Schedulability Analysis at Early Design Stages with MARTE

Chokri Mraidha, Sara Tucci-Piergiovanni, Sébastien Gerard

Chapter 6
Schedulability Analysis at Early Design Stages with MARTE

Chokri Mraidha, Sara Tucci-Piergiovanni, Sebastien Gerard

Abstract The construction of a design model is a critical phase in real-time systems (RTS) development as the choices made have a direct impact on timing aspects. In traditional model-based approaches, the design relies largely on the designer experience. Once the design model is constructed, a convenient schedulability test has to be found in order to ensure that the design allows the respect of the timing constraints. This late analysis does not guarantee the existence of a test for the given design and does not allow early detection of unfeasible designs. In order to overcome this problem, this chapter proposes the first UML/MARTE methodology for schedulability-aware real-time software design models construction.

6.1 Introduction

Model-based approaches for real-time systems (RTS) development aim at going from requirements specification to binary code production with the insurance of respecting the functional and non-functional requirements of the system. These model-based approaches (e.g. [61, 108, 229]) introduce a number of intermediate models between the requirements and the binary code. Requirements are usually formalized with use case scenarios. Even when modeled with other formalisms, critical scenarios that represent the system response to external stimuli are specified along with the system response deadlines. The functional model aims at representing functional blocks and their interactions and to show how functional blocks participate in above defined scenarios.

The functional model is then refined in a design model that introduces the mechanisms/patterns for the realization of the functional model on the underlying platform. Actually, the design model defines the architecture of the system still being independent of specific platforms. To this end abstracted resources and services are

CEA, LIST, 91191 Gif-sur-Yvette CEDEX, France e-mail: {chokri.mraidha|sara.tucci|sebastien.gerard}@cea.fr

assumed. Threads, which are the unit of execution handled by OS platform services, have to be structured to determine the threading strategy of the system. By defining a mapping of functional blocks on threads, it is defined how the system reacts to external stimuli. Finding a convenient threading strategy is a complex task. The designer has a set of real-time design patterns [85] that he can use to determine the adequate number of threads and the good grouping of functions to threads towards the definition of a of the system. This concurrency model is of primary importance with respect to the system's . In order to ensure that the concurrency model satisfies timing requirements, a timing validation , like schedulability analysis , is necessary at this point. However, this timing validation can lead to the following problems: (i) the resulting design model can be too difficult or even impossible to analyze [26], (ii) the resulting design model is analyzable but the designer would like to explore other possible design models, to explore several candidates from a schedulability point of view. From these considerations, it follows that it is necessary to guide the designer in the construction of an analyzable design model, i.e. a design model for which a schedulability test there exists and possibly support the cohabitation of several analyzable design candidates in order to support a comparative analysis.

This chapter focuses on a real-time methodology, called Optimum, offering a UML front-end for the designer that conforms to a formal model for schedulability analysis and supports the evaluation of different architecture candidates. In order to offer such a UML front-end, standard UML needs, on one hand, to be extended to schedulability concepts and, on the other hand, to be restricted in the use of some elements to express a precise semantics. To this purpose Optimum uses MARTE [219], a standard specialization of UML for real-time and embedded systems development. It provides support for specification, design and verification/validation stages. Actually, before MARTE, the UML SPT profile [218] was available. In [25] authors show how SPT was not sufficient to express some basic concepts for schedulability and they propose to extend the profile in ad hoc way with those concepts. Optimum, which uses similar concepts used in [25], does not need to extend MARTE. MARTE proved to be enough rich to express all the concepts needed to build schedulability models. However, as the MARTE profile is quite rich, a restriction of the language is necessary in order to delimit the usage of MARTE concepts in the context of the Optimum methodology. Nevertheless, this restriction of the language has to preserve the compatibility with MARTE standard. In the best of our knowledge, Optimum is the first methodology for schedulability analysis using a standard specialization of UML.

The chapter is organized as follows. Section 6.2 presents an overview of the Optimum methodology . Section 6.3 presents the formal schedulability analysis model considered in the Optimum methodology. In section 6.4, a detailed description of the methodology and its conformance to the formal schedulability model is given. Section 6.5 illustrates the methodology usage on an automotive example. Related work is discussed in section 6.6. Section 6.7 concludes the chapter.

