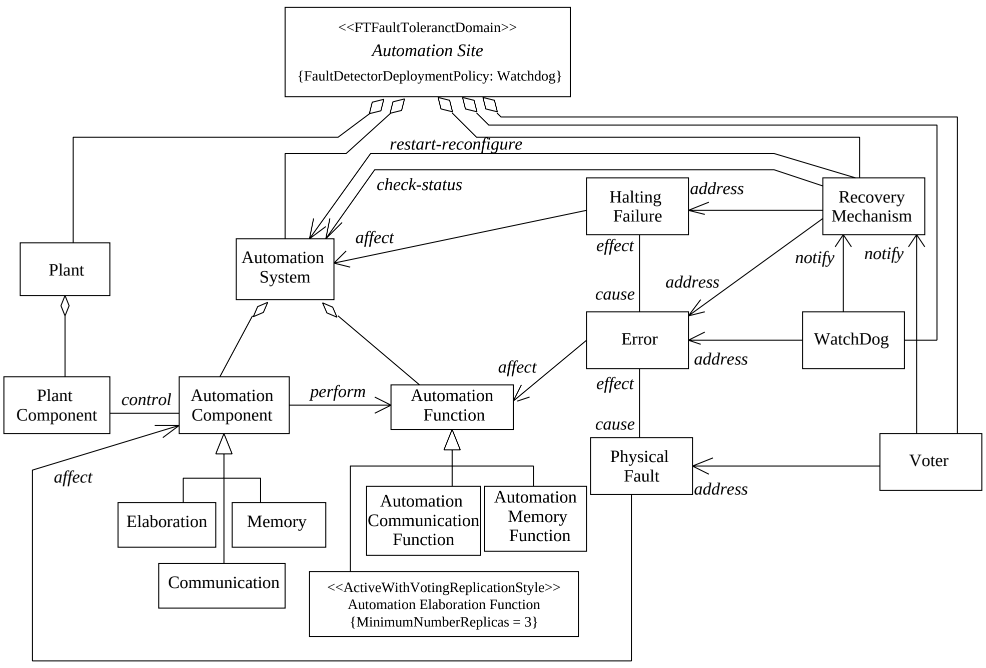

Fig. 7. QoS annotation of the Class Diagram.

Specification of the fault-tolerance architecture The QoS Profile provides UML extensions for the description of fault tolerance architecture, in particular, for the specification of the fault tolerance strategies adopted, for the identification of the groups of software elements that compose redundant blocks and that offer a common service and for the specification of the type of replication adopted. We have applied some of such UML extensions to classes of the Class Diagram of the running example: the resulting annotated Class Diagram is depicted in Figure 7. The automation site has been stereotyped as *FaultToleranceDomain* since it includes both the automation system, together with the controlled plant, and the software mechanisms that implement the fault tolerance strategy.

Automation elaboration functions are the replicated software elements of the example (stereotyped as *ActiveWithVotingReplicationStyle*): they are active replications that execute each elaboration request independently and, finally, a majority voting of results produced by these functions is performed by the voter mechanism. The minimum number of concurrent elaborations required is of 3 replicas.

4 SPT Profile annotations

The annotation approach proposed by the Schedulability, Performance and Time (SPT) Profile is more straightforward but less flexible with respect to the one proposed by the QoS Profile. Indeed, the SPT Profile provides a set of stereotypes and related tags (i.e., "attributes" in UML2.0 and from now on) that can be used directly by the modeler for the annotation of the model elements and for a given type of analysis. There is no possibility of customizing the stereotypes/ attributes provided by the SPT Profile to the specific system domain under study nor to the specific analysis to be performed.

The SPT annotation consists in properly stereotyping the model elements that have to be characterized by quantitative properties and assigning values to the related attributes using the comment-based notation.

