Towards a UML-Based Modeling Standard for Schedulability Analysis of Real-Time Systems

Huáscar Espinoza, Julio Medina*, Hubert Dubois, Sébastien Gérard, and François Terrier
CEA Saclay, DRT/DTSI/SOL/L-LSP, F-91191, Gif-sur-Yvette Cedex, France
{huascar.espinoza, julio.medina, hubert.dubois, sebastien.gerard}@cea.fr

Abstract

This paper presents a generic modeling framework for specifying analyzable UML models of real-time systems. The underlying work was carried out in the context of the OMG initiative for standardizing a UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), which replaces and extends the profile for Schedulability, Performance, and Time Specification (SPT). We focus on some usability and flexibility weaknesses of the SPT modeling framework and describe the improvements brought off in MARTE. The UML extensions include new analysis-specific concepts aimed to support a broader range of quantitative analysis techniques. Additionally, the modeling approach involves a precise language to unambiguously declare and annotate non-functional properties with extended data types, variables, and complex expressions. As illustration, we show how this profile is used in the ACCORD|$_{UML}$ methodology to enable schedulability analysis of annotated UML models.

1. Introduction

The creation of models suitable for predictive and quantitative analysis is a key concern in any methodology for real-time systems development. The tighter the link between the real-time annotations and the modeling artifacts used along the development process, the sooner the verifications for detecting and removing errors can be done, and the easier the design tradeoffs for resources allocation become [11]. Indeed, the ability to evaluate Non-Functional Properties (NFPs) based on mathematically derived results stemming from accurate models is far better than relying exclusively on intuition. The real-time systems community has invested special efforts in incorporating the abilities to describe timeliness properties with enough expressive power, while still preserving the modeling abstraction level used by practitioners.

Like in other software engineering domains, the necessity to raise the level of abstraction, in order to increase designers' productivity and cope with complexity, as well as the facility that it brings to communicate design intents and generate documentation, have encouraged the usage of the Unified Modeling Language (UML) [13] in the real-time domain. As a standard, it adds a number of other well-known advantages that make it a primary modeling language along the software life cycle. Moreover, the Model Driven Development paradigm, and particularly the Model Driven Architecture (MDA) initiative [12] whose modeling support is provided by UML, lead a promising approach for reducing time-to-market and enrich software engineering practices, by moving the development process from lines-of-code to coarser-grained architectural elements.

However, UML per se is somehow still imprecise, and therefore in many cases insufficient for enabling quantitative analysis of real-time systems [2], [3], [14]. The question of how to add real-time modeling capabilities to UML has been addressed in different research and industrial contexts, mostly by the definition of "profiles", which is the built-in mechanism to extend UML to different specialized domains. Thus, for example, the OMEGA [14] profile provides a precise formal semantics dedicated to temporal aspects, UML-MAST [10] considers timing properties and schedulability analysis for distributed real-time systems, and recently a (draft) UML profile for the Avionics Architecture Description Language (AADL) has been proposed [19], which would allow to easily integrate AADL analysis tools with UML CASE tools. Profiles are a powerful extension mechanism; however, most of these profiles suffer a lack of standardization. This is very problematic as the idea of a standard language would get lost.

Standardization provides common semantics and notations, while adopting the best practices for modeling real-time systems. Standards are also a cost effective solution, since interoperability allows using different tools on the same model, and because of the lower overall training costs. Additionally, this helps to manage the risk on the evolution of the tools providers market, reducing the dependency on one single tool. As an industrial standardization board, the Object Management Group (OMG) has been carrying out important efforts to provide

* Post-Doctoral internship. Grupo de Computadores y Tiempo Real, Universidad de Cantabria, Spain.

UML extensions for modeling temporal properties as well as other non-functional aspects of computer systems.

The UML Profile for "Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms" (QoS&FT) [5] makes up part of these efforts. It defines a framework to define and annotate QoS properties in UML models (QoS is here used as a general term to denote NFPs). The QoS&FT profile provides a flexible mechanism to define most common QoS characteristics for different application domains, by means of QoS catalogs. The QoS&FT annotation mechanism is supported on a two-step process which implies the creation of extra objects required just for annotation purposes. This two-step process, however, requires too much effort for the users and may induce not readable models [17]. Moreover, QoS&FT ignores some necessary aspects to model NFP specifications in the real-time domain, such as variables specification, and complex expressions (e.g., time expressions).

The recent SysML language [8], is being considered to be adopted by the OMG as a language to model complex systems, including software, hardware, facilities, and process aspects. Although still insufficient for the real-time domain, SysML provides some useful mechanisms to model this kind of concerns. For instance, the "block" modular unit provides capabilities to specify reusable "model-oriented components", defining a collection of features that includes parameters, constraints, and value specifications qualified by measurement units.

A third OMG standard intended to support non-functional annotations is the UML profile for Schedulability, Performance and Time specification (SPT) [1]. It provided a straightforward annotation mechanism with a minimal set of common annotations to perform very basic quantitative analysis (essentially RMA and queuing theory based analysis). Nevertheless, its structure was not flexible enough to allow for new user-defined annotations or for different analysis techniques. Most experiences of the profile have led to a number of significant suggestions for improvement and consolidation [2, 3] deemed too disruptive for a simple revision. This included also new requirements for specifying both, software, and hardware aspects, MDA compliance for defining separately the platform, the application and allocation models, and modeling of other kinds of NFPs such as power consumption or memory size.

