A UML Profile and a Methodology for Real-Time Systems Design*

Antonia Bertolino, Guglielmo De Angelis
ISTI – CNR, Pisa, Italy
{antonia.bertolino, guglielmo.deangelis}@isti.cnr.it

Cesare Bartolini, Giuseppe Lipari
Scuola Superiore Sant'Anna, Pisa, Italy
{cbartolini, lipari}@sssup.it

Abstract

Modern real-time systems are increasingly complex and pervasive. Model Driven Engineering is the emerging approach for the design of complex systems, strongly based on the usage of abstract models as key artifacts, from which an implementation is derived through a series of well-defined (automated) transformations. The widely adopted input notation in MDE is the Unified Modelling Language. To express models in a particular domain, and notably for the modelling of real-time embedded systems, UML profiles have been proposed, which enrich the set of UML native elements with a consistent set of extensions. In this trend, this paper develops an approach to the design of real-time systems, based on a UML profile which is obtained from the OMG standard SPT-Profile, with a few necessary modifications. The profile also incorporates a methodology for automatically mapping the UML model into a task set with periods and deadlines, in line with the MDE philosophy. An illustrative example is provided.

1 Introduction

Embedded real-time systems are used in many systems, from safety critical automotive applications to entertaining systems.

The complexity of embedded real-time software systems, together with the cost pressure for such high volume products, has convinced designers to evaluate and adopt appropriate design and analysis tools. The required features of such tools include early assessment of performance, exploration of the space of parameters, code generation, formal analysis of correctness, schedulability analysis, testing, and much more.

Unfortunately, no commercial tool is currently able to support the designer at all stages of the development. Also, different tools that are able to support specific phases of the design, are not able to interoperate for the lack of an assessed common methodology.

In this paper, we focus on a problem that has been addressed only recently by the real-time research community: given a functional specification and its real-time contraints, how to design the set of concurrent real-time tasks and assign functional behaviour to the tasks. This problem is commonly referred as the problem of *mapping* the functional behaviour into an architectural specification, i.e. a set of concurrent tasks, their parameters and synchronization mechanisms.

Early efforts in this direction have already been made by some tool providers. For example, the Real-Time Workshop tool of the Matlab/Simulink toolset can generate a real-time implementation of a control system schematic, specified in a graphical language. The implementation consists of one or more periodic real-time tasks, and their priority. However, the tool has very little flexibility. On one hand, it does not allows to set real-time constraints in the graphical system specification. Also, to maintain consistency between simulation and implementation, in some cases the tool forces the introduction of unwanted communication delays between the functional blocks, modifying the original design specification.

To solve this type of problems, from many sides, several attempts are being made to incorporate already at the modeling stage appropriate notations for expressing the desired non-functional properties, in a way that can be subsequently analysed and transformed, via successive refinements, into a conforming implementation. The idea of using and annotating models as a first class element in software development is the philosophy behind the emerging Model Driven Engineering (MDE) approach [5, 6, 18].

MDE is a breach in software engineering concerning application modeling and implementation. In this paradigm, models provide abstractions of a physical system that allow engineers to reason about that system by ignoring extraneous details while focusing on the relevant ones [6]. This is not actually different from traditionally recommended practice in any software development process since the earliest days of programming. The main difference between

*This work is suported by Ericsson Lab Italy in the framework of the PISATel initiative. Web Site: http://www.iei.pi.cnr.it/ERI/

usual practice and the MDE stays in the fact that in the new vision models are not only used for documentation purposes, but they become well defined input/output elements for computer-based tools that implement precise operations [5].

Hence, models used for reasoning and analysis are then directly transformed into the desired implementation. This usage of models as a baseline for implementation permits to rise the abstraction level at which solutions are described, and opens a realm of novel possibilities for creating, analyzing, and manipulating systems through various types of tools [18]. However, not to deceive these expectations, the customary attitude to system development based on the usage of individual tools that tend to only operate on specific models conforming to their own internal format has to change in favour of standard techniques and languages [18].

Currently, the standardization process of the MDE is strongly oriented towards the Unified Modeling Language (UML) as the primary modeling notation [6]. UML provides a variety of instruments to describe the characteristics of a generic system in corresponding models. However, it is not complete, in the sense that the basic elements of the language cannot cover all potential needs for describing specific systems from any domain. Hence in some cases the definition of domain-specific variants of the UML may be required. The UML however has already been conceived for extensibility, for which purpose it provides a built-in *profiling* mechanism to extend the language with elements and constructs apt to describe specialized features, though remaining compliant with its standard definition.

One domain particularly sensible to the exigence of profiling the UML is real-time embedded system design, which is the scope of our research. Several proposals have been launched of UML profiles for dealing with non-functional aspects of systems (we provide an overview in Section 2.2). Most of those approaches however address the extensions of the UML modeling elements, but do not also directly incorporate engines to analyse the models for RT design purposes. From this point of view, the contribution of this paper can be seen in two different perspectives: in the long term, our goal is that of unifying a real-time design methodology with the MDE approach. In this direction, this paper provides a fully developed exercise of how such goal can be pursued: we take an existing mapping and scheduling method, "import" it within a compliant UML profile, and show how a task set can be obtained directly from the UML model through a series of automated steps. In the short term, the methodology resulting from this exercise already provides by itself a useful, applicable solution to fit an embedded application, described by means of fully compliant UML diagrams, into a set of real-time tasks, ready to be executed on a real-time scheduler.