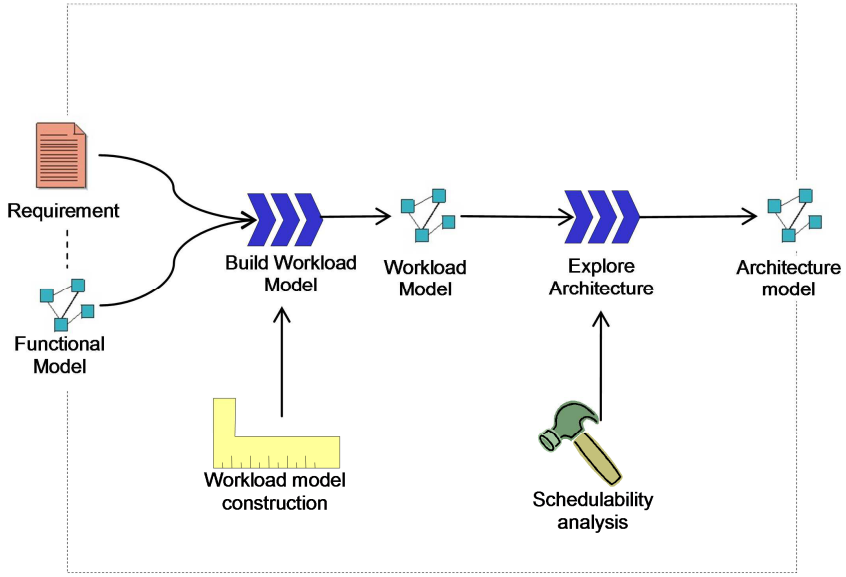

Fig. 6.1: Optimum process overview

6.2 Overview of the Optimum Process

The Optimum process guides the selection and the design of the concurrency model on the base of hard real-time constraints, expressed at the end of the requirement specification phase, towards the construction of concurrency models analyzable in automated way. To this end the methodology defines a process depicted on Figure 6.1 for the generation of the architecture model specifying a concurrency model and a deployment model on the execution platform .

The methodology process has as entry point two artifacts: 1) the functional model, i.e. a description of system end-to-end scenarios and 2) the description of timing requirements . Both artifacts are assumed to be available at this stage of the development process. Some basic characteristics of the hardware abstraction layer are also assumed to be available, as the type of the execution hosts that constitute the resource platform for the execution of identified scenarios. In fact, the type of execution host is used to allow the assignment of time budgets to the functions in the scenarios. This available information is organized in a specific UML model enriched with the MARTE profile called Workload model . The Workload model specifies two concerns: a graph of function activations along with execution time budgets and end-to-end deadlines , and an abstract layer representing the execution platform . In order to obtain a satisfactory architecture model, the methodology provides at this point a way of exploring different alternatives by setting a so-called

analysis context . Each analysis context contains an architecture model subject to evaluation. The architecture model contains two views. The first view is generated by transformation of the graph of function activations contained in the Workload model. This transformation, also called , defines the tasks and the functions allocated on them. The second view represents the software platform resources in terms of tasks along with their scheduling parameters , and the hardware resources in terms of processing and communication. More in detail, this architecture model describes the evaluated concurrency model by schedulability analysis techniques and embeds also the following information (i) the mapping of functional blocks on tasks, (ii) the level of threading for each functional block and possible synchronization resources needed to manage multithreading (if any), (iii) the needed OS support in terms of the scheduler algorithm and preemption capabilities, (iv) the policy for serving external events, (v) the needed to protect . The evaluation of the dynamics paradigm can cover different aspects such as resource utilization , response times , jitters , etc.

6.3 The Schedulability Model

In this section we present the formal model to analyze the schedulability of a real-time system. We introduce the formal notation to later show the conformance of the UML/MARTE model to the presented formal model.

The presented model captures a distributed system with fixed-priority tasks and task dependencies. This model is the one assumed by the test of Palencia and Harbour [223] and available in the MAST open source tool. The model is quite general, considering that fixed-priority scheduling is the most common scheduling algorithm available in practice. Let us also remark that whenever the tasks set under consideration is deployed on a single CPU, the Palencia and Harbour test is anyway applicable, reducing to the Lehoczky test [174]. Tasks are characterized by an arrival pattern that can be periodic or aperiodic with minimum inter-arrival time. We consider an event-activation paradigm. Given a set of external events, each external event triggers one computation, the system response for that event. One single task may execute the whole response or the response may be ”segmented” by a sequence of tasks possibly spanning several computation resources. In case of segmentation, a number of tasks are not directly triggered by the external event. For those tasks, the event-activation paradigm states that the trigger for activation is execution’s end of the previous task. In case more than one trigger there exists for the same task, AND/OR semantics may be specified. The OR semantics implies that the task is triggered by the execution’s end of any of the previous tasks, while the AND semantics implies that the task is triggered only when all the previous tasks have completed. In this paper we restrict our attention to the OR semantics. The reason of this choice is twofold: on one hand this assumption let simplify the model and makes the presentation clearer, on the other hand the test of Palencia and Harbour does not fully support the AND semantics (the AND semantics can be specified only among

events of the same system response, events with the same rate), therefore we could not use available open source tools to analyze a model containing AND semantics.

The model is characterized by a set of events $E = (e_1, e_2, ..., e_n)$, a set of tasks $T = (\tau_1, \tau_2, ..., \tau_m)$, a set of computational/communication resources $R = (r_1, r_1, ..., r_l)$, a characterization of event occurrences and a characterization of task executions. The characterization of event occurrences is constituted by a set of tuples, one for each event $e_i \in E$ of the form $(e_i, t_i, R_i, D_i, WR_i)$, where e_i is the event whose occurrence being characterized, t_i is the period of the event, R_i is the response to the event e_i. R_i is a total order of tasks $(T_i, \rightarrow_i)$, where $\emptyset \subset T_i \subseteq T$ is the set of tasks to be executed in response to the event e_i. For any two tasks τ_h, $\tau_k \in T$: the task τ_h has to be executed before τ_k, in the response R_i if and only if $\tau_h \rightarrow_i \tau_k$. D_i is the end-to-end deadline for response R_i. WR_i is the worst case response time for response R_i.