Hence, the OMG called for the development of a new UML profile for the Modeling and Analysis of Real-Time and Embedded systems (MARTE) [4]. The results presented in this paper were performed in the framework of the ProMARTE [20] working team, and comprise some of our proposals integrated in the definition of this new profile [18]. We particularly deal with MARTE modeling capabilities to enable predictive quantitative analyses, namely schedulability. We incorporate in MARTE a number of additional features while reduces the implicit complexity of QoS&FT. MARTE also introduces a precise value specification language that supports symbolic variables and complex expressions for specifying NFPs. But mainly, this framework provides a common modeling framework for different schedulability analysis techniques by factorizing out concepts used in different analysis models.

The paper is organized as follows. In the next section, we present a quick view of the MARTE profile. Section 3 describes its modeling framework to specify NFPs. Section 4 describes the framework for Schedulability Analysis Modeling (SAM). In Section 5, we show the usage of the profile by means of an example in the framework of the ACCORD$|_{UML}$ methodology [7].

2. The UML profile for MARTE

UML uses the metamodeling technique to define its abstract syntax. Metamodels define languages enabling to express models. This means that a metamodel describes the various kinds of contained model elements and the way they are arranged, related and constrained. Hence, a UML model is said to *conform* to its metamodel.

Profiles customize UML for a specific domain or purpose using extension mechanisms able to enrich the semantics and notation of the metamodel elements. A *stereotype* is the basic notion that allows extending UML. A stereotype can be viewed as a subclass of an existing UML concept, which provides the capability of modeling domain-specific concepts or patterns. Stereotypes may have typed properties called *tag definitions*, which may represent attributes or relations with other metamodel elements. Complementary, stereotypes can be also influenced by restrictions expressed in *constraints*.

The MARTE standard comprises both a conceptual *domain model* and the concrete *UML extension profile*. The first one defines key concepts and relationships between them used for describing real-time computing systems. The second one is the specification of how the elements of the domain model are realized by means of stereotypes, and how they extend UML metaclasses.

As Figure 1 shows, MARTE is structured around a core package (TCRM), which describes modeling constructs for time, resources, concurrency, allocation and NFPs, and two main groups of specialized packages that use and refine the core one. One group to model development features of real-time and embedded systems (RTEM), and the other one to annotate real-time models so as to support quantitative analysis (GQAM).

RTEM provides means to define application-modeling patterns, and refined modeling constructs to model hardware and software execution platforms.

On the other hand, the GQAM package provides a generic basis for different quantitative analysis sub-domains. The GQAM package supports two main sub-profiles for *schedulability analysis* to predict whether a set of software tasks meets its timing constraints, and *performance analysis* to determine if a system with non-deterministic behavior can provide adequate performance. Additionally, the profile structure allows for adding further analysis domains, such as power consumption, memory use or reliability. It is the intention to encourage modular sub-profiles for such domains.

Figure 1 also describes some of the main potential actors that may use this specification. Thus, *model designers* are dedicated to define the hardware and software architecture of real-time systems. *Model analysts* are modelers concerned with annotating system models in order to perform specific analysis techniques. Execution *platform providers* are developers and vendors of run-time technologies (hardware- or/and software-based platforms) such as Real-Time CORBA, real-time operating systems and specific hardware components.

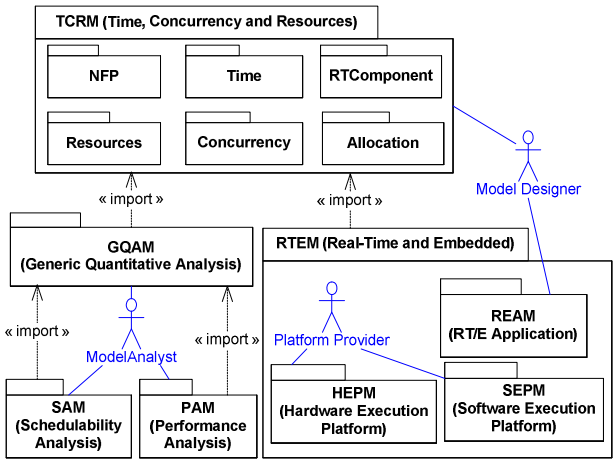

Figure 1. Architecture of the MARTE Profile

In the context of UML-based quantitative analysis, annotating NFPs is of fundamental relevance and implies a number of modeling mechanisms for complexity management in the development cycle, but mainly, for specifying time expressions semantically well formed. The following section describes the NFP modeling package as the basic framework to model quantitative analysis aspects.

3. The NFP Modeling Framework

The model of a computing system describes its architecture and behavior by means of model elements (e.g., resources, resources services, behavior features, logical operations, configurations modes, modeling views), and the properties of those model elements. It is common to group element properties into two categories: functional properties, which are primarily concerned with the purpose of an application (i.e., what it does at run time); and non-functional properties (NFPs), which are more concerned with its fitness for purpose (i.e., how well it does it or it has to do it) [18]. NFPs provides information about different characteristics, as for example throughput, delays, overheads, scheduling policies, correctness, security, memory usage, and so on.

The NFP modeling framework [17][18] is especially focused on formalizing a set of modeling UML constructs to specify non-functional information in a precise way. In fact, one of the common criticisms to the SPT profile is the very superficial semantic information that it provides; in particular, for promoting common understanding of specification and exchange of specifications between different tools. Thus, different tools or analysis techniques could use different metrics for the same concept. For instance, a task *priority* attribute, from a scheduling perspective, is defined in the context of a "priority" scale with either increasing or alternatively decreasing numerical values. Unless we properly qualify NFPs, no common meaning will be adopted, regardless of how detailed and expressive a property name might be. The NFP modeling framework attempts to overcome these kinds of ambiguities.

