Modeling On-Board Software Dynamic Architecture: A Related Experience using UML-MARTE

Alexandre Cortier[1], Marie-Hélène Deredempt[1], Jacques Seronie-Vivien[1], Jean-François Rolland[2], François Bossard[3]

[1] Astrium SAS, Toulouse, France, {firstname.lastname}@astrium.eads.net

[2] Atos, Toulouse, France, Jean-francois.rolland@atos.net

[3] CNES, Toulouse, France, Francois.Bossard@cnes.fr

Keywords : OBSW, UML, MARTE, dynamic architecture, computational model, use case

Abstract

MARTE (Modeling and Analysis of Real-Time and Embedded Systems) is the UML extension profile dedicated to the modeling of Real-time and Embedded Systems (RTES). Standardized by the OMG, UML-MARTE is well accepted in the Model Based Driven Engineering community. However there still exists a big gap to bridge for its use in operational space projects. Some of the identified limiting factors are (1) the high density of the MARTE specification which provides thousands of defined concepts and though requires a deep investment to be correctly handled and understood, (2) the absence of methodology associated to the notation and (3) the lack of experiences relating to the use of MARTE on realistic and operational system in space domain. This paper presents an experience of using UML-MARTE to model the dynamic architecture of an operational space On-Board Software (OBSW) to make a step towards the adoption of UML-MARTE. The modeling methodology adopted in this study is illustrated by a use case based on an operational OBSW. This experience has been conducted in the scope of a R&D study founded by the CNES with the collaboration of Astrium Satellites and Atos.

1. INTRODUCTION

Objectives of dynamic architecture modeling. The expected benefits of modeling dynamic architectures are to bring capabilities of:

- Providing an unambiguous communication support and an unambiguous expression of requirements and architecture;
- Supporting the use of automated documents generation (as for instance SW-ADD document as required by ECSS standards in space domain);
- Supporting the use of analysis tools, especially schedulability and performance analysis tools;
- Supporting the use of automated code generation.

Encapsulating in a unique model the necessary information to bring these capabilities is a pre-condition for the adoption of an operational modeling methodology because it allows ensuring the coherency between documentation, analysis and generated code. In the scope of this study, we focus our effort on defining a dynamic architecture modeling strategy which allows handling the three first objectives. Although the use of automated code generation still requires additional investigations, the adopted modeling strategy and produced models have been conceived by keeping in mind this capability.

UML-MARTE. The UML-MARTE [1] notation has been chosen to support our modeling strategy for three main reasons. First, this profile extension for the modeling of real-time systems is a well-accepted OMG standard which provides support for specification, design and verification/validation stages. The second reason is the existing of Open Source modeling platforms and model editor which support customization features (Topcased platform and Papyrus editor in this study). The last reason is a more specific one and relies on the internal experience of Astrium: UML is already used in operational projects to model the static architecture of the OBSW. These models are used for automated documentation and code generation. In this context, the MARTE solution was the most adapted to ensure a consistency in terms of modeling techniques used in a global design process, and to ensure traceability of modeling artefacts.

Organisation of this paper. This paper proposes in section 2 an overview of the methodology used to model the dynamic architecture of an operational system. Section 3 presents in details the formalization of the computational model used to model the dynamic architecture. Notice that MARTE is mainly used for the description of this computational model: other sub-models and diagrams make a sporadic use of MARTE concepts. Section 4 presents how to capture the dynamic architecture of the system by instantiating the computational model entities. Section 5 briefly discuss about the use of this model for schedulability analysis. To conclude, section 6 gives some feedbacks on the use of MARTE and associated tooling.

2. MODELING METHODOLOGY OVERVIEW

The Figure 1 presents the model organisation which supports the adopted methodology.

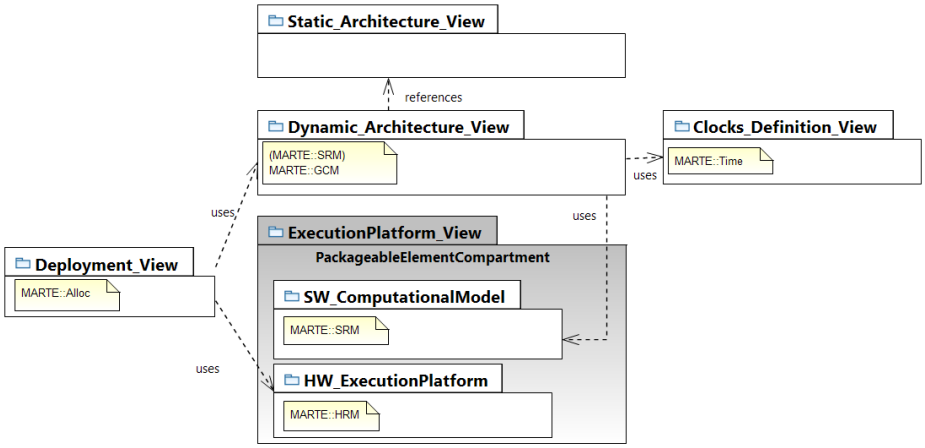

Figure 1 : Overview of the Modeling Methodology

The model is structured in 5 packages which catches different views of the system:

The Static Architecture package contains the static architecture of the on-board software. This model, formalized in term of class diagrams, has not been developed in the context of the CNES study: this model is the one operationally developed for the AS250 program. This model has been used to generate the C source code skeleton of the AS250 OBSW using the Acceleo UML2EC plugin of the Topcased platform. Some class operations of this model are referenced by the dynamic architecture via the activity diagrams which model the tasks control flows.

The Execution Platform package contains the description of the Software and Hardware parts of the execution platform:

1. **The Software Computational model** describes how computations are carried out in a system by prescribing the form of legal basic entities and their associated properties in the dimension of execution and synchronization, in time and space. The core elements defined in the computational model are instantiated in the dynamic architecture model. The MARTE *Software Resource Modeling* profile (*SRM*) is used to formalize the computational model of the AS250 OBSW.
2. **The Hardware Platform model** describes the hardware part of the system. The hardware execution platform is formalised using the MARTE *Hardware Modeling Resources* sub-profile (*HRM*). In the scope of this study, the main purpose was to evaluate the use of MARTE for schedulability analysis purpose. As the use case was very simple (mono-processor), notice that this model as well as the deployment model is not very useful here for the schedulability analysis. For this reason, this model will not be discussed in the following.

The Clocks Definition package allows modeling the full clock hierarchy of the system using the MARTE *Time* profile. The Time Expression language of MARTE allows to formally defining the clocks tree of the system by specifying the clock derivation constraints from one clock to another.

The Dynamic Architecture describes the run-time instances of the system: tasks, synchronizations, protection mechanisms for shared resources… All entities defined in the dynamic architecture are instances of the basic entities defined in the Computational Model view.

The Deployment package allocated application components on hardware platform component. This view will not be detailed in this paper.

3. COMPUTATIONAL MODEL

The modeling of dynamic architecture and its use for schedulability analysis is intrinsically related to the Computational Model on which it relies. A computational model describes how computations are carried out in a system by prescribing the form of legal basic entities and their associated properties in the dimension of execution and synchronization, in time and space, of run-time entities. A computational model is significant and useful if:

1. It can be related to analysis theories such as schedulability analysis theory for instance;

2. It can be related to a Programming model which allows to convey the implementation of the dynamic properties statically asserted by analysis and so allows automated code generation perspectives;

3. It is compliant with services offered by the operational system.

Formal analysis tools rely on well-defined computational model such as the Ravenscar profile [2] which allows performing Rate-Monotonic Schedulability Analysis (RMA). However operational project uses rarely this computational model in concrete implementation. Keeping in mind the related objective to exploit the produced models at ends of code generation, the modeling strategy adopted in this paper relies on an explicit modeling of an operational computational model. This computational model is then a re-usable resource package which defines the types of dynamic entities to be used to describe the dynamic architecture of the OBSW. UML-MARTE is principally used for this purpose.

As a consequence of this strategy, a transformation has to be defined to produce a model which can be analysed by existing analysis tool. This transformation needs to be formally defined to ensure that

the source model relying on the operational computational model is equivalent to the transformed analysis model.

3.1 DOMAIN MODEL: CORE ELEMENTS

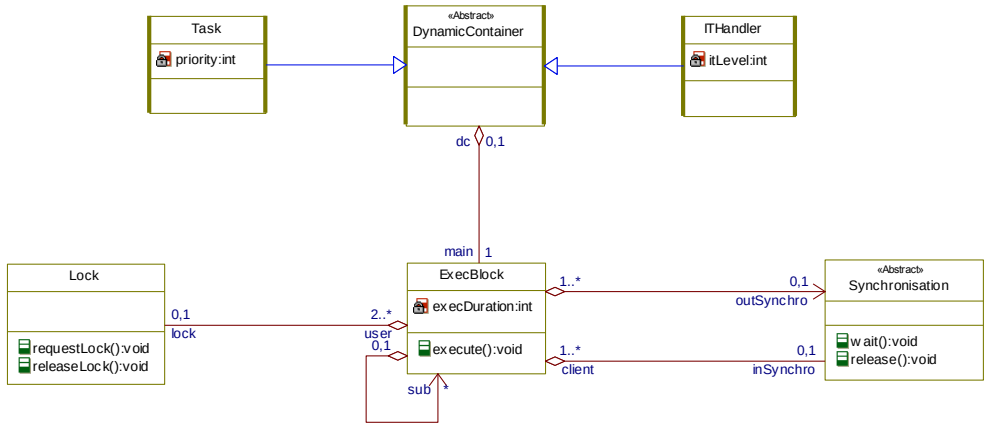

Figure 2 : Domain Model: Core Elements

The Figure 2 presents the Computational Model used for the architecture definition of the AS250 observation satellite. The *Dynamic Container* is an abstraction which describes the handle by which the RTOS schedules the various processing.