The paper is structured as follows: in the next section we provide the background of related work, on which the methodology is built, and in Section 3 we summarise the scheduling algorithm considered. Then, in Section 4 our UML profile is defined: this is derived from the well known OMG UML Profile for Schedulability, Performance and Time [13]. The methodology is hence illustrated by means of an example in Section 5 and discussed in Section 6. Conclusions are finally drawn.

2 Related Work

2.1 RT Methodologies

The vast majority of research work on real-time systems is centered around the concept of task. Real-time theory does not focus on the problem of generating the task set and assigning non-functional properties to tasks. Generally, task sets are assumed to be given by the designer, using some ad-hoc software design methodology.

However, there is the lack for a well-established methodology that guides the designer in generating the task set. Instead, in the industrial practice some experienced engineer uses heuristics for establishing the tasks set. In many cases, the task set and the parameters of the tasks are decided well before the complete design of the functional behavior of the application is completed. Functional blocks are then assigned to tasks depending on their requirements in terms of criticality or importance. This decision makes the design unflexible: as new funtionality is added, it would probably be better to re-design or even completely re-generate the task set and its parameters.

Most system modeling approaches are based on this structure. Even development tools focusing on platform-based design assume tasks to be manually generated by the designer. Manual generation makes the methodology very flexible, as it allows the designer to quickly change the mapping and test the functionality and performance of many configurations rapidly. However, in complex applications, the number of possible mapping configurations is very large. Therefore, the designer needs a tool that aids him/her in automatically exploring the space of possible configurations for identifying the one that leads to the best performance.

An example of such tools is the Cierto VCC by Cadence. The idea is to implement the platform-based design paradigm [16], separating the definition of the behavior and architecture models, and then mapping the first on the second. Unfortunately, the mapping phase must be done manually, and no aid is given to the designer. In addition, it is not possible to specify real-time constraints in the behavior model.

Similarly, the Metropolis environment [2], which is being developed in joint-work between several research cen-

ters and inherits several of the objectives of VCC while making an effort at reducing its limitations, does not solve the problem we are trying to address, since it focuses primarily on providing a general way of describing the different aspects of a system through a metamodel.

Some work by Gerber and Saksena [9] aims at designing a distributed system which satisfies performance requirements. The goal of their methodology is to assign the task parameters (i.e., periods and priorities), and the communication resources, so to minimize a global cost function that takes into account throughput and delay.

However, they assume that the task set has already been generated. Moreover, they assume that tasks have already been assigned to nodes in the distributed system. Although this work can serve as a basis for future extensions of our methodology, it has a slightly different scope.

Most closely related to our work is the more recent work by Saksena and Karvelas [14, 15]. In their work, they try to define a methodology to group jobs (similar to what we call functional blocks in Section 3.1) into tasks and assign each task a scheduling priority. A great contribution of their work is the use of object-oriented design methodologies: jobs are not self-contained functions, but object methods. However, this proposal comes with two problems: first, the task creation algorithm is not made available by the authors; in addition, their methodology is based on static priorities.

Recently, Bartolini et al. [3] have proposed an algorithm that, starting from a functional model of the system and the associated timing constraints, generates the task sets by assigning functional blocks to tasks, and periods and deadlines to tasks. The algorithm assumes an EDF scheduler, and has been proved to be optimal for single processor systems [4]. The approaches in [3] and in [15] have been compared in [4]: the results obtained with the methodology in [3] were superior from a schedulability point of view, mainly from the use of dynamic priority assignments. However, the functional specification symbols used in [3] are not standard, and the algorithm is not part of any methodology.

This is actually the goal of this paper: to use a standard MDE-based methodology for modeling a real-time system, and then directly apply the algorithm in [3] to generate the implementation and perform the schedulability analysis.

2.2 UML Profiles

The UML answered the industrys needs for a common language to discuss information technology. UML is now the standard notation for Object-Oriented development and is generally recognized as a very useful tool for modeling the functional characteristics of a system [8]. However, in several current research areas such as real-time embedded systems, requirements related to response time, performances or schedulability that cannot be expressed functionally are crucial.

Since 1999, when the Object Management Group (OMG) issued an explicit request for proposals (RFP) for a domain-specific interpretation capable to deal with non-functional requirements, the main challenge in these kinds of approaches was to maintain the fully conformance with the UML standard.

The UML relies on its own extension mechanisms to define additional features by means of the notion of a *profile*: a profile customizes UML for a specific domain or purpose using extension mechanisms able to modify the semantics of the metamodel elements. The main advantage in the use of a profile is that this practice allow the modelers to choose the style and modeling constructs that they feel the best fit to their needs of the moment, adapting the language to a specific method, organization, or user.

The specification of a UML profile focuses on the definition of a set of *stereotypes*. A stereotype provides a method to extend a *metaclass*, creating a new UML metamodel element or adding new or extra semantics to the existing ones. The metaclass extended by the stereotype is usually referred to as *base class*. Each stereotype is characterized by a set of properties called *tagged values* and can be influenced by a semantic condition or restriction called a *constraint*.