Figure 2 shows a partial view of the metamodel that supports this framework. This metamodel provides key constructs for modeling this kind of properties at two fundamental stages: declaration and specification. NFP declaration is intended to qualify and assign extended data types to NFP values. NFP specification allows describing values as constants, variables, complex expressions (including time expressions) in a standard textual language.

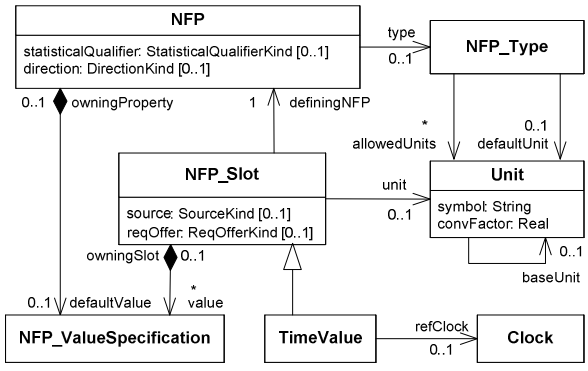

Figure 2. Partial view of the metamodel for NFP

NFPs are qualified by two basic attributes and by a data type (*NFP_Type*). Both attributes of NFPs, *statistical qualifier* and *direction*, have been adopted from the UML profile for QoS&FT [5]. A statistical qualifier indicates the type of "statistical" measure of a given property (e.g., maximum, minimum, range, mean, variance, percentile, distribution). The *direction* attribute (i.e., increasing or decreasing) defines the type of quality order relation in the allowed value domain of NFPs. Indeed, this allows multiple instances of NFP values to be compared with the relation "higher-quality-than" in order to identify what value represents the higher quality or importance. On the other hand, NFP Types add the ability to carry a measurement *unit* for NFP values associated with physical dimension measures. Additionally, we provide the capability of defining new user-specific units in terms of existing base units, through a given conversion factor.

Since a UML viewpoint, NFPs are implemented in MARTE as *tag definitions* qualified and typed as NFPs. MARTE defines a set of NFP Types commonly used in the real-time and embedded system domain. Examples of NFP Types are *Duration*, *Data Transmission Rate*, *Data Size*, *Power*, between others. As illustration, Figure 3 shows the declaration of the Duration NFP type.

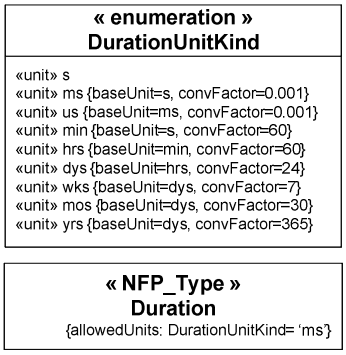

Figure 3. Example of declaration of the Duration NFP type

A model of a system (which is considered to be expressed in UML) can be annotated by specific NFP values, expressing concepts from a given modeling concern or domain viewpoint (for example Schedulability Analysis). An annotated model element describes certain of its non-functional aspects by means of NFP annotations. These annotations are specified by the designer in the models and attached to different model elements. Examples are the *response time* of a *task* when executed, the *utilization* of a *resource*. Thus, a *NFP Slot* specifies that an annotated model element has a value or values for a specific NFP. The values in a slot must conform to the defining NFP of the slot (in type, multiplicity, etc.). NFP Slots include the attributes *source* to qualify different value versions (as for example *required*, *estimated*, or *calculated*), an associated measurement unit (e.g., ms, KB/s, MB) that overwrites the default value declared in the NFP Type definition, and a qualifier for describing the required or offered nature of a NFP value (*reqOffer*).

On the other hand, *NFP Value Specification* defines the textual expressions associated with NFP slots. The NFP modeling framework provides an abstract syntax and a grammar for specifying the values of NFPs. Indeed, while NFPs values are often assumed to be simple values, there are certain cases where it may be necessary to express such values in a more complex way. For example, it may be required for one NFP value to be related in some way to another. This requires both a way of referencing the value of another property as well as the ability to use expressions, such as arithmetic or time expressions. Thus, each value can be specified as a constant, as a variable, as a complex expression value, a special duration or instant expression, or as an interval of value specifications.

Table 1 shows typical examples of the notation for the body of value specifications.

NFP Value Specification	Example
Real Number	`1.2E-3`
Variable	`$timeout`
Ordered Collection	`(1, 2, 5, 88)`
Interval	`[1..251]`
DateTime	`12/01/06 12:00:00`
Duration (between two events)	`($startEvent, $endEvent)`
Duration (number of clock ticks)	`5*t{refClock}`
Constraints	`$deadline < $timeout + 5.0`
Logical Expression	`$V1=(($clients<6)?(exp(6)))`

Table 1. Examples of NFP Value Specifications

In Section 5, we show some examples for specifying NFP values and attaching them to UML elements by means of tagged values.

4. Modeling for Schedulability Analysis

All analysis methods use a simplified/abstract view of the system to analyze, which focuses on those aspects that are relevant to the associated analysis technique. Thus, the analysis modeling concepts rarely map one-for-one to application-level modeling concepts. Particularly, in schedulability analysis, a key abstraction is the notion of a unit of concurrency, representing some execution entity that requires the scheduling services of the system platform. However, application models typically do not show the units of scheduling explicitly. Instead, they are

implied by the presence of model elements such as active objects and asynchronous messages.

The SPT profile already included a basic collection of UML extensions required to define analysis modeling views, particularly for performance and schedulability analyses. However, it was actually stated that they were insufficient for a number of analysis techniques and for certain real-time computing implementations, like distributed systems, or hierarchical scheduling [3]. As a consequence, most UML-based analysis methods did not use/refine the SPT profile as planned, instead they created their own refined (non-standard) extensions [10][14].