Execution Blocks. A *Dynamic Container* owned a main *Execution Block*. *Execution Blocks* are closely related to actual implementation of functional behaviour. An *Execution Block* can be decomposed in sub-blocks. Leafs of this tree structure represents a pure sequential piece of code which is characterised by its *execution duration* (WCET of the Execution Block). An *Execution Block* can wait and release *Synchronisation* through the *Wait() / Release()* interface to conform to the constraints of strict behaviour sequencing and can request and release *Locks* to access shared resources by respecting mutual exclusion constraints. In order to ease schedulability analysis, some restrictions are imposed:

- If an *Execution Block* waits for a Synchronisation, this wait is “at the beginning” of the Execution Block, before any CPU usage;
- If an *Execution Block* releases a Synchronisation, this release is “at the end” of the *Execution Block*, after all CPU usage.
- If an *Execution Block* requires use of *Lock*, it requires it for its whole duration, excluding *Synchronisation* accesses;
- If an *Execution Block* is not a *leaf Execution Block*, its CPU use is zero (all CPU use is assigned to the leaf Execution Block).

Tasks & ITHandler. *ITHandler* is a specialisation of *Dynamic Container* which have the following restrictions: it is associated which a single *Execution Block* that has limited capabilities: no *Lock* request, no “IN Synchronization” and has one “OUT Synchronisation”.

Task is the other specialisation of *Dynamic Container*. It is associated with a set of *Execution Blocks* that includes at least one “IN *Synchronization”*. *Task* is characterised by its *priority* which is used by the RTOS scheduler to determine CPU allocation in case of competition. Tasks are specialized in

Cyclic Tasks and *Asynchronous Task.* Difference is not structural but concerns the dynamic behaviour. A *Cyclic Task* has one constraint: it hosts an *Execution Block* that is controlled by a *Synchronization* linked to a periodical HW interrupt signal. A non-cyclic task is called by default *Asynchronous*.

Synchronisations. A Synchronisation is a passive object that provides two interfaces to enforce a strict sequencing between one Execution Block (taken from the set of server Execution Blocks for this synchronization) and a set of Execution Blocks clients. Synchronisations are refined in three categories: **OutAndGateSynchro** which characterised a 1 to N Synchronisation (One Server, Multiple Clients); ***ControlSynchro*** is a refinement of *OutAndGateSynchro* that provides an additional interface operation called *checkAndRelease()* which checks whether all "expected" clients are actually blocked on a wait() call when the release is performed; Finally, **InOrGateSynchro** is an N to 1 Synchronisation**.**