In response to the OMG's RFP to address the needs of adding non-functional annotations to a UML model, in particular in the case of real-time embedded systems, various UML profiles were proposed.

Among these, the UML Profile for Schedulability, Performance and Time (SPT-Profile) [13] has been proposed by a working consortium of OMG member companies, and has been recently adopted as an OMG standard.

The SPT-Profile does not add elements to the UML metamodel, but defines a set of domain profiles for UML allowing for the construction of models that can be used to make (also early in the life cycle) quantitative predictions regarding the characteristics of timeliness, schedulability, and performance. It was not conceived for a specific analysis method, but is intended to provide a single unifying framework encompassing the existing analysis methods, still leaving enough flexibility for different specializations. Basically, its underlying idea is to import as annotations in the UML models the characteristics relative to the target domain viewpoint (performance, real-time, schedulability, concurrency), in such a way that various (existing and future) analysis techniques can usefully exploit the provided features.

The overall structure of SPT-Profile is modularized so to allow system designers and developers to only use those elements of the profile that they need, and to consent future extensions. The profile is in fact partitioned into a number of sub-profiles, i.e., "profile packages dedicated to specific aspects and analysis techniques". At the core of the

profile is the general resource modeling framework, itself consisting of three sub-profiles dealing respectively with resource modeling, concurrency and time-specific concepts. Then, based on this common framework, more specific sub-profiles are defined. The UML profile we propose later in Section 4 focuses specifically on the Scheduling sub-profile of the SPT.

A different profile, called the UML-RT [17], has been proposed to model software architectures in real-time systems. UML-RT extends UML with stereotyped active objects, called *capsules*, to represent system components, where the internal behavior of a capsule is defined using state machines. The interaction with other capsules takes place by means of protocols that define the sequence of signals exchanged through stereotyped objects called *ports* and specify the desired behavior over a connector.

The importance for real-time and embedded systems communities to converge on a unified notation, is also confirmed by some important experiences that, originated from a *not standard* notation to describe their metamodels, are now moving towards a UML-based input notation [7]. In particular, the UML Platform Profile (UML-PP) has been inspired by the work earlier done within the already mentioned Metropolis design environment [2]. Its semantics is defined in terms of the Metropolis Metamodel by establishing a direct correspondence between modeling elements of the UML-PP and elements of the Metropolis Metamodel. In this view the behavior of an embedded system can be captured at different levels of abstraction using different types of UML diagrams.

As said, our aim is to develop a MDE approach to real-time systems design based on a UML compliant input notation. Therefore, at the moment our choice naturally goes to SPT, which is the currently adopted OMG standard profile. A contingent problem with the SPT-Profile exists concerning UML versioning: SPT is still based on version 1.5 of the UML [10], now superseded by the new UML 2.0 superstructure [11]. Indeed, to respond to the changes introduced by the UML 2.0 and also to address several requested improvements to better specify the properties of real-time embedded systems, the OMG has now issued a new RFP for UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [12], and a new profile is under development to replace the existing UML SPT-Profile. However, waiting for such a profile to be released, this work refers to the current standard version that is still [13].

3 Scheduling Methodology

3.1 Model of execution

The UML profile and the methodology that we propose in the next section incorporate an algorithm for software mapping, which is the operation which fits functions into software tasks, and assigns parameters (periods and deadlines) to tasks.

To make the paper self-contained, in this section we are going to summarize the properties of the methodology proposed in [3] and extended in [4]. A full description of the developed algorithms can be found in the original papers.

We assume the scheduler to have real-time features; our chosen target is a preemptive EDF scheduler, though other algorithms for real-time scheduling might be used.

To make the methodology as more general as possible, we do not require in-depth knowledge of the application's internal workings: especially, we do not require the knowledge of worst-case execution times (WCET) of functions for generating the tasks. The task set can be generated, and the deadlines assigned, based only on the system topology and the end-to-end temporal constraints. For this reason, we require the system to be schematically described with a DAG, or *Directed Acyclic Graph*.

Our methodology allows a system designer to use the DAG view of the system to partition the application into a set of tasks and assign them real-time deadlines; in addition, we provide a feasibility test which can be run on the graph, to check if a system created with these algorithms will be schedulable.

A functional model is formally a t-uple $\{\mathcal{V}, \mathcal{E}, \mathcal{R}\}$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ the set of edges, and $\mathcal{R}$ is the set of mutually exclusive resources used by the functions[1].

- $\mathcal{V} = \{F_1, \ldots, F_n\}$ is the set of *functional blocks*. They represent the basic operations of the system, and can be aggregated into tasks. A block F_i has an *input port* and an *output port*. An input port is a *buffer of infinite size* that may contain *activation tokens*. When at least one *activation token* is present on the input port, the functional block is active. A functional block can execute only if it is active. Each functional block is executed once for each activation token; therefore, if multiple tokens are input to a block, we assume an OR activation semantics, i.e., one activation for each token. When the block executes, it first consumes one token, then runs for the duration of the a random variable comprised between 0 and γ_i. At the end of the execution it fires its output port. If no other token is present on the input port, the block becomes *inactive*, otherwise it remains active, and will be executed again according to the scheduling policy.