Characterization of task executions is constituted by a set of tuples, one for each task $\tau_h \in T$ of the form $(\tau_h, P_h, r, CS_h, C_h, B_h, WTR_h)$, where: τ_h is the task being characterized, P_h is the assigned priority, $r \in P$, is the computational/communication resource the task is executed on, C_h is the computational cost of the task (not considering any contention time), CS_h: ordered list of critical sections $(cs_{h1}, cs_{h2}, ..., cs_{hl})$ the task will pass through. A critical section represents an interval of non-preemption. The scheduler cannot interrupt a task in critical section even if a higher priority task is ready for execution. Critical section cs_{hi} has a duration $c_{hi} \leq C_h$. B_h is the blocking time, i.e. the time a task is blocked by another task with lower priority. This happens during concurrent access to critical sections: when a lower priority task is in critical section as it cannot be preempted until it releases its lock. A higher priority task has to wait the lock release before acquiring the lock. WTR_h is the worst-case response time of the task τ_h.

Note that in our model the same task can appear in more than one response and that each response establishes a total order in the tasks execution during the response. For the model to be valid, the following property must hold:

Partial-Order Property: the set of event characterizations induces a cyclic-free partial order on the set of tasks T. This property means that a unique order between dependent tasks can be established.

Schedulability condition: The model is schedulable if and only if all responses R_i have a worst-case response time $WR_i \leq D_i$.

In our model the following additional assumptions hold:

Assumption 1. (Fixed-priority scheduler). We assume that for each computational/communication resource there exists one scheduler arbitrating the access to the resource by a fixed-priority policy.

Assumption 2. (Priority-ceiling). We assume that passive resources, accessed in critical section regions, are protected by a priority ceiling protocol (to avoid priority inversion).

Assumption 3. (CAN-like channels). We assume that communication channels are arbitrated by a CAN-like protocol, where messages inherit the priority level of sending tasks. At destination buffered messages are dequeued on priority basis.

6.4 Detailed Optimum Methodology

In this section a description of the standard modeling language on which the Optimum methodology is based is given. Then the models used and produced by the Optimum process will be characterized. A third subsection will present the conformance of the produced software architecture model with the formal schedulability analysis model presented in section 2. At last, the software architecture exploration phase will be detailed.

6.4.1 Modeling Language Description

The MARTE standard [219] provides all the concepts needed for the analysis and design of real-time and embedded systems (RTES) in a model-based approach. As a standard, MARTE proposes 158 concepts in order to cover a large broad of development needs for RTES. Furthermore, MARTE gives several concepts with a close semantics to represent a common global notion. For instance, a schedulable resource could be a SwSchedulableResource from MARTE::SRM or a SchedulableResource from MARTE::GRM. Differentiation of these nearby equivalent concepts can be difficult to make. Thus, a MARTE-based methodology should specify a subset of MARTE concepts that is sufficient for its purpose.

This section introduces the concepts of MARTE on which the Optimum methodology relies. The usage of these concepts in the methodology is then restricted by a profile, called the MARTE4Optimum profile.

6.4.1.1 The MARTE4Optimum Profile

Table 6.1 enumerates the 14 useful concepts for the Optimum methodology out of the 158 ones offered by MARTE. The needed concepts deal with platform re-sources modeling, schedulability analysis modeling and allocation modeling. MARTE stereotypes extend too general metaclasses of UML. Allowing such a large applicability may make validation of methodological rules more complex and limit automation of a refinement process for the methodology. The Figure 6.2 below represents a stereotype specialization principle. A MARTE4Optimum GaWorkloadBehavior stereotype that specializes the MARTE GaWorkloadBehavior stereotype is created. The specialization does not add any property and preserves all properties of the original stereotype but the extension of the NamedElement metaclass. We make use of the UML redefinition capability to redefine this extension with a more special one (a UML Activity is also a UML NamedElement). Therefore, we restrict the application of the GaWorkloadBehavior stereotype to a UML Activity which is a rule of the Optimum methodology described in the following section.

Table 6.1 lists the UML extensions specialization for the MARTE subset.

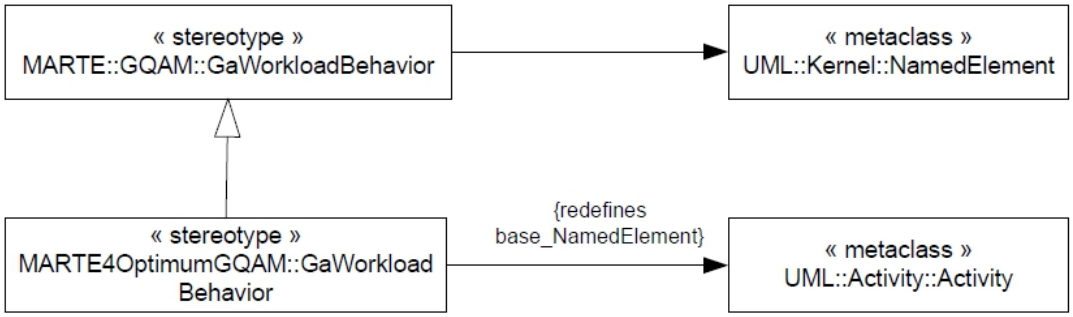

Fig. 6.2: Stereotype specialization principle

Table 6.1: Optimum restriction of MARTE subset

MARTE4Optimum stereotype	**UML extensions**
Alloc::Allocate	Abstraction
Alloc::Allocated	CallAction, Property
GRM::SchedulableResource	Property
GQAM::GaPlatformResources	Class
GQAM::GaWorkloadBehavior	Activity
GQAM::GaWorkloadEvent	AcceptEventAction
HRM::HwComputing::HwProcessor	Class
SAM::SaAnalysisContext	Package
SAM:: SaEndToEndFlow	ActivityPartition
SAM:: SaCommHost	Connector
SAM::SaExecHost	Property
SAM::SaSharedResource	Property
SAM::SaCommStep	ControlFlow
SAM::SaStep CallBehavior	Action