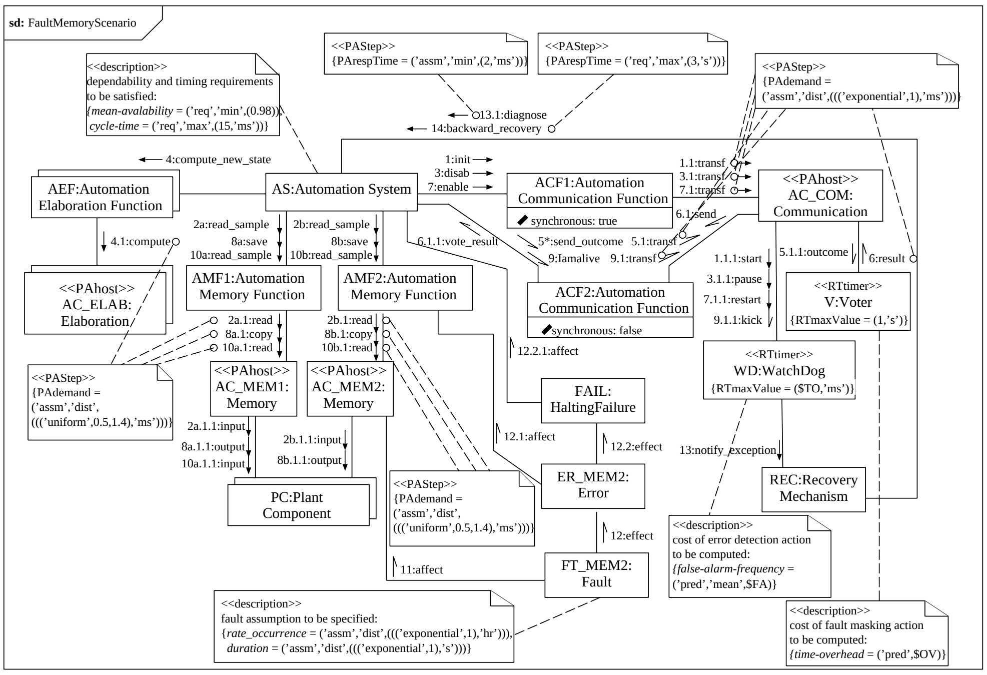

Fig. 8. SPT annotation of the Communication Diagram.

The type of analysis to be carried out on the embedded automation system example is performability and the quantitative characterization of the system includes time, performance and dependability aspects. The SPT Profile supports the annotation of only timing and performance characteristics by means of the General Time Modeling sub-Profile and the Performance Analysis sub-Profile.

Figure 8 shows the Communication diagram of Figure 2 annotated according to the SPT Profile. The concrete concept for modeling timer mechanisms (stereotype *RT-timer*), introduced in the General Time Modeling sub-Profile, has been used for the specification of the maximum duration (attribute *RTmaxValue*) of the timers associated to the watchdog and to the voter mechanisms. The other stereotypes are instead defined in the Performance Analysis sub-Profile and they have been used to identify the processing resources of the system (stereotype *PAhost*) and to specify service times of resources and response times, either assumed or required, associated to the execution of diagnosis and recovery activities (stereotype *PAstep*). Note that, unlike the QoS annotation, the service time of a resource has to be specified, in an indirect manner, as a property of the steps that are hosted by the resource. So that, instead of annotating directly the resource with the specification of its service time, we have to annotate all the messages sent to the resource as *steps* and to associate to them the same value for the execution demand (attribute *PAdemand*). This is a good approach when the steps are characterized by different execution demands but when the execution demand is the same for all the steps the direct annotation is a better solution: the SPT Profile should allow the modeler to use also the direct annotation by adding the service time to the list of properties associated to the resources.