  A functional block F_i can also be characterized by a maximum computation time γ_i. This is only used for schedulability analisys and not for the generation of

[1] To ease the notation and to clarify the presentation, in this paper, we will not consider shared resources.

the task set. Therefore, in a first stage of the methodology, it can be ignored.

- $\mathcal{E} = \{l_1, \ldots, l_m\}$ is the set of *functional links*. A functional link $l_i = (F_h, F_k)$ connects the output port of functional block F_h (the source block) to the input port of block F_k (the sink). One functional block can be the source or sink of many links. When a functional block completes execution, it fires its output port: this means that tokens are sent on all the links starting from that output port. When fired, a token is instantaneously received on the input port of the sink, and no communication latency is accounted for.

An *external event* is produced by the external environment and is represented as a special functional block with no input links, denoted e_i. An external event is generated at a minimum interarrival time, denoted by T_i. It can be either *periodic* or *sporadic*; in the former case, events are generated at a constant rate T_i.

An *output* o_j is a special functional block with no output link. It represents a consumption by the environment of the data produced by the system and sent for instance to some actuator. For our purposes, an output is merely a stub where execution ends.

A *path* is the end-to-end execution of the system's functionality. When an external event arrives, one or more sequences of functional blocks are executed in an order defined by the policy of the underlying scheduler. These execution chains, which start from an external event and continue executing until an output is reached, are the paths; a path is an ordered sequence of functional blocks. We denote them with $P_k(e_i, o_j)$; the k index is related to the fact that more than one path can exist between the same external event and output stub. For simplicity, we will assume the paths to have a certain order, thus we can denote them without expliciting the source event and the sink output.

Since paths represent the end-to-end computations of the system, we can assign them a deadline that represents the maximum reaction time allowed for that computation. When event e_i is triggered, the functional blocks in path P_k are executed until output o_j is reached.

Δ_k is the relative deadline for P_k. Since a path can be activated many times (once for each trigger of the external event), every instance $P_{k,l}$ of the path will have its own absolute deadline, denoted with $\delta_{k,l}$. If we denote with $A_{k,l}$ the time when the l-th instance of path P_k is activated, then $\delta_{k,l} = A_{k,l} + \Delta_{k,l}$.

3.2 Real-Time Notation

The target of our methodology is to generate a task set based on the system topology. Since we are aiming at generating real-time tasks, we have to assign periods and deadlines for their execution.

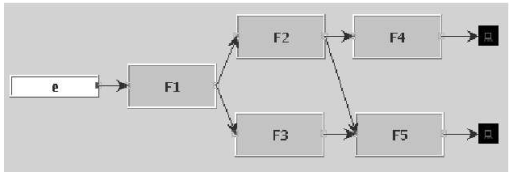

Figure 1. A sample DAG.

Path	P_1	P_2	P_3
Functions	F_1, F_2, F_4	F_1, F_2, F_5	F_1, F_3, F_5
Δ	20	25	25

Table 1. Path properties for the sample DAG.

A periodic task is denoted with τ_i and its j-th instance with $\tau_{i,j}$. The time when $\tau_{i,j}$ is activated and becomes ready for execution is $a_{i,j}$. Each task instance will be assigned an absolute deadline $d_{i,j}$ when activated. In our model, it is not possible to define a relative deadline for a task, as the difference between the absolute deadline (which depends on the path deadline) and the activation time of the task's instance may vary. Therefore, we define the *base deadline* as the difference between the absolute deadline of the instance and the arrival time of the event that caused the activation of the instance. The base deadline depends only on the path deadlines the task's instance belongs to, and hence it is constant.

A task instance is dispatched to the CPU for execution at time $s_{i,j}$ and will finish at time $f_{i,j}$. Its total *response time* will be $r_{i,j} = f_{i,j} - s_{i,j}$. The task set is schedulable if $\forall i, j,\ r_{i,j} \leq f_{i,j}$.

The *computation time* $c_{i,j} \leq r_{i,j}$ for task instance $\tau_{i,j}$ is the response time of the unpreempted and unblocked task instance. It will be upper bounded by the worst-case execution time (WCET) C_i, which is a property of the task and depends by the maximum computation times of its functional blocks. Since we do not account for inter-block communication, $C_i = \sum_l \gamma_l,\ l \in \tau_i$. However, a-priori knowledge of the WCET is not required to apply our algorithms, but only for schedulability tests.

3.3 Mapping

We currently provide two algorithms, called *Late Activation* (LA) and *Joined Late Activation* (JLA); we will summarize the latter in brief. As a reference example, the simple graph shown in Figure 1 will be used. The only properties which are required to use the JLA algorithm are the end-to-end temporal constraints, expressed in the form of path deadlines, shown in Table 1.

JLA is an algorithm which partitions the system into a set of tasks and assigns them periods and deadlines. The task partitioning works according to the following rules:

1. A task is a chain of functional blocks and must be entirely included in a path. This means that the functional blocks in a task are sequentially connected.