6.4.1.2 MARTE Compatibility

The MARTE4Optimum profile is constructed with respect to the following rules: (1) MARTE4Optimum stereotypes and general MARTE stereotypes have the same name, (2) MARTE4Optimum stereotypes inherit from MARTE stereo-type properties and no additional property is added to them, (3) UML metaclass used in the extension redefinition is necessary a specialization of the UML meta-class used in the general MARTE stereotype extension.

Compatibility with the MARTE profile is then preserved by the MARTE4Optimum profile. The export of a MARTE4Optimum model in MARTE can be easily realized by a one-to-one mapping consisting in unapplication of Optimum4MARTE stereotypes and the application of the corresponding MARTE stereotypes.

6.4.2 The Optimum Models

6.4.2.1 The Workload Model

The Workload model is constituted of a workload behavior specifying the controlled sequence of actions triggered by external stimuli enriched with timing information. The construction of the workload behavior proceeds by the generation of a UML Activity diagram containing a canonical form of the controlled sequence of actions contained in the functional model. For the canonical form, the following properties hold.

Properties on the Activity diagram are:

P1 - The subset of activity diagram elements used is: AcceptEventAction nodes, CallBehaviorAction nodes, FlowFinalNode, MergeNode, ControlFlow, ActivityPartition. All other activity diagram elements are not used.

P2 - Events are modeled as UML AcceptEventActions that have UML Triggers referencing the Events. These events are modeled with UML SignalEvents.

P3 - In response to an event, the invocation of a function (modeled as UML Activity) is modeled with UML CallBehaviorAction. The CallBehaviorAction represents the call to the UML Activity representing the function.

P4 - The last action to be executed in response to an event is always followed by exactly one UML FinalFlowNode.

P5 - For each path connecting an AcceptEventAction to a FlowFinalNode, there exist an ActivityPartition containing all the nodes of the path.

P6 - An AcceptEventAction has exactly one outgoing control flow that targets a CallBehaviorAction.

P7 - Each path of the Activity begins with an AcceptEventAction node, terminates with a FlowFinalNode.

P8 - The controlled sequence of actions does not contain cycles.

Properties on MARTE-based annotations are:

P9 - The Activity is stereotyped ≪GaWorkloadBehavior≫.

P10 - Each UML AcceptEventAction is stereotyped ≪GaWorkloadEvent≫. The arrival pattern (periodic, sporadic) of the event is specified in the arrivalPattern property. For the periodic pattern the period is specified following the Value Specification Language (VSL) syntax periodic=(value, unit) where value is a nu-merical value and unit is the time unit used (e.g. s, ms,..). Similarly, the minimum inter-arrival time is specified with the following syntax sporadic(value, unit).

P11 - Each UML ActivityPartition is stereotyped ≪SaEndToEndFlow≫. An end-to-end deadline is specified in the end2EndD property of the ≪SaEndToEndFlow≫ stereotype. This property is a duration specified with the following VSL syntax (value, unit).

P12 - Each action of type CallBehaviorAction is stereotyped ≪SaStep≫. Each step has a computational budget (execTime property) specified for a given type of execution host (host property).

At this stage of the process, the platform is not determined yet because no deployment is specified. However, as previously explained we need to specify an estimation of computation time budget for the actions stereotyped ≪SaStep≫. A computation time budget makes sense if it is related to a type of execution host. This type of execution host is modeled with a UML Class stereotyped ≪HwProcessor≫. We can notice that this type can be modeled in model library and reused in several workload models. This type of execution host will be used in the architecture model for the definition of the platform processing resources.

6.4.2.2 The Architecture Model

The architecture model enriches the workload model with (1) an explicit concurrency strategy and (2) a deployment model on a target hardware platform.

The concurrency model

A new UML Activity is built for the modeling of the concurrency strategy. This activity contains the information provided in the workload model and adds to it the information on the mapping of function executions (steps) to tasks. Moreover, this UML activity will explicitly describe synchronization resources used to treat contention in the multi-threading case. All the properties defined above for the characterization of the UML Activity for the workload model specification are valid for this new Activity. Yet, the following additional properties characterize the new UML Activity for the concurrency strategy modeling:

P13 - A UML ActivityPartition is created for each task. The property "represents" of this ActivityPartition is valuated with a reference to the corresponding element modeling the task in the platform resources (see next section).

P14 - Each CallBehaviorAction is mapped to at least one task. This mapping is modeled by including the CallBehaviorAction in the ActivityPartitions representing the tasks.

P15 - Let us consider two ActivityPartitions representing tasks t1 and t2 and two sets of CallBehaviorActions A1 and A2. If there exists a path from one or more actions of A1 to one or more actions of A2 and there exists a path from one or more actions of A2 to one or more actions of A1, then the mapping actions to tasks is not valid.