Observe that most of the values assigned to the attributes are strings specifying complex performance values including information on 1) the origin of the value - e.g., either a system requirement (‘req’) or assumed as input to the specification (‘assm’) or a metric to be computed by a performance tool (‘pred’) - 2) its statistical meaning and 3) the time value (that can be a complex time value such as a probability distribution function). The Performance Analysis sub-Profile provides a standard syntax for their specification: the same syntax should be extended to the other sub-profiles, for example, the duration of the timer mechanisms is implicitly an offered QoS characteristic but it could be also a system requirement (a required QoS characteristic).

The SPT Profile allows to specify both input and output parameters in the system model, that are expressed as symbolic variables and prefixed by the dollar ($) symbol: e.g., the timer duration of the watchdog is specified as a parameter by assigning the variable $\$TO$ to the *RTmaxTime* attribute.

In Figure 8, we have used the *description* stereotype to keep track of those requirements, properties or metrics to compute that cannot be specified with the concrete concepts provided by the current SPT Profile, for example, the timeless requirement related to the cycle time of the automation system. Actually, the cycle time is the response time of a sub-scenario of the *Fault Memory Scenario* that starts with the sending of the first message that initializes the watchdog (*1: init*) and terminates with the sending of the signal of life to the watchdog (*9:Iamalive*). The SPT Profile allows to annotate the response time only for the whole scenario or for a single step but not for a subset of steps.

More in general, as already observed in [16], it does not let the user to define a delay measure between an arbitrary pair of events.

The remaining comments stereotyped as *description* are related to dependability aspects of the embedded automation system that cannot be annotated according to the SPT Profile since the latter does not provide support for dependability analysis.

Dependability is *the ability of a system to avoid failures that are more frequent (or more severe) and failure durations that are longer than is acceptable to the users* [8, 9]. Dependability encompasses three main groups of concepts: *threats* (or *impairments*) to dependability, that are undesired events that affect the system dependability (such as faults, errors and failures); *attributes* of dependability, that are the properties characterizing the system dependability (such as availability, reliability, ...); and *means* for dependability, that are sets of methods and techniques used either to prevent fault occurrence (fault prevention) or to guarantee the delivery of correct service despite the presence of faults (fault tolerance) or to reduce the presence of faults (fault removal) or to estimate the present number, the future incidence and the consequences of faults (fault forecasting).

The introduction in the current SPT Profile of a sub-Profile for dependability analysis that provides concrete dependability concepts for *threats*, *attributes* and *means* should be taken in consideration and the QoS dependability characteristics defined in the QoS Profile should be exploited for this purpose.

5 QoS Profile versus SPT Profile

In this section a comparison between the QoS Profile and the SPT Profile is carried out. We will focus on a list of key-points that have emerged from the application of the two annotation approaches on the example of embedded automation system presented in the previous sections.

Type of analysis supported The QoS Profile supports the specification of non-functional requirements and properties of software systems for different types of analysis (e.g., schedulability, performance, dependability). In the Profile, an exemplification of the application of QoS extensions is given for the description of real-time models analyzable with scheduling analysis techniques. In this paper, we have applied the QoS annotation process on an embedded automation system model: a fault tolerance strategy for the system is devised in order to increase its dependability level and performability analysis has to be carried out to assess the timeliness and availability requirements of the system. The Generic Fault Tolerance framework of the QoS Profile provides concrete concepts for the specification of fault tolerant software architectures. In particular, a set of class stereotypes is defined that allow to identify the replicated entities of the system as well as to specify the replication styles and the fault detection solutions adopted for the system. A further annotation step, not applied on the running example of Section 2, consists in the specification of the QoS behavior of software components in terms of their different execution modes, and the associated QoS levels, and of the allowed transitions from one execution mode to another.

The SPT Profile supports the annotation of UML system models either for schedulability or for performance analysis purposes. It should be extended with a Dependability

Analysis sub-Profile that, as the Performance Analysis sub-Profile, imports concepts from the General Time and Resource Models.