To overcome these problems, the SAM framework described in this section defines a collection of extended modeling concepts, as well as a set of generic and extensible/replaceable NFPs for them, oriented to model real-time computing systems from a wider range of schedulability analysis techniques perspective. MARTE provides a minimal set of common annotations for model-based schedulability analysis. This minimal set furnishes enough information to perform common schedulability analyses. However, each vendor is encouraged to supply specialized NFP annotations that extend this set in order to perform model analysis that is more extensive.

4.1 Domain model for schedulability analysis

The Schedulability Analisys Modeling (SAM) domain model is organized as in SPT under the concept of a *Real-Time Situation*. From the predictive point of view schedulability analysis models are intrinsically instance-based. Hence, a real-time situation is still a kind of analysis context that represents a specific situation of the system, working in a particular mode and configuration, and with concrete computational resources. Nevertheless, high-level descriptor-based models can also be established using the RTEM sub-profiles, and then concrete analysis models may be instantiated for specific analyses.

A real-time situation collects the relevant quantitative information required to perform specific analysis. Starting with the real-time situation and its elements, a tool can follow the links of the model to extract the information that it needs to perform the model analysis. A real-time situation is described by separated models associated with three generic modeling concerns (Figure 4):

- *Workload Situation*: a constant load of end-to-end responses triggered by external (e.g., environmental events) or internal (e.g., a timer) stimuli.
- *Behavior Execution*: a description of the executed actions chain as response to the workload, including access to shared resources and their services.
- *Resources Platform*: a concrete architecture of hardware and software computational resources used.

This separation of modeling concerns permits to organize the domain models into comprehensible parts. Moreover, these concerns may act as a set of design sub-views at the user modeling level, thus allowing to reduce changing impact and to facilitate evolution and reusability. However, this is not a mandatory user-model organization. Users may create different views, if desired, with basis on the fundamental SAM concepts.

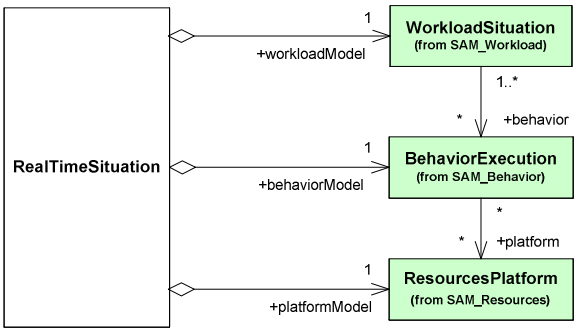

Figure 4. The modeling concerns of a Real Time Situation

4.1.1 The Workload domain model. This model describes the constructs required to specify the computation load on the system and the associated quantitative information about end-to-end stimuli, responses, and temporal requirements.

A *workload situation* of a real-time system is typically defined by the set of stimuli starting computations. In the SAM framework, we refer to an instance of a particular stimulus as an event occurrence. Since the stimulus can occur repeatedly, we refer to recurrence of events as a *trigger*.

A computation that is performed as a consequence of a trigger is referred to as the *response* to its event occurrences. Depending on the implementation nature of responses, they could be concretized in a single task executing in one processor or in dependent tasks into single or multiple processors. We do not include in this model the detailed behavior involving a response (we do it in the Behavior model).

Notice that the *trigger* and *response* concepts specify only end-to-end behaviors, specifically, their “cause” side and their “effect” side. Thus, in order to group these two concepts into a single modeling unit, we adopt the concept of *transaction*. A transaction [[15]] refers to the entire cause-effect end-to-end behavior describing a separate computation in the system. Hence, the set of particular transactions defining one load scenario for analysis purposes composes a single workload situation. A workload situation may correspond to a mode of system operation (e.g. starting mode, fault recovering, or normal operation) or a level of intensity of environment events.

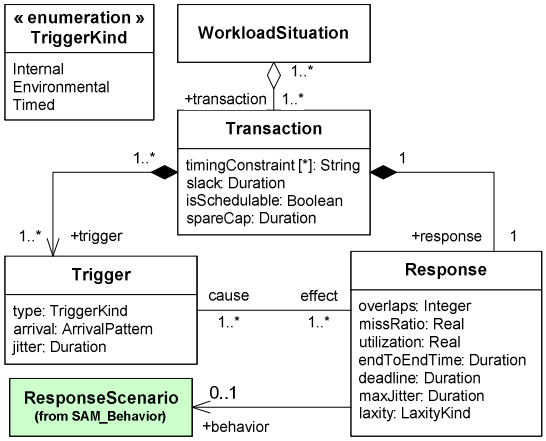

Figure 5. The Workload domain model

Additionally, the transaction, trigger and response concepts have a set of NFPs used for performing quantitative analysis. We define NFPs as tag definitions disambiguated by a statistical qualifier and a direction attribute (not shown here for space reasons). Different analysis techniques commonly use dissimilar NFPs. However, the SAM model defines a pre-declared set of NFPs useful for most of the analysis techniques.

For instance, a trigger is characterized by its *arrival pattern*, which is a structured data type containing concrete attributes, as for example *arrival kind*, *period*, *minimum arrival time*, *distribution function*, among others. On the other hand, responses specify a set of latency NFPs concretized by end-to-end delays or temporal requirements, e.g., *end-to-end time, deadlines*. Transactions may be also annotated with *efficiency* NFPs, as for example *slack* or *spare capacity* obtained from schedulability analysis tools. Timing constraints of transactions are expressed in the form of NFP constraints, which relates to observers (implemented by NFP variables) that must be attached at some point in the sequences of actions that describe the response scenario.