Additional properties on MARTE-based annotations are:

P16 - A UML Package is stereotyped ≪SaAnalysisContext≫is created. The platform property of this stereotype is valuated with a reference to the classifier stereotyped ≪GaResourcesPlatform≫that models the platform resources (see next section). The workload property is valuated with a reference to the Activity representing the

concurrency model. Note that this stereotype is the key language concept to support architecture exploration as detailed below.

P17 - Each ActivityPartition representing a task is stereotyped ≪SaStep≫.

P18 - For each task the execution time is derived by making the additions of the computational budgets of each step allocated to the task. The value of this execution time is saved in the execTime property of the ≪SaStep≫stereotype of the ActivityPartition representing the task.

P19 - For each task, the priority is automatically assigned using the rate monotonic priority assignment algorithm [9]. The assigned priority is saved in the priority property of the ≪SaStep≫stereotype of the ActivityPartition representing the task.

P20 - Each ActivityPartition representing a task is mapped to an execution host. A reference to this execution host is saved in the host property of ≪SaStep≫stereotype of the ActivityPartition representing the task.

P21 - For CallBehaviorActions that are shared between tasks and whose code must be protected the sharedResources property of these actions, stereotyped ≪SaStep≫are valuated with a reference to synchronization resource that is add-ed in the platform resources (see next section). The UML Behavior associated to the shared CallBehaviorActions are non-reentrant (isReentrant property is set to false).

P22 - Schedulability analysis results for the tasks are serialized in ≪SaStep≫stereotype properties of ActivityPartitions representing the tasks: respT for the response time and blockT for the blocking time. End-to-end response time is serialized in the endToEndT property of the ≪SaEndToEndFlow≫stereotype.

Platform resources

The platform resources are modeled in a classifier stereotyped by ≪GaResourcesPlatform≫. The platform resources model is an abstraction of the underlying software and hardware platform that is needed to define the deployment of the concurrency model on the available resources.

The following resources are considered in this abstraction: (i) tasks, here called schedulable resources, along with their scheduling parameters, (ii) synchronization resources used to solve contention in case of multi-threading (if necessary) and the protocol used for accessing them, (iii) the processing and communication resources topology, (iv) the scheduler algorithm used by the processing resources.

Each task, represented as an ActivityPartition in the concurrency model, is represented in the platform as a UML property stereotyped ≪SchedulableResource≫. The element ActivityPartition for the task is bound to corresponding UML property through the Activity Partition's meta-attribute "represents" which will reference the corresponding UML property stereotyped ≪SchedulableResource≫. Scheduling parameters for the task have to be specified. In the case a scheduling algorithm based on fixed priorities is assumed, priorities have to be specified as scheduling parameters. Scheduling parameters are specified through the schedParams property of ≪SchedulableResource≫. The value of a priority is in form of a VSL expression: schedParams=[fp(priority-value)]

Each synchronization resource is stereotyped as ≪SaSharedResource≫. The synchronizing protocol has to be specified in the protectKind property of the stereotype

≪SaSharedResource≫. The PriorityCeilingProtocol is used to protect the access to these critical section regions.

The processing resources are modeled with UML Properties stereotyped ≪SaExecHost≫and typed with the ≪HardwareProcessor≫Class used in the workload model for the definition of the computation time budgets of functions. These processing resources are interconnected with communication resources that are modeled with UML Connectors stereotyped ≪SaCommHost≫.

The offered scheduling algorithm is specified for the processing and communication hosts. The scheduling algorithm is specified through the property schedPolicy. The value of schedPolicy is equal to FixedPriority.

6.4.3 Conformance to the Formal Schedulability Model

In this section we formally show the conformance of the Optimum model to the formal model of Section 6.3.

Table 6.2: Conformance on sets

Conformance Property	**Formal Model**	**Optimum Model**
Event conformance	$E = (e_1, e_2, ..., e_n)$	Set of AcceptEventActions in the Activity Diagram stereotyped ≪GaWorkloadEvent≫(property P10).
Task conformance	$T = (\tau_1, \tau_2, ..., \tau_m)$	In the Optimum model, each task corresponds to one ActivityPartition stereotyped ≪SaStep≫and representing a UML property stereotyped ≪SchedulableResource≫(P13).
Resource conformance	$P = (r_1, r_2, ..., r_l)$	Set of UML properties stereotyped ≪SaExecHost≫/≪SaCommHost≫, properties of the UML Class stereotyped ≪GaResourcePlatform≫

As for assumptions on the formal model, **Assumption 1 (Fixed-priority scheduler)** is specified by property schedPolicy in the stereotype ≪saExecHost≫applied to processing/communication resources. The schedPolicy is equal to FixedPriority. **Assumption 2 (Priority-ceiling)** is expressed by the protectKind property of the stereotype ≪SaSharedResource≫applied to shared re-sources. The protectKind is equal to PriorityCeilingProtocol. **Assumption 3 (CAN-like channels)** means considering a communication channel where policy on messages to be sent/received is based on fixed-priorities. The assumption implies thus to have each ≪saCommHost≫with schedPolicy equal to FixedPriority. As for the **Partial-order property** on the set of task T, this property is satisfied in the Optimum model by property **P15**.