2. If a functional block has more than one input link, it must be at the beginning of a task (and its only entry point). By applying the previous two rules, a task set generated by the DAG of Figure 1 can be the following:

Task	τ_1	τ_2	τ_3
Functions	F_1, F_3	F_2, F_4	F_5

3. If a functional block has more than one output link, then the task which contains it is extended along the shortest deadline path. For instance, if a block has two output links, one along a path P_1 and the other along the path P_2, with $D_1 < D_2$, then the task will extend along P_1, including the corresponding blocks, unless they contradict the previous rule or have already been assigned to another task.

In addition, the following rules determine the assignment of deadlines to tasks:

- Every time a task is activated by an external event or by a functional link, it is assigned an absolute deadline equal to the minimum absolute deadline of all subpaths originating from the activation event.

- A task activated directly by an external event, let it be τ_1, has an absolute deadline of

$$d_1 = \min_i\{\delta_i \mid \tau_1 \in P_i\};$$

- The absolute deadline of a task $\tau_{i,l} \in P$, which is activated by $\tau_{i-1,h} \in P$ at time $f_{i-1,h}$, is

$$d_{i,l} = \min_j\{\delta_j \mid \tau_{i,l} \in P_j\}.$$

An example of the results of the application of JLA to the example of Figure 1 is shown below:

Task	τ_1	τ_2	τ_3
Functions	F_1, F_2, F_4	F_3	F_5
D	20	25	25

A complete, step-by-step description of the JLA algorithm can be found in [3] and [4].

4 Proposed Extended Profile

The main effort of this paper versus a UML view point is to extend the SPT scheduling sub-profile (introduced in Section 2.2) in order to model the mapping of functional blocks into a earliest deadline schedule threads (as described in the previous section). This section describes how the UML SPT-Profile can be used as it is, or possibly where it was necessary to extend it.

The usual structure in the UML community followed for presenting a profile is to introduce first the basic abstractions we need and their semantics (which is called the "Domain Viewpoint") and then to define how these abstractions can be modeled in UML (which is called the "UML Viewpoint").

With regard to the Domain Viewpoint, this has been broadly discussed in Section 3. The main abstractions we need to model focus on the notions of *resources*, *functional blocks*, *external events* and *paths*. Even though something quite similar to the concepts of resources, functional blocks and external events are present and expressed in a very detailed way in the UML extensions proposed in [13], there are no references to the notion of a path as intended here.

The rest of this section focuses on the UML viewpoint, i.e., we explains how to model the above notions into a UML-compliant notation. In particular, as explained below, we have identified a specific type of diagram, the Activity Diagram, as the central element of our profile. Activity Diagrams [8] were introduced in UML to model the procedural flow of actions that are part of a larger activity. Since the concepts and the scheduling algorithm we refer in this paper are based on a *dataflow model* (in particular on a *Direct Acyclic Graph* representation), we note a natural correspondence with these kind of diagrams.

Resource. To model the resources, in principle we would inherit the whole set of characteristics proposed in the SPT-Profile within the ≪SAresource≫ stereotype definition. In particular, this stereotype identifies a kind of resource that can be contended by multiple concurrent actions and whose access is protected by some mechanism. Several tagged values are associated with the definition of resource, but these are too detailed for our purposes, so they will be not considered at this stage. Unfortunately, such stereotype cannot be applied directly to the ObjectNodes of an Activity Diagrams [11], since this concept does not specialize any element of the stereotype base class list [13]. In order to exploit the definition of resource as we utilize it in our profile, without infringing the semantics of SPT-Profile elements, we introduce a new stereotype (≪Resource≫), which is defined in identical way as ≪SAresource≫, but can be applied also to ObjectNodes.

Functional block. A functional block is basically a unit of work with a defined execution pattern, which contends the

Stereotype	Base Class	Parent	Used Tags
≪Resource≫	ObjectNodes	≪SAresource≫	
≪SAction≫	Action		SAworstCase
≪ExtEvent≫	InitialNode	≪SAtrigger≫	SAoccurrence
≪Path≫	ActivityPartition		Deadline

Table 2. Stereotypes Definition

use of some resources with other functional blocks. Such definition corresponds to the specification of ≪SAaction≫ used for the scheduling domain description in the SPT [13]. The SPT Profile again introduces a lot of characteristics in order to clarify the role of an action, but in our case only the SAworstCase tagged value will be considered at the present stage. Such value refers to the maximum computation time γ of the functional block as defined in Section 3.1. This stereotype can be directly applied to an Action element of an Activity Diagram, since it specializes the concept of an Action from BasicActions [11], which are allowed as base classes for the stereotype ≪SAaction≫ [13].

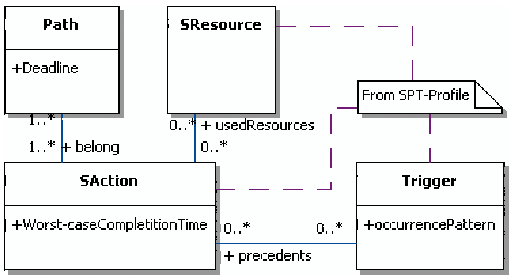

Figure 2. The extension on SPT-Profile with the Path

The scheduling algorithm in Section 3 allows a functional block to have a set of input or output ports. Such ports can manage tokens to transfer control, step by step, between functional blocks through the defined dataflow. In the UML Activity Diagram, the actions already provide this concept by means of the notion of *pins*. Since we have represented a functional block by means of an extension of the action of an Activity Diagram, hence the ports can be rightly represented using the pins.