4.1.2 The Behavior domain model. Figure 6 summarizes the domain concepts for defining Behavior modeling aspects. In this model, the *behavior execution* concept serves to collect detailed descriptions of the behavior of responses, which are called in turn *response scenarios*. By response scenario we mean the specification of the smaller segment of code execution and their precedence and concurrence relationships. This modeling aspect is core for quantitative analysis in order to evaluate how the execution segments contend for use of the platform resources from a timing viewpoint.

In this manner, the detailed behavior of a given response is represented by an ordered series of step executions called scheduling actions *(SAction)*. Considering a holistic approach for the analysis, an RT action may represent the time it takes a piece of code execution as well as the sending of a message through a network. The ordering of SActions follows a predecessor-successor pattern, with the possibility of multiple concurrent successors and predecessors, stemming from concurrent thread joins and forks respectively. The granularity of a SAction is often a modeling choice that depends on the level of detail that is being considered. Hence, a SAction at one level of abstraction may be decomposed further into a set of finer-grained SActions. Response Scenarios use resource services for execution through the allocation of SActions to *schedulable entities*, and for other platform services, through calls to *shared resources* (*acquire* and *release* actions).

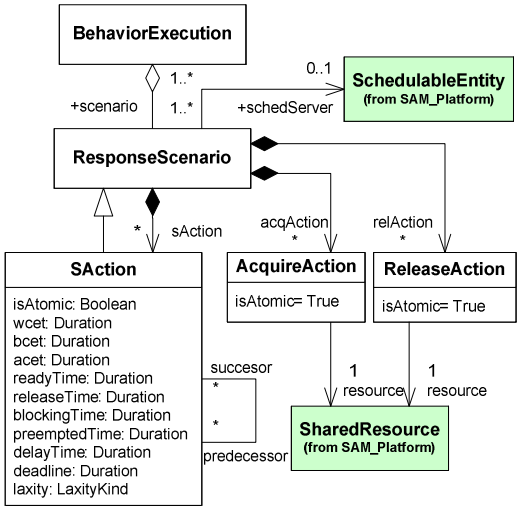

Figure 6. The Behavior domain model

The SAction concept is consistent with SPT, and hence it is characterized by a similar set of NFPs, enriched with some other and extensible latency properties like best and average case execution time.

4.1.3 The Platform domain model. In the SAM framework, the concept of *resources platform* matches to the model of engineering resources introduced in the SPT profile [[1]], [[6]]. This includes not only hardware resources (CPU, devices, backplane buses, network resources), but also software ones (threads, tasks, buffers). Figure 7 shows a framework to describe the platform of resources.

We shape an abstracted version of a more structured and detailed platform model (software and hardware execution MARTE platform sub-profiles), which is especially useful for expressing NFPs oriented to

schedulability analysis and without distinguishing among different abstraction levels (hardware, RTOS or middleware).

The platform model consists of a set of resources with explicit NFPs. This model distinguishes two kinds of processing engines: *execution engines* (e.g., processors, coprocessors) and *communication engines* (e.g. networks, buses). For each, the SAM framework assigns generic NFPs. Specifically, *throughput* properties e.g., *processing rate* or *transmission rate*, *efficiency* properties e.g., *utilization*, *slack*, and *overhead* properties as for example *blocking times*, *interrupt overhead times*.

Schedulable entity is a kind of active protected resource that is used to execute SActions or complete Response Scenarios. In a RTOS this is the mechanism that represents a unit of concurrent execution, such as a task, a process, or a thread. In a network, it represents a channel or message management unit that can be characterized by concrete scheduling parameters (like the priority for a CAN bus).

Processing engines own *shared resources* as for example I/O devices, DMA channels, critical sections or network adapters. Shared resources are dynamically allocated to schedulable entities by means of an access policy. Common access policies are *FIFO*, *priority ceiling protocol*, *highest locker*, *priority queue*, and *priority inheritance protocol*.

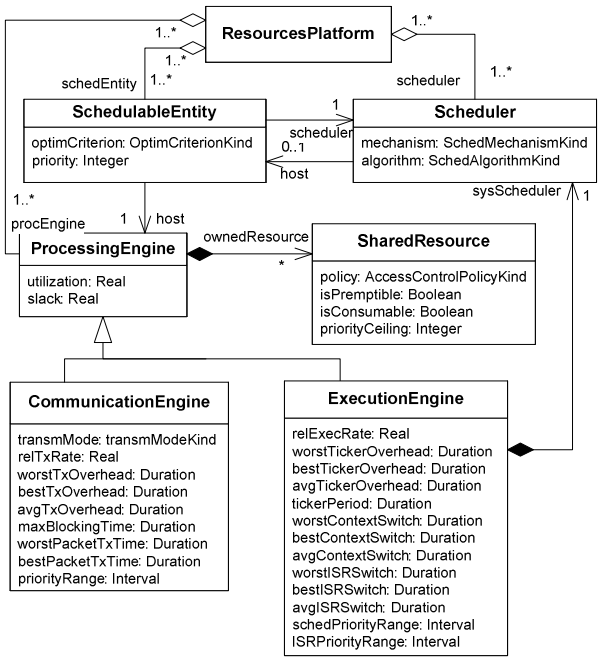

Figure 7. The Resources domain model

Schedulers can play two roles in this model, *system schedulers* (typically a RTOS scheduler) that offer the whole processing capacity of its associated base processors to its allocated schedulable entities, and other *secondary schedulers* that only provide the processing capacity offered by its hosting schedulable entity. This hierarchical structure is typically used in RTS when users are interested in applying dynamic scheduling on top of commercial RTOS supporting only static scheduling. Likewise, novel algorithms exist that make possible to perform real-time analysis of these hierarchical configurations of schedulers [16].