Table 6.3: Conformance on event occurrences characterization

Formal Model	**Optimum Model**
ti	From property P10 each AcceptEventAction is stereotyped ≪GaWorkloadEvent≫. The period is specified in the arrivalPattern property.
$R_i = (T_i, \rightarrow_i), \emptyset \subset T_i \subseteq T, T = (\tau_1, \tau_2, ..., \tau_m)$	T_i is obtained as follows: For each CallBehaviorAction *cbh* belonging to the ActivityPartition stereotyped ≪SaEndToEndFlow≫containing AcceptEventAction (ei), take the ActivityPartition stereotyped ≪saStep≫representing a schedulable resource (τ_h) that contains *cbh*. By property P1 no fork node there exists in the Activity diagram and by property P6 each AcceptEventAction is connect to exactly one CallBehaviorAction, this ensures that for each event there exists only one path towards the final flow node. The set of CallBehaviorActions in the ≪SaEndToEndFlow≫is then totally ordered. The order on tasks is the order established by control flows sequencing the actions and by property P15, no cycles can occur, reducing to a total order on tasks. By property P5, exactly one ActivityPartition stereotyped ≪SaEndToEndFlow≫there exists. By property P14 at least one ActivityPartition stereotyped ≪saStep≫representing a schedulable resource (τ_h) there exists, thus the set of tasks $T_I \neq \emptyset$
D_i	By properties P1, P5 and P6, each AcceptEventAction belongs to only one ActivityPartition stereotyped ≪SaEndToEndFlow≫(as already shown for R_i conformance). Deadline Di corresponds to the property end2EndD of the stereotype ≪SaEndToEndFlow ≫(P11).
WR_i	By properties P1, P5 and P6, each AcceptEventAction belongs to only one ActivityPartition stereotyped ≪SaEndToEndFlow≫. WR_i corresponds to the endToEndT (P22)

Table 6.4: Conformance on task execution characterization

Formal Model	**Optimum Model**
P_h	≪SaStep≫stereotype, attached to the ActivityPartitions representing schedulable resources, contains the property *schedParam:SchedParameters[0..*]=[fp(priority-value)]*
$r \in P$	The ≪SaStep≫stereotype, attached to the ActivityPartitions representing schedulable resources, contains the property host by property P20.
C_i	The ≪SaStep≫stereotype, attached to the ActivityPartitions representing schedulable resources, contains the property execTime by property P18.
CS_h	The list of critical sections the task pass through is specified in the property sharedResources of the stereotype ≪SaStep ≫applied to the CallBehaviorActions belonging to the ActivityPartition representing the task by property P21.
c_{hi}	The duration of the critical section csi is equal to the execution time of the CallBehaviorAction that includes the csi value in sharedResources of the applied ≪SaStep≫.
B_h	≪SaStep≫stereotype, attached to the ActivityPartitions representing schedulable resources, contains the property *blockT* (P22).
WTR_h	≪SaStep≫stereotype, attached to the ActivityPartitions representing schedulable resources, contains the property *respT* (P22).

6.4.4 Software Architecture Exploration Phase

Architecture exploration aims at finding an architecture model satisfying timing requirements of the system and possibly other designer criteria. Many techniques can

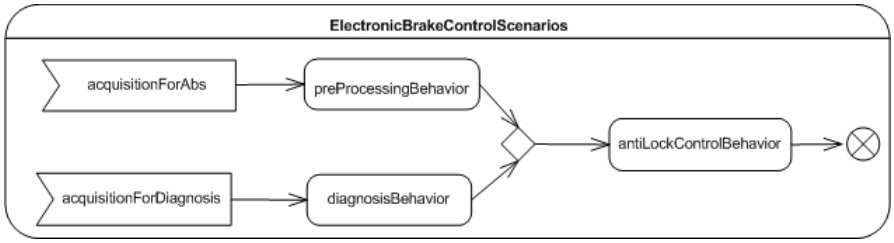

Fig. 6.3: System-level end-to-end scenarios

be employed in order to explore the design space, but here we are only interested in supporting some comparative process in which different architecture models can be built and co-habit for comparison. The key MARTE concept to support architectural exploration is the ≪SaAnalysisContext≫concept used in P16. Different UML packages stereotyped ≪SaAnalysisContext≫can be organized in order to store different candidates. Let us remind that the stereotype ≪SaAnalysisContext≫allows making reference to a platform and workload, which can be different for each context. Interestingly this stereotype has an attribute optCriterion used to annotate the context with designer criteria (optimization objectives) and their weights used for the context. When the designer is satisfied by one candidate, the exploration phase terminates and the candidate architecture is given as output to the process.

6.5 Application on an Automotive Case Study

In this section we will illustrate the application of the methodology on an automotive case study. This subsystem is a sensor-controller-actuator system com-posed of the following functions: a data processing function for data coming from the sensor, the anti-locking brake function calculating the command to send to the actuator, and a diagnosis function that disables the anti-locking function in case a fault in the subsystem is detected.

Figure 6.3 below presents the functional model describing the system end-to-end scenarios. Each function has an associated behavior modeled as a UML Activity which is referenced by a CallBehaviorAction. Two events (acquisitionForAbs and acquisitionForDiagnosis) are triggering the sequences of functions behavior execution. Between these events and the final flow node, there are respectively two end-to-end timing requirements of 60 ms and 100 ms.