3.2 COMPUTATIONAL MODEL : MARTE MODEL (SRM)

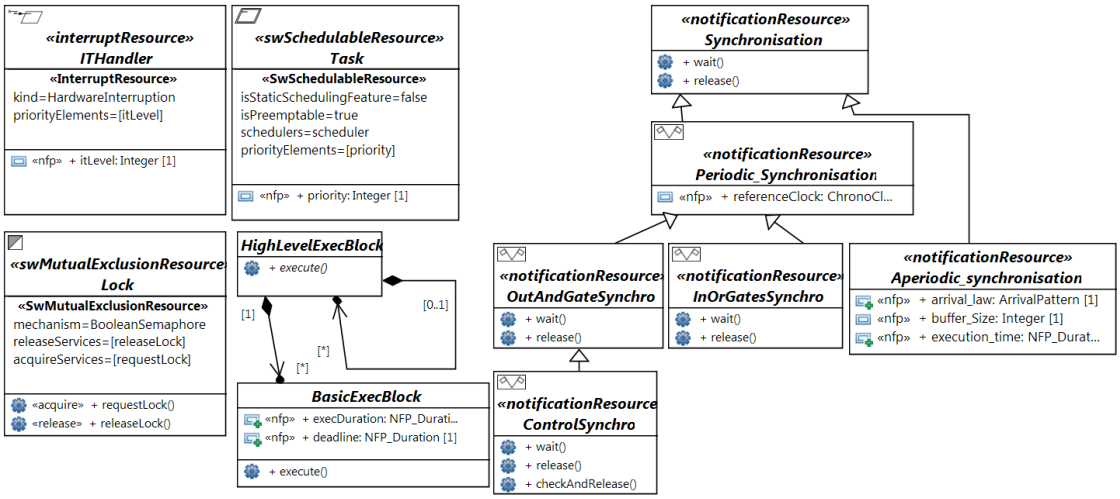

Figure 3 : Computational Model - MARTE::SRM Model

The domain model for dynamic architecture definition exposed in the previous section has been formalized using the SRM clause of *MARTE*. The SRM profile specifies a set of modeling artefacts that can be used to describe the structure of multi-tasking real-time applications built upon a real-time operating system (RTOS) [3]. More specifically it is looking to depict software resources and software services described in multi-tasking (API). The typical use of the MARTE::SRM sub-profile is the description in a unified way of software multi-tasking API in order to integrate the execution supports in the design flow.

The Figure 3 shows the translation of the Astrium Computational Model in MARTE::SRM. Each core elements of the computational model has been successfully mapped onto MARTE::SRM except the *Execution Block* concept. The stereotype << NFP>> which stands for non-functional property has been used to add the significant properties of this concept: the execution duration and the deadline of the pure sequential piece of code. The meaning of each MARTE stereotypes is not the purpose of this presentation : more information is available in the MARTE standard.

Notice that this model has to be done once and can be used as library artefact in others dynamic architecture models which relies on the same computational model.

3.3 CLOCKS DEFINITION

According to the computational model defined in the previous sections, periods of cyclic tasks are implicit in our model. This period attribute can be derived from the *referenceClock* property of the Periodic Synchronisation which releases the task.

The clock tree hierarchy of the system is formalized using the *MARTE::Time* sub-profile. The first step is to define a type of chronometric clock. Then, all clocks are derived from the ideal clock using clock constraints. The Figure 4 presents a part of the clocks hierarchy defined in the AS250 case study.

Clock constraints are formalized using the *Time Expressions* package of MARTE. For instance, the fastest 16Hz clock is defined using a clock constraint which stipulates the discretization step (62.5ms) and the stability of the clock. The other clocks (8Hz, 4Hz) are defined by under-sampling of the fastest clock. The "*aocs_avb_end_acq*" clock is defined has a 16Hz periodic clock with a 13700 us phase.

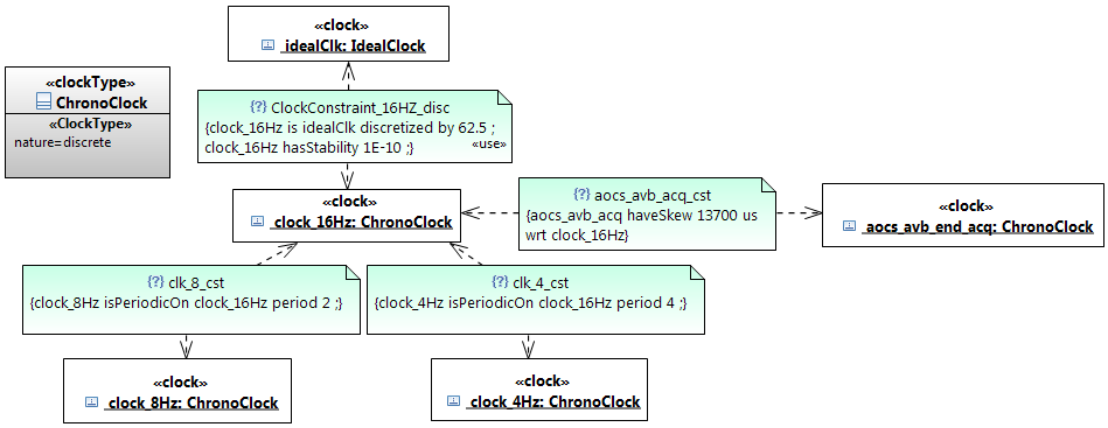

Figure 4 : Clocks definition

4. DYNAMIC ARCHITECTURE MODELING

Once the modeling of the computational model has been done, the dynamic architecture can be formalized. This model is defined in three main steps as described in the following sub-sections.

4.1.1 Identification of Execution Blocks

The first step is to identify the execution blocks of the system. High Level Execution Block (HLEB) and Basic Execution Block (BEB) are identified in a UML Class diagram using a generalisation link pointing to *HighLevelExecBlock* and *BasicExecBlock* entities defined in the computational model.

Figure 5 shows the decomposition of the Attitude And Orbit Control System (AOCS) of the AS250 OBSW. A main class *AOCS_Appli* is defined. This functional chain is composed of three HLEB. The *AOCS_CYCL_HLEB* which captures the cyclic functional behaviour of the AOCS is further decomposed in seven basic execution blocks.