5. Using the SAM sub-profile

We now examine how the domain concepts previously presented can be represented (mapped) in the UML modeling space. The annotations have been made over a case study application for the real-time modeling and analysis of a simple distributed system for the teleoperated control of a robotized cell [10]. This example has been reformulated in the context of the ACCORD|$_{UML}$ methodology for developing real-time embedded systems. ACCORD|$_{UML}$ consists of a full MDA development process (and the underlying modeling and execution platforms) covering from requirements modeling, early quantitative analysis, to full implementation.

The application system is composed of two processors interconnected through a *CAN bus*. The first processor is a teleoperation station (*Station*); it hosts a GUI application, where the operator commands the robot and where information about the system status is displayed. The second processor (*Controller*) is an embedded microprocessor that implements the controller of the robot servos and its associated instrumentation.

The software architecture is described by means of the class diagram shown in Figure 8. The software of the Controller processor contains three active classes (called real time object –RTO in ACCORD|$_{UML}$) and a passive one which is used by the active classes to communicate. *Servo Controller* is a periodic RTO that is triggered by a ticker timer with a period of 5 ms. The *Reporter* RTO periodically acquires, and then notifies about, the status of the sensors. Its period is 100 ms. The *Command Manager* RTO is aperiodic and is activated by the arrival of a command message from the CAN bus.

The software of processor Station has the typical architecture of a GUI application. The Command Interpreter RTO handles the events that are generated by the operator using the GUI control elements. The Display Refresher RTO updates the GUI data by interpreting the status messages that it receives through the CAN bus. Display_Data is a protected object that provides the embodied data to the RTO in a safe way. Both processors

have a specific communication software library and a background task for managing the communication protocol.

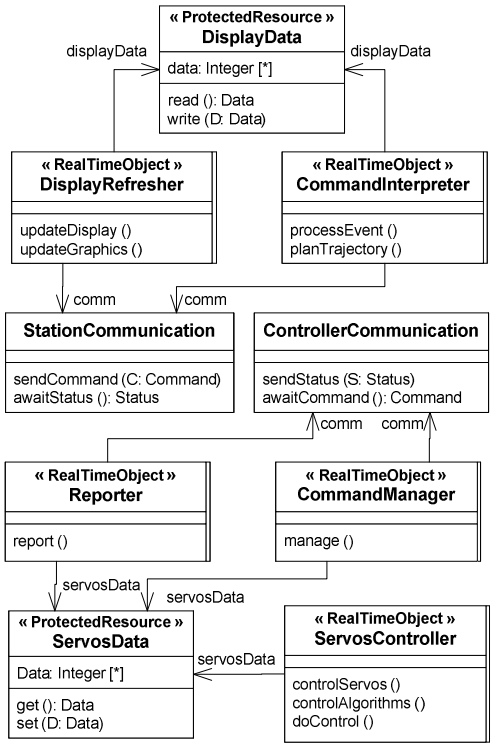

Figure 8. Software architecture of the Teleoperated Robot

To organize the ACCORD|$_{UML}$ models annotated for schedulability analysis, we adopt the concept of *views*, which represent the concern models composing the SAM's analysis context concept. In this way, we provide separated diagrams for specifying the SAM concepts of *workload situation*, *behavior execution*, and *resources platform*. Next, we show some examples that illustrate this organization.

5.1 Example of Workload Situation Model

The RT_Situation to be analyzed contains three transactions with hard real-time requirements. They all use the processing resources Station, Controller and CAN_Bus and interact by accessing protected objects.

The *Control Servos* transaction executes the Control response with a period and a deadline of 5 ms. The *Report Process* transaction transfers the sensors and servos status data across the CAN bus, to refresh the display with a period and deadline of 100 ms. Finally, the *Execute Command* transaction has a sporadic triggering pattern, but its inter-arrival time between events is bounded to 1 s.

Figure 9 shapes a UML Interaction Overview Diagram (IOD) for the Teleoperated Robot example. This activity diagram represents a *workload situation* model consisting of the three above-mentioned transactions characterized by their *triggers* and *responses*. These three *transactions* explicitly introduce the semantic of concurrency for the *activity partitions*. *Triggers* introduce the semantic of event sequence arrivals for the execution of each *interaction* invocation. We also include some NFP annotations for trigger and responses.

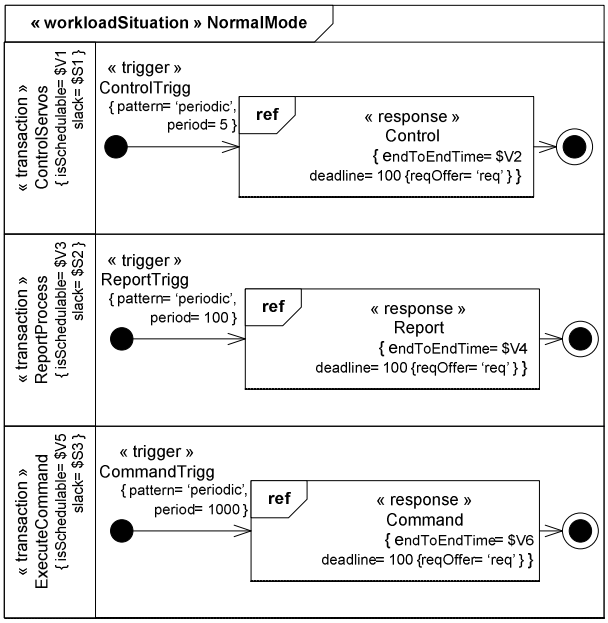

Figure 9. Example of Workload Situation model

5.2 Example of Behavior Execution Model

The *Behavior Execution* concept extends the UML metaclass *Interaction,* which can be modeled by a set of sequence or communication diagrams in UML2. In our example, we applied it to sequence diagrams. Thus, *triggers* extend the metaclass *message*. *SActions* extend the UML2 concept of *execution specification*. Finally, *shared resources* are *lifelines* of the sequence diagrams. The chain of actions (connected by the successor-predecessor patterns) symbolizes the concept of *response scenario*.