6.5.1 Workload model

The set of system's end-to-end computations, called workload behavior, is represented with a UML activity diagram stereotyped with MARTE ≪GaWorkload-

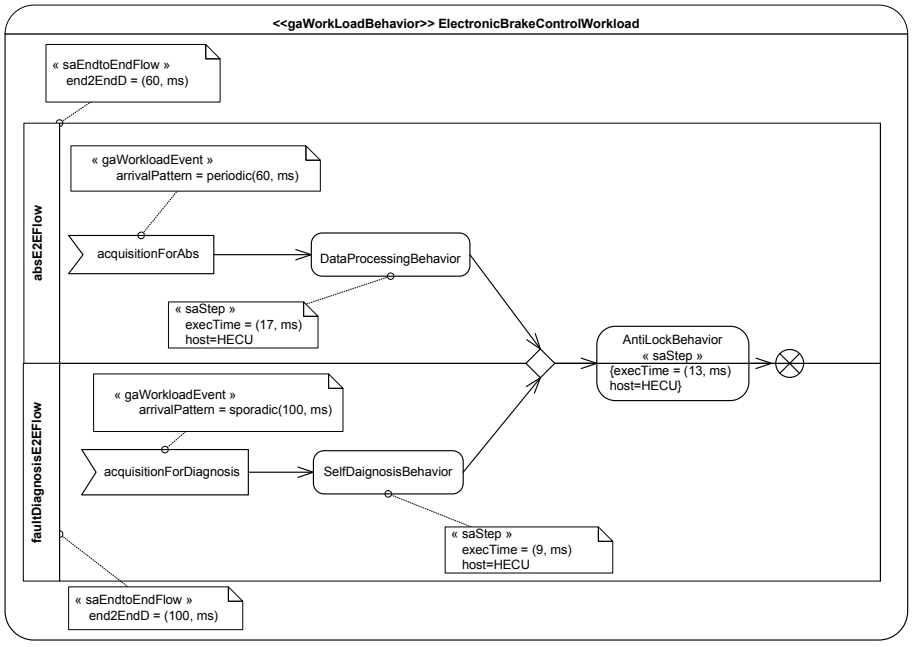

Fig. 6.4: Workload Behavior

Behavior≫stereotype. This workload behavior actually represents the behaviors activation graph. Each behavior is specified by a UML CallBehaviorAction annotated with MARTE ≪SaStep≫stereotype. Each step has a computational budget (execTime property) specified for a given type of execution host (host property). A step can be linked to a successor step with a control flow.

The workload behavior also specifies external events that trigger the steps. Each external event is modeled with a UML AcceptEventAction stereotyped ≪GaWorkloadEvent≫. The arrival pattern of the event is specified in the arrivalPattern property. For instance on Figure 6.4, aquisitionForAbs event is periodic with a period of 60 ms.

Once that control flow graph is defined, the steps and their respective activation events are grouped in so called end-to-end flows. An end-to-end flow is modeled with a UML activity partition stereotyped ≪SaEndToEndFlow≫. An end-to-end deadline can be specified for each end-to-end flow in the end2EndD property of the ≪SaEndToEndflow≫stereotype. The absE2EFlow has an end-to-end deadline equal to 60 ms. Let us to note that AntilockBehavior is a shared step between the two specified end-to-end flows.

As explained previously, at this stage of the methodology, we needed to define a type of processing resource in order to specify an estimation of computation time budgets for steps. This estimation can be used to perform feasibility tests with respect to expressed end-to-end deadlines and external events activation rates. The processing resource type is modeled with a UML class stereotyped ≪HwProces-

sor≫. In this example, HECU is the type of execution host for the steps. This type will be used in the architecture model to type the platform processing resources.

6.5.2 Generation of the Architecture Model

The following subsections give details of the generation of the Architecture Model starting from the Workload Model.

6.5.2.1 Mono-Processor Platform Definition

The proposed generation produces a mono-processor architecture. The platform (SaResources Class) includes then the property hecu of the type HECU. Inside the property host of the stereotype ≪saStep≫applied on CallBehaviorActions, the hecu value is set. For the execution host hecu the FixedPriority scheduling algorithm is set.

6.5.2.2 Independent Tasks Generation: Protecting Shared Functions

In order to apply a scenario-based task model generation (to get one single thread of execution for each event), the situation of a function belonging to two different end-to-end flows turning at two different rates and with different dead-lines must be handled. If we map the function on two different threads (one per scenario), schedulability analysis will compute an additional blocking time necessary to protect the function code duplicated in the two threads. In fact, to avoid inconsistencies the function virtually shared by two threads has to be accessed in mutual exclusion. Blocking time for the access at the critical section (shared function code) is thus computed.

The transformed graph, therefore, will explicitly describe synchronization resources used to treat contention in this case. In our example the step AntilockBehavior, originally shared between the two specified end-to-end flows, is here mapped into two different threads, namely task1 and task2. The synchronization among the two steps preceding AntilockBehavior, i.e., DataProcessingBehavior and SelfDiagnosisBehavior is modeled through the presence of a synchronization resource here named AntiLock, appearing in the property sharedResources, which actually represents a critical section for the execution of AntilockBehavior. In our example a UML property AntiLock of type SharedResource is stereotyped as ≪saSharedResource≫, and then it is included in SAResources and the synchronization protocol specified.