Specification level In general, the concrete concepts defined in the QoS Profile can be used for annotating any UML model. Nevertheless, as observed in the exemplification of the QoS annotation approach for schedulability analysis in the QoS Profile, to derive an analyzable model (that is a model on which it is possible to apply directly analysis methods and/or simulation techniques) is necessary to apply the annotation on instance-level models, such as Communication and Deployment diagrams.

SPT extensions can be applied both to instances and to their descriptors, in the latter case, with the meaning that a quantitative property associated to a descriptor element (e.g., a class) is a default value for all its derived instances, which can override it. Moreover, an SPT annotated descriptor-level model has to be interpreted as a special case where there is precisely one instance created for every descriptor.

Annotation approach The QoS annotation is basically a three-step approach: in the first step, a set of QoS characteristics, specific for a system domain and/or for the type of analysis to be performed, is defined through the re-use of some QoS characteristics included in the general QoS catalog. In the second step, a "Quality Model" is defined for the specific case of study in which all the parameters of QoS characteristic template classes are resolved. In the third step, the UML system models are annotated with QoS constraints and QoS values, related to the QoS characteristics defined in the previous steps. The QoS annotation approach is not straightforward from the point of view of a UML modeler, in particular, the execution of the first step. The set of QoS characteristics defined in the first step should be used for the QoS annotation of different applications either of the same domain or for which the same type of analysis has to be carried out. Hence, they should be defined by expert analysts of the application domain and provided in the QoS Profile to the end-user modelers, as done for the set of QoS characteristics for schedulability analysis.

The annotation proposed by the SPT Profile is simpler to apply, from the end-user modeler point of view, since it does not require neither the definition of QoS characteristics that will be used in the annotation of the system model nor the assignment of actual values to model parameters. On the other hand, unlike the QoS Profile, the SPT Profile does not allow to define problem specific measures as function of basic measures.

The SPT annotation approach for performance analysis purposes is exclusively based on scenarios. This scenario-based analysis limits the type of UML models that can be annotated to Communication and Sequence Diagrams and Activity Graphs. As suggested in [10] an "object life" based analysis, that assumes the system specified by a set of StateCharts modeling the behavior of its main components, should be considered as an alternative approach to scenarios.

Annotation in the system model There are three different possible ways of annotating system models according to the QoS Profile: in Figure 6 all the three possibilities have been exemplified. There is no clear criteria to establish which type of annotation is better than the others for the specification of a given QoS constraint; it depends on the modeler as well as on the UML tool capability.

Concerning the SPT annotation, there is a unique way to annotate system models that consists in applying the stereotypes on the proper model elements and in assigning values (or parameters) to the related attributes of interest (for complex values, a syntax for their specification is provided).

Parameterization capabilities The QoS Profile does not allow to annotate system models with parameters: all the parameters of QoS characteristic template classes must be resolved in the Quality Model, for the specific system under study, before their usage. There is the possibility of associating different values to a parameter through the creation of different binds; unfortunately, this is not sufficient since there are cases in which the value(s) of a parameter is unknown and can be derived through the analysis of the system model, as for the performability metrics to be computed (the frequency of false alarms raised by the watchdog and the time overhead due to the fault masking action) for the embedded automation system example.

The SPT Profile, instead, supports the specification of both input and output parameters: by convention, they are expressed as symbolic variables prefixed by a dollar ($) symbol.

Discrimination of the type of specification When annotating a model, the different types of constraints defined in the QoS Profile allow to distinguish between the system requirements (*QoSrequired* constraints), the properties that should be guaranteed by the system or its components (*QoSOffered* constraints), and the agreements between requirements and the quality provided (*QoSContract* constraints). From the analysis point of view, there is no explicit way of differentiating between metrics to be simply computed, system requirements and system assumptions.

In the SPT Profile, performance values are characterized instead by a modifying field that allows one to indicate whether the value represents a) a system requirement, b) a system assumption, c) a metric computed in the analysis and reported back in the UML model or d) a measured value.