External event. As usual, in the case of embedded systems there appears quite often the notion of some event, generated for example by a sensor, which triggers system execution. Activity Diagrams can model the starting point for executing an activity by means of their InitialNodes [11], but in its usual definition no real-time annotations are possible. The SPT-Profile, on the otehr side, provides the instrument to model events of this kind by means of the ≪SAtrigger≫ stereotype. However, this stereotype cannot be applied to an InitialNode, since neither the latter, nor any of its generalization, appears in the base class list of ≪SAtrigger≫ [13]. Therefore for this case we introduce a new stereotype (≪ExtEvent≫) defined as ≪SAtrigger≫, which can be applied to the InitialNode modeling concept. At the moment, the only tagged value that we associate to this new stereotype is the SAoccurrence one, which defines the pattern of inter-arrival times between consecutive occurrences of the trigger, as specified in [13]. Referring to the notation adopted in Section 3.1, this tagged value corresponds to the characteristic of an external event denoted as T.

Path. In their classical formulation, Activity Diagrams allow to group flows of actions by means of *activity partitions* (these are also sometime referred to as *swimlanes*). The grouping is made with respect to the role of some entity or process who is responsible for carrying on those actions. In our case we want to force somehow this concept and model the notion of paths as introduced in Section 3. A path is an ordered sequence of steps that the system has to execute following up an input by an external event: we can then see an activity partition as a container of a set of related actions triggered by an external event and whose execution has to be performed in accord to some timing constraints. Figure 2 depicts the notion of path into the SPT-Profile domain viewpoint. The stereotype ≪Path≫, applicable to the ActivityPartition UML modeling element, defines its end-to-end constraint for the computation by means of a Deadline tagged value. In the execution model of Section 3.1 the latter represents the maximum reaction time expected for a computation denoted as Δ.

In some cases an action could belong to more paths; in these cases the usual graphical and textual notations of Activity Diagrams can be adopted. In particular, the classical UML Activity Diagram textual notation entails to annotate each node with the activity partition or partitions to which it belongs, enclosed within round brackets.

A resume of the definitions and uses of the previously described stereotypes is given in Table 2. Each row specifies: the name of the stereotype; the classes of the UML metamodel to which it can be applied (**Base Class**); the name, if it exists, of the extended SPT-Profile stereotype (**Parent**), and the tagged values defined or required (**Used Tags**).

5 Example

This section exposes an illustrative example on use of the proposed profile and explains how the methodology works in order to schedule functional block in a real-time situation. In particular this example shows a scenario where a system collects informations form the environment by means of a sensor, filters them and update the its control subsystem. At the same time the filtered informations capturer, can be transformed in a human readable form and stored in a log repository. Periodically a user can query such repository in order to monitor the whole system.

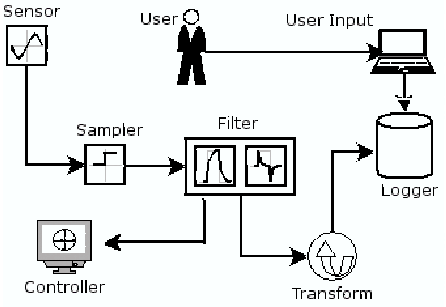

Figure 3. The Scenario in the Example

Figure 4 depicts the use of the extended notions of UML Activity Diagrams in order to model the informal scenario in Figure 3.

The model is composed by three paths (DataControl, SignalRecord, UserLog). Each of them is represented by an Activity Partition stereotyped as ≪Path≫ and their deadline is represented by the tagged value Deadline set to 18, 40, 200 respectively. DataControl and SignalRecord share the same external event (e1) and the first two functional blocks (Sampler, Filter). e1 produces the stimulus for the system with a constant interval of time and in particular each 40 msec. On the other hand, UserLog has its own periodic initial node (e2), but share its last action node with SignalRecord (Logger). At last, each functional block was modeled by an action stereotype as ≪SAction≫ and the duration of the their execution is represented by the tagged value SAworstCase.

From this formalization of the system, by means of a transformation engine the UML scenario can be provided as input to the algorithms described in Section 3.3 in order to obtain an activity scheduling if it is possible. For the motivations argued in Section 6, at the moment the engine is not already implemented, although theoretically such transformation is possible.