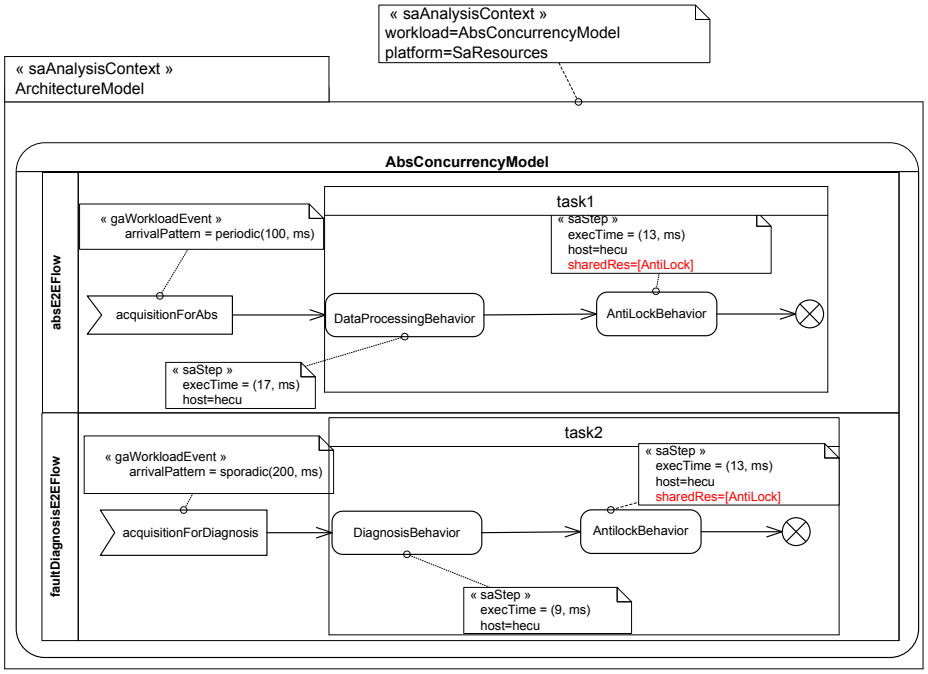

Fig. 6.5: TaskMapping

6.5.2.3 Rate-Monotonic Priority Assignment

In the example the two threads task1 and task2 are included in the platform resources for schedulability analysis SaResources along with priorities. Priorities are assigned as inversely proportional to deadlines following a rate-monotonic priority assignment [16] that is optimal in case of independent tasks and fixed-priority.

6.5.3 Schedulability Analysis Results

The Optimum model as defined in the previous subsection contains all the needed information to perform schedulability analysis. Let us note that in this case responses are constituted by one single task, tasks are independent (no order) but contain a critical section with duration equal to the execution time of the Control action (13ms). A schedulability analysis test can be carried out on this model. Such test calculates worst case response times for each event. Note that the worst-case response time include the blocking time B_i calculated by the test.

Note that in this case, the highest priority task experiences a blocking time. In fact in the worst case, the lower priority task (which has a computational time lower than the highest priority task) acquires the lock before the highest priority task. Only

Table 6.5: Schedulability analysis results

task	Bi	WRi
τ_1	13	43
τ_2	0	52

after the lock is released (after 13 ms), task τ_1 continues its execution. For task τ_2, the worst case response time takes into account, the time for the highest priority task to execute (30) and its execution time (22).

These results can be satisfactory for the designer, but he can also evaluate other alternative architectures. In that case, multiple analysis contexts can be defined. Each analysis context will contain the evaluated architecture completed by schedulability results. Among these analysis contexts, the designer will choose the one that better satisfies its own criteria.

6.6 Related Works

In order to support the development of real-time applications a wide number of methodologies have been proposed for early analysis of non-functional requirements. While a vast number of model-based approaches have been proposed for performance prediction [24], methodologies for schedulability prediction are more recent and are gaining a growing interest with the increasing complexity of embedded real-time systems [61, 25, 143]. COMET [108] proposes a methodology for the development of concurrent and distributed real time systems but does not directly deal with the issue of defining a methodology for schedulability validation. In Saksena et al [260] a methodology for schedulability validation of object-oriented models is proposed. The methodology starts from a design model where specification of active and passive objects, message semantics and object interaction is available. Two threading strategies are proposed: a single threading solution and a multi-threading solution. Unfortunately, while the single threading solution is analyzable and applicable, the multi-threading solution is difficult to analyze or inapplicable [26]. The problem with schedulability analysis at the design level is that a non schedulability aware design could have a concurrent model too difficult to analyze or for which no automated support exists. This problem is also shared by other methodologies such as [25, 116]. In [25], for instance, the UML-based methodology envisages and supports the use of task mapping algorithms and edf schedulability tests developed in [26] but there is no automated support for the test which is ad-hoc. In [151], the author has explored the usage of MARTE to perform schedulability analysis with the MAST tool. In this work MARTE has been used to build a MAST model library to build MAST-specific analysis models. Unlike this approach, our approach proposes methodological rules to build schedulability analysis models which are independent from any schedulability analysis tool.

6.7 Conclusions and Future Work

A lot of work has been achieved in formal approaches for timing analysis during the past decades. Despite the importance of applying timing analysis for real-time systems development, the formal nature of these approaches is an obstacle for their adoption by the software engineering community. This paper presented a UML/-MARTE front-end for these formal timing analysis approaches, focusing on the schedulability aspect and integrated in the software life cycle since the very beginning. The methodology is fully implemented in the UML modeling tool Papyrus [88]. Bridges to the MAST [228] and Rt-Druid [95] tools are integrated for schedulability tests. Let us remark that the methodology has been successfully applied in the automotive domain in the context of two collaborative projects, the European INTERESTED project and the French national EDONA project.