Specification of stochastic timings and related issues The QoS Profile does not support the specification of probability distributions. In Section 3, a solution has been proposed that consists in assigning to a *QoS dimension* attribute the return value of a predefined method without the need of introducing, at meta-model level, new properties for the *QoSDimension* stereotype in order to specify the type of distribution and that, actually, are useful only for stochastic-timing specification.

The main drawback of the proposed solution is that in the definition of the QoS characteristic we have to declare as many methods as the types of distributions that will be used in the annotation of the system model. Since a QoS characteristic is defined for an application domain, and not for a specific system or project, an exhaustive list of *standard* predefined methods should be included. This may become a problem when somebody wants to add a new distribution that is not already in the list. In any case, the analysis tool should be able to support all these methods. An example of an (open) list of methods that return a sample value from the most commonly used time-based distributions is given in Table 5.

Method	**Description of the related distribution**
EXPO(*mean*: real): real	Exponential distribution with mean *mean*
UNIF(*a*: real, *b*: real): real	Uniform distribution over the interval [*a*,*b*]
NORMAL(*mean*: real, *s*: real): real	Normal (Gaussian) distribution with mean *mean* and standard deviation *s*
DET(*delay*: real): real	Deterministic distribution with delay equal to *delay*
GAMMA(*k*: integer, *mean*: real): real	Gamma distribution with mean *mean* and integer parameter *k*
WEIBULL(*a*: real, *b*: real): real	Weibull distribution with parameters *a* and it b
ERLANG(*k*: integer, *l*: real): real	Erlang distribution with *k* number of stages and stage rate equal to *l*
...	...

Table 1. List of methods that return a sample value from the related time-based distributions.

Unlike the QoS Profile, the SPT Profile supports the specification of probability distribution functions (PdFs). A general format for specifying time value expressions is described by an extended BNF and includes also a (not exhaustive) list of continuous and discrete PdFs.

From UML models annotated with stochastic timings, according either to the QoS Profile or to the SPT Profile, it is possible to derive analysis stochastic models that are used to carry out V&V activities through the application of analytical methods and/or simulation techniques.

When a stochastic model is characterized by activities whose duration is specified by general distributions (i.e., non-exponential distributions) it is necessary to associate to them *memory policies* that allow to decide, in case of interruption, whether or not to take into account the amount of work carried out from the starting of the activity until its interruption (for example, due to a system resource unavailability). The preemptive memory policies clearly affect the underlying stochastic process of the model and, in consequence, the solution techniques that can be applied on the latter.

In the context of Stochastic Petri Net models [11], three types of preemptive memory policies to be associated to a Petri net timed transition (modeling a system activity) have been defined in the literature [4]. We report them in the following together with an informal explanation of their meaning:

preemptive repeat different (prd): the amount of work done during the execution of the activity is considered lost and when the interrupted activity restarts its execution a new duration is re-sampled from its distribution;

preemptive resume (prs): the amount of work done is not lost and when the activity restarts its execution it recovers from the point it was been interrupted.

preemptive repeat identical (pri): the amount of work done during the execution of the activity is considered lost, as for *prd* policy, but when the interrupted activity restarts its execution its duration is equal to the last sampled value.

Moreover, there may be certain cases in which it is necessary to assign different memory policies to the same activity depending on the type of interrupting event. For example, let us consider a timer mechanism, as the watchdog used for error detection in the embedded automation system of Section 2, in which the timer duration is equal to a random variable distributed according to a deterministic PdF. The timer mechanism can be paused and restarted: when re-activated from the pausing state, the count-down activity resumes from the point it was interrupted so that the current value of the timer is equal to the remaining time to expire. During the count-down activity, the timer mechanism can also also receive an "Iamalive" message from the controlled application and, in this case, its timer has to be reset to its initial value. Both the "pause" event and the "heartbeat" event (the latter, caused by the reception of an "Iamalive" message) interrupt the count-down activity; however, while in case of a "pause" event a *prs* policy should be adopted, in case of an "heartbeat" event a *prd* policy should be applied.