The table below shows in a compact form the path structure of the example:

Path	Functional blocks	Δ
DataControl	Sampler, Filter, Ctrl	18
SignalRecord	Sampler, Filter, Transform, Log	40
UserLog	UserInput, Logger	200

We can now apply our partitioning methodology, as explained in Section 3.3. We first use rule 1, which states that every task must be included in a path. One possible resulting task set is shown below:

Task	Functional blocks
τ_1	Sampler, Filter, Transform, Logger
τ_2	Ctrl
τ_3	UserInput

This partitioning is not in accordance with rule 2, because the Logger block has more than one input link. By applying the second rule, we obtain the following:

Task	Functional blocks
τ_1	Sampler, Filter, Transform
τ_2	Ctrl
τ_3	UserInput
τ_4	Logger

Last, we apply rule 3. It specifies that τ_1 should not include the Transform block, but rather the Ctrl block; this is because the DataControl path has a stricter deadline than the SignalRecord path. If the partitioning used were the one described above, the Transform function would be executed before the Ctrl block, resulting in a priority inversion. The final task set is as follows:

Task	Functional blocks	Deadline	Period
τ_1	Sampler, Filter, Ctrl	18	40
τ_2	Transform	40	-
τ_3	Logger	40/200	-
τ_4	UserInput	200	100

Deadlines have also been assigned according to the rules described in Section 3.3. These are the absolute deadlines relative to the first instance of the tasks; subsequent instances will have greater deadlines. It is possible to assign relative values for deadline; these would be the base deadlines; however, they are not in the scope of this article.

τ_1 and τ_4 are activated by external events, therefore they can be assigned a period equal to that of the event. On the other hand, τ_2 and τ_3 do not have an explicit period, since they are activated at non-periodic intervals after the execution of their predecessors.

The reason why τ_3 is assigned two different deadlines is because it is activated twice, once by τ_2 and the other by τ_4. When activated by τ_2, its deadline is that of the SignalRecord path, i.e. 40; when activated by τ_4, it has the deadline of the UserLog path, which is 200.

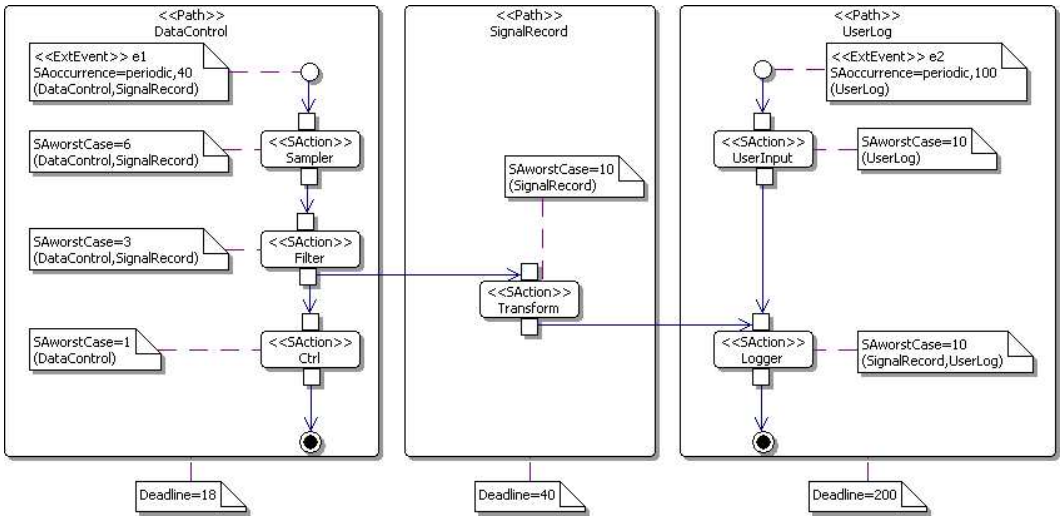

Figure 4. A DAG representation by means of the UML Activity Diagram extension.

This task set is ready for execution on an EDF scheduler. The Gantt chart produced by a simulation of the schedule is shown in Figure 5. In the chart, up-pointed arrows represent task activation times, while a task deadline is depicted with an arrow pointing down. Rectangular blocks (independently of their color) show the task running on the CPU.

Starting from an XML representation of the DAG, the whole methodology, from the task partitioning and the deadline assignment to the simulation, has been executed with a tool we have developed.

6 Discussion

In these last years, UML communities are involved in an important and difficult transactions from the 1.5 version of the language to the more recent 2.0. The new specification aims to be more "*well defined*" and complete than the previous ones, and it is also included in a wider context related to MDE techniques. However, the last and final UML 2.0 specification was relased only the last July [11]. This aspect is quite important on a practical point of view, since there is a lack of fully compliant tools yet. In addition, the new version of the UML superstructure is not compatible with the old one, especially on the issues about the definition of Activity Diagrams. As direct consequence, as briefly introduced in Section 2.2, lots of description instruments such as standard profiles can be used in theory with UML 2.0, but not in practice yet. Of course, these problems presently exist, but due to their temporary nature caused by current technological changes, they do not leak the basic ideas suggested by this work.

The profile presented in this work, beyond providing an interface toward the standard UML world for the work described in Section 3.1, inherits all the advantages of the underlying methodologies. In particular, we want to point our the following features:

- The methodology allows to specify and develop applications taking into account their temporal behavior, expressed directly by means of constraints on its functionalities.

- Thanks to the automatic generation of the task set, changes in the functional behaviour are automatically and optimally reflected in the task set organization, thus reducing the development time.