In Figure 10, we present one of the three scenarios for the Report transaction. This scenario represents the behavior description of a *response scenario*. Thus, this response scenario is completed with real-time constraints that should be checked in the further schedulability analysis.

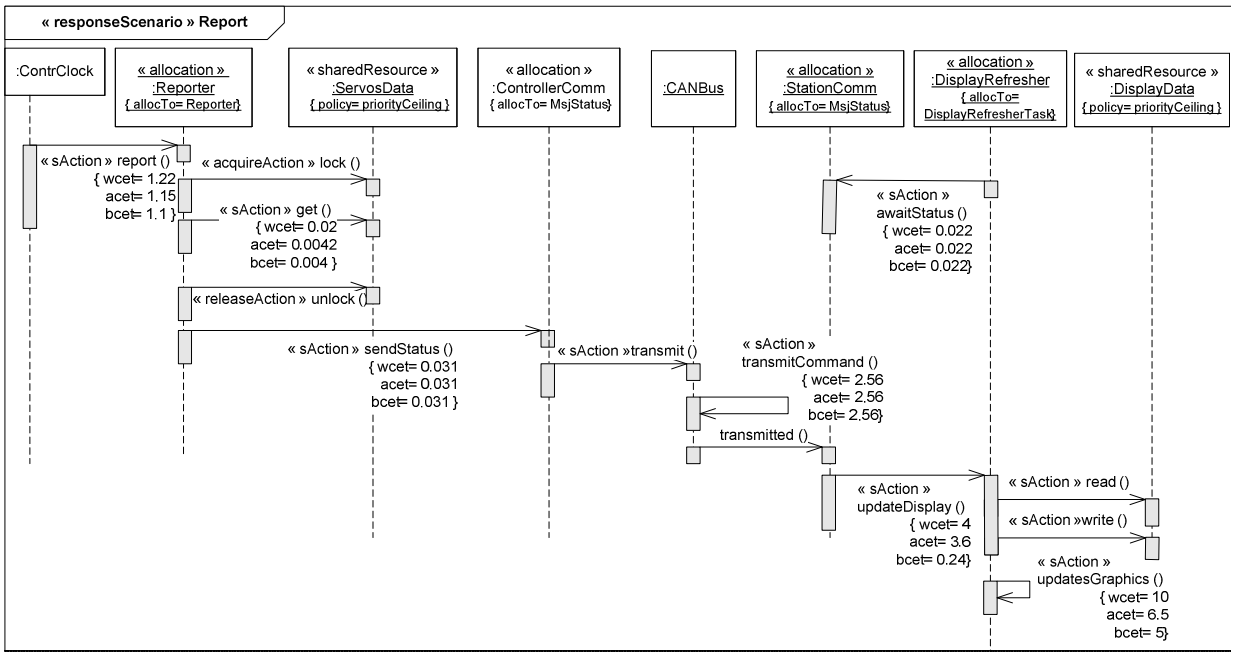

Figure 10. Example of Behavior Execution model

5.3 Example of Resources Platform Model

In Figure 11, the "Execution Engine" domain concept extends the metaclass *nodes*, and the metaclass *Instance Specifications* are used to represent schedulable entities, shared resources, and eventually schedulers.

5.4 Some Schedulability Analysis Results

Table 2 shows the most relevant results obtained from the MAST schedulability analysis tools [10]. In this table, we have compared the end-to-end times of each of the three transactions of the Real Time Situation with their associated responses.

Transaction/Response	Slack	EndToEndTime	Deadline
Control_Servos	101.56%		
Control		3.05 ms	5 ms
Report_Process	189.84%		
Report		39.1 ms	100 ms
Execute_Command	186.72%		
Command		359 ms	1000 ms

Table 2. Results of Schedulability Analysis with the MAST tool

In order to get a better estimation of how close the system is from being schedulable (or not schedulable), the MAST toolset is capable of providing the transaction and system slacks. These are the percentages by which the execution times of the operations in a transaction can be increased yet keeping the system schedulable.