In the QoS Profile it is quite easy to include such new concepts, since the annotation approach allows to define new QoS characteristic through specialization of the ones included in the general QoS catalog. An exemplification is shown in Figure 9(a): the **preemptive-memory-policy** QoS characteristic allows to specify the type of preemptive memory policy to be associated of a timed activity for a (set of) interrupting event(s) by means of two attributes *policy* and *interrupt-event*, respectively. The usage of the newly introduced QoS characteristic is exemplified in Figure 9(b) where a StateChart, representing the behavior of a timer mechanism, is shown. The count-down activity is represented by the do-activity in state *count*; basically two QoS constraints have been associated to the do-activity: a constraint specifies its duration, distributed according to a deterministic distribution, and the other allows to specify the preemptive memory policies adopted as already discussed.

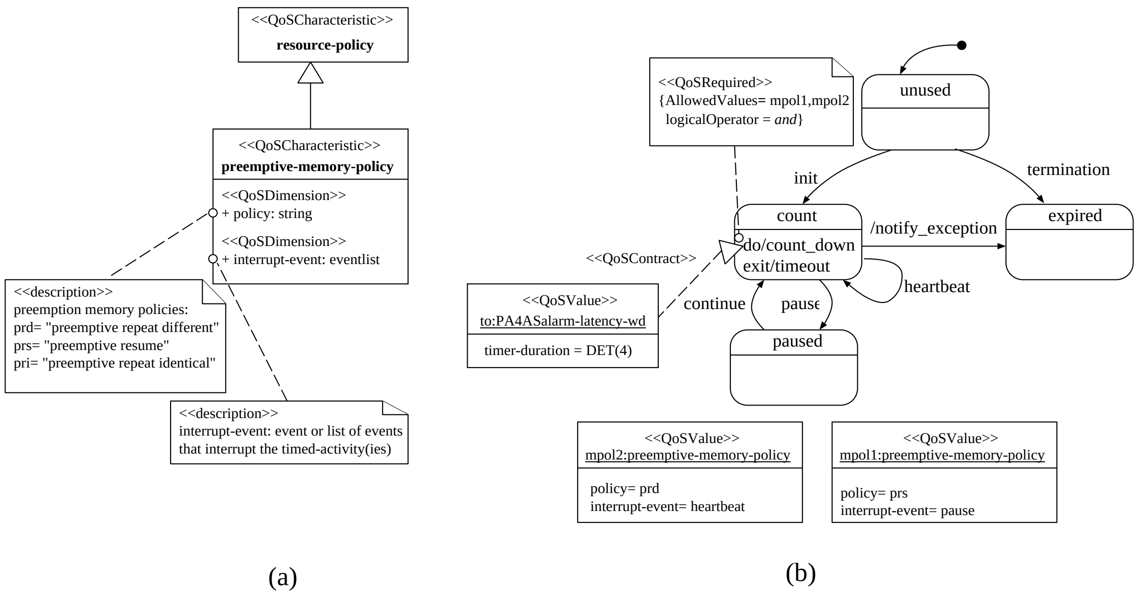

Fig. 9. Definition of the memory policy QoScharacteristic (a) and the QoS annotation on the system model (b).

Basic common concepts The resource concept is common to both the Profiles: in particular, the QoS Profile reuses the meta-models of the General Resource Model defined in the SPT Profile to provide concrete concepts for the specification of the QoS offered by resources.

6 Conclusions

The paper compares the *UML Profile for Schedulability, Performance and Time* (SPT) and the emerging *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* (QoS).

The comparative analysis shows that new concepts are needed in both profiles to express time intervals between two arbitrary events. Also, the two profiles will need to reach a common agreement on the specification of complex timing values, especially of those with stochastic characteristics. Another open problem is the parameterization of models, as in many cases fixed values for model parameters are not enough. The SPT Profile goes a step further by supporting symbolic variables and expressions, but the QoS Profile does not have such a capability yet.