- The functional specification ensures a correct use of shared resources, by automatically generating the synchronization mechanisms that are necessary to ensure mutual exclusion. This prevents possible errors, simplifying the job of the designer. In addition, as described in [3] and [4], the use of the SRP [1] protocol togheter with the EDF scheduling policy ensures optimality from a schedulability point of view.

- The methodology directly supports schedulability analysis. The result of the analysis can easily be back-annotated on the UML specification. The designer can thus directly act on the specification for removing bottlenecks, if the system is not schedulable; or she/he can evaluate the possibility to add new functionalities, if there is still free space. Annotating the results directly on the diagrams permits a faster evaluation of the trade-offs.

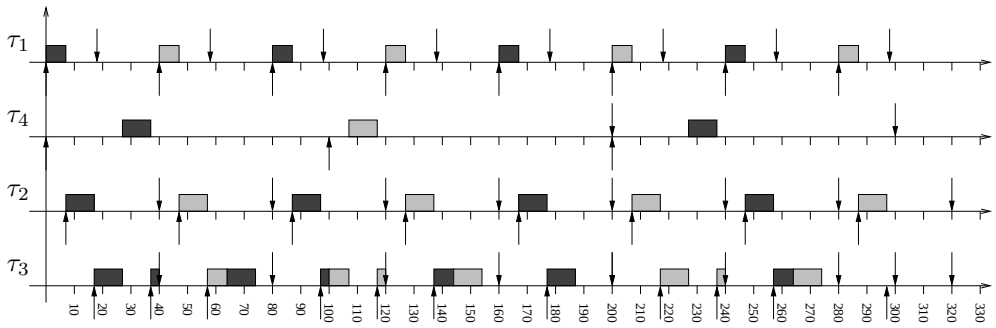

Figure 5. Schedule chart example.

7 Conclusions and Future Work

In this paper we have presented an extension to the SPT-Profile in order to provide a UML interface to the methodology illustrated in [3]. The use of UML notations should involve the interoperability with the other design methodologies and tools. Furthermore, we believe that UML in the MDE context represents, even in real-time and embedded scenarios, a useful and easier support toward the systems design automation from specification to implementation.

Actually, the proposed profile also preserves the advantages of the underlying methodologies such as the explicit specification of the application's temporal behavior or the automatic generation of the task set and mutually exclusive resource sharing.

As a possible future work, it is our intention to implement a model into a model transformation engine from the profile specification to the input schema actually used by the tool that generates the task set and simulate an EDF scheduler execution. In accordance with the problems related to current technological limitations argued in Section 6, such implementation will be as complete and UML 2.0 compliant as possible.

Furthermore, as soon as the new MARTE [12] stable and complete specification will be released, it is our intention to redirect these ideas to take into account the new results.

References

[1] T. P. Baker. A stack-based allocation policy for realtime processes. In *IEEE Real-Time Systems Symposium*, december 1990.

[2] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An Integrated Electronic System Design Environment. *Computer*, 36(4), 2003.

[3] C. Bartolini, G. Lipari, and M. Di Natale. From functional blocks to the synthesis of the architectural model in embedded real-time applications. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 458–467, 2005.

[4] C. Bartolini, G. Lipari, and M. Di Natale. A methodology for mapping functional blocks into earliest deadline scheduled threads. Technical report, Scuola Superiore Sant'Anna, 2005.

[5] J. Bézivin. On the unification power of models. *Journal of Software and Systems Modeling*, 4(2):171–188, May 2005.

[6] A. Brown. Model driven architecture: Principles and practice. *Journal of Software and System Modeling*, 3(4):314–327, 2004.

[7] R. Chen, M. Sgroi, L. Lavagno, G. Martin, A. Sangiovanni-Vincentelli, and J. Rabaey. *UML for real: design of embedded real-time systems*, chapter 5 – UML and platform-based design. Kluwer Academic Publishers, 2003.

[8] H. Eriksson, M. Penker, and D. Fado. *UML 2 Toolkit*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[9] D.-I. Kang, R. Gerber, and M. Saksena. Parametric design synthesis of distributed embedded systems. *IEEE Transactions on Computers*, 49(11):1155–1169, 2000.

[10] OMG. *OMG Unified Modeling Language Specification*, OMG Document – formal/03-03-01 edition, March 2003.

[11] OMG. *UML 2.0 Superstructure Specification*, OMG Document – formal/05-07-04 edition, July 2005.

[12] OMG. *UML Profile for Modeling and Analysis of Real-Time and Embedded systems*, OMG Document – realtime/05-02-06 edition, February 2005.

[13] OMG. *UML Profile for Schedulability, Performance and Time Specification*, OMG Document – formal/05-01-02 edition, Jannuary 2005.

[14] M. Saksena and P. Karvelas. Designing for schedulability: Integrating schedulability analsysis with object-oriented design, June 2000.

[15] M. Saksena, P. Karvelas, and Y. Wang. Automatic synthesis of multi-tasking implementations from real-time object-oriented models. In *Proceedings of the IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, March 2000.

[16] A. Sangiovanni Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[17] B. Selic and J. Rumbaugh. *Using UML for modeling complex real-time systems*, 1998. White Paper.

[18] L. Tratt. Model transformations and tool integration. *Journal of Software and Systems Modelling*, 4(2):112–122, May 2005.