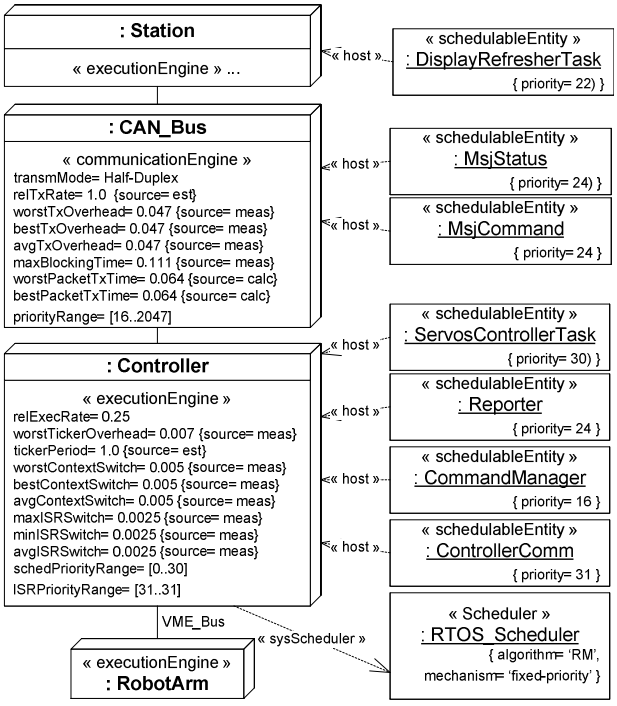

Figure 11. Example of Resources Platform model

Notice that most analysis tools operate on a simplified view of a system, as was illustrated in this example. However, this profile allows annotations and interpretations to be attached at the level of detail desired by the designer. Indeed, even if the specification contains extreme detail, the annotations may optionally be applied to aggregates. This is an overriding reason to find a path to annotations that require a minimum of effort, with a minimum of additions to the design model, and with clear, non-fragmented specifications of NFPs. It is also essential that NFPs can be attached to a real software design, rather than requiring a special version of a design created only for analysis.

6. Conclusions

This paper describes the MARTE schedulability analysis sub-profile for enabling timing predictions. One of the main goals behind this sub-profile is to provide a common framework within UML that fully encompasses the most common schedulability analysis techniques but still leaves enough flexibility for different specializations.

The approach is supported on the NFP modeling framework. It defines a set of mechanisms to declare and specify NFPs that are necessary for different kinds of quantitative analyses. The relationships between NFPs annotations and UML modeling elements is discussed in order to show how to declare domain-specific NFPs, and how to express NFP values attached to model elements.

The schedulability analysis modeling (SAM) framework provides a common modeling basis for different analysis techniques by factorizing concepts that are used by the various schedulability methods. This framework extends the previous SPT's schedulability sub-profile and reorganizes it into generic and consistent modeling concerns (workload, behavior and platform). SAM attempts to enhance the expressive power of UML models and the profile usability by easing the comprehension of the global framework.

The usage of NFP and SAM frameworks are illustrated by the utilization of the proposed annotations on a typical distributed real-time application, formulated in the frame of the ACCORD$|_{UML}$ methodology. This work can be seen as a first reflection of the UML MARTE profile's schedulability analysis capabilities.

7. References

[1] Object Management Group: UML Profile for Schedulability, Performance, and Time, Version 1.1. 2005. OMG document: formal/05-01-02.

[2] Object Management Group: Pending Issues sent to the OMG Finalization Task Force: UML Schedulability, Performance and Time profile.

[3] S. Gérard (edited by): Report on SIVOES'2004-SPT Workshop on the usage of the UML profile for Scheduling, Performance and Time Mai 25th, 2004, Toronto, Canada.

[4] Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), RFP. 2005. OMG document: realtime/05-02-06.

[5] Object Management Group: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. 2004. OMG document ptc/04-09-01.

[6] B. Selic, A Generic Framework for Modeling Resources with UML. IEEE Computer, Vol.33, N. 6, pp. 64-69. June, 2000.

[7] S. Gérard: "Modélisation UML exécutable pour les systèmes embarqués de l'automobile", PhD Thesis. 2000, Evry, Paris.

[8] Object Management Group: Systems Modeling Language (SysML) Specification, Version 0.9. Draft. 2005.

[9] T. H. Phan: "Analyse d'ordonnançabilité d'applications temps réel modélisées en UML", PhD Thesis. 2004, Evry, Paris.

[10] J.L. Medina, M. G. Harbour, and J.M. Drake: MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems. Proc. of the 22th IEEE Real-Time Systems Symposium, pp. 245-256. December 2001.

[11] R. Chen, M. Sgroi, G. Martin, L. Lavagno, A. L. Sangiovanni-Vincentelli, J. Rabaey: UML for Real: Design of Embedded Real-Time Systems, Edited by B. Selic, L. Lavagno, G. Martin, pp. 189-270, Kluwer Academic Publishers, May 2003.

[12] Object Management Group. MDA Guide Version 1.0.1. 2003.

[13] Object Management Group. Unified Modelling Language: Superstructure Version 2.0. 2004. OMG document ptc/04-10-02.

[14] S. Graf, I. Ober, I. Ober: A real-time profile for UML. STTT, Int. Journal on Software Tools for Technology Transfer Springer Verlag. 2004.

[15] K. Tindell: Adding Time-Offsets to Schedulability Analysis: Technical Report YCS 221, Department of Computer Science, University of York, January 1994.

[16] Sha, L., Abdelzaher, T., Arzen, K., E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A., K.: Real Time Scheduling Theory: A Historical Perspective: Real-Time Systems Journal, Vol. 28, No, 2-3, pp. 101-155, ISSN:0922-6443, November-December 2004.

[17] H. Espinoza, H. Dubois, S. Gerard, J. Medina, D.C. Petriu, M. Woodside, "Annotating UML Models with Non-Functional Properties for Quantitative Analysis", Proc. of MoDELS'2005 Satellite Events, Lecture Notes in Computer Science, Springer, 2005.

[18] Object Management Group: UML Profile for Modelling and Analysis of Real-Time and Embedded systems (MARTE), Initial Submission: ProMARTE team. 2005. OMG document: realtime/05-11-01.

[19] E. Colbert, "Overview of the UML Profile for the SAE AADL" (presentation): http://la.sei.cmu.edu/aadlinfosite/AADLPublications&Presentations.html, SAE World Aviation Congress Nov 2004.

[20] The ProMARTE home page: http://www.promarte.org