In general, both Profiles struggle with the balance between flexibility (i.e., allow the user to introduce its own definitions) and simplicity/convenience of expression. The challenge when defining a UML profile is to find convenient yet powerful mechanisms of expression for complex concepts, yet to remain within the limits of the UML standard extension mechanisms, which is necessary to insure that the annotated models could be understood by standard UML tool.

Acknowledgement The authors would like to thank Miguel A. de Miguel from Universidad Politécnica de Madrid for his help in clarifying some issues related to the QoS Profile.

References

1. S. Balsamo and M. Marzolla. Simulation Modeling of UML Software Architectures. In *Proceedings of the European Simulation Multiconference*, pages 562–567, Nottingham, UK, June 2003.
2. S. Bernardi and S. Donatelli. Building Petri net scenarios for dependable automation systems. In *Proc. of the* 10*th* *International Workshop on Petri Nets and Performance Models (PNPM2003)*, pages 72–81, Urbana-Champain, Illinois (USA), September 2003. IEEE Computer Society ed.
3. S. Bernardi, S. Donatelli, and J. Merseguer. From UML Sequence Diagrams and Statecharts to analysable Petri Net models. In *Proc. of the* 3*rd* *Workshop on Software and Performance (WOSP'02)*, pages 35–45, Roma, Italy, July 2002. ACM press.
4. A. Bobbio and M. Telek. Combined preemption policies in MRSPN. *Fault Tolerant Systems and Software*, pages 92–98, 1995.
5. Miguel A. de Miguel. General Framework for the description of QoS in UML. In *Proc. of the* 6*th* *International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)*, pages 61–70, Hakodate, Hokkaido (Japan), May 2003. IEEE Computer Society ed.

6. Matthew Hause. ARTISAN Real-time Studio support for Model Driven Architectures. ARTISAN Software Tools whitepaper, 2002.
7. ILogix. The “Rhapsody” CASE tool. http://www.ilogix.com.
8. J. C. Laprie. Dependability – Its attributes, impairments and means. In B. Randell, J.C. Laprie, H. Kopetz, and B. Littlewood, editors, *Predictably Dependable Computing Systems*, pages 3–24. Springer Verlag, 1995.
9. Randell B. Laprie J.C. and Avizienis A. Fundamental Concepts of Dependability. Technical report, LAAS - NewCastle University - UCLA, 2001. LAAS Report no. 01-145, NewCastle University Report no. CS-TR-739, UCLA CSD Report no. 010028.
10. J. Merseguer and J. Campos. Exploring roles for the UML diagrams in Software Performance Engineering. In *Proceedings of the 2003 International Conference on Software Engineering Research and Practice (SERP’03)*, pages 43–47, Las Vegas, Nevada, USA, June 2003. CSREA Press.
11. M.K. Molloy. Performance analysis using Stochastic Petri Nets. *IEEE Transaction on Computers*, 31(9):913–917, September 1982.
12. OMG. UML Profile for Schedulability, Performance, and Time Specification. Version 1.0, formal/03-09-01, September 2003.
13. OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. OMG Adopted Specification, ptc/2004-06-01, June 2004.
14. D.C. Petriu and H. Shen. Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications. In *Proc. of the* 12^{th} *International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, LNCS n. 2324, pages 159–177. Springer Verlag, 2002.
15. D.C. Petriu and C.M. Woodside. Performance Analysis with UML. In B. Selic, L. Lavagno, and G. Martin, editors, *UML for Real: Design of Embedded Real-Time Systems*, pages 221–240. Kluwer Academic Publisher, 2003.
16. M. Woodside and D.C. Petriu. Capabilities of the UML Profile for Schedulability, Performance and Time (SPT). Workshop on the usage of the UML Profile for Scheduling, Performance and Time (SIVOES-SPT), Toronto (Canada), June 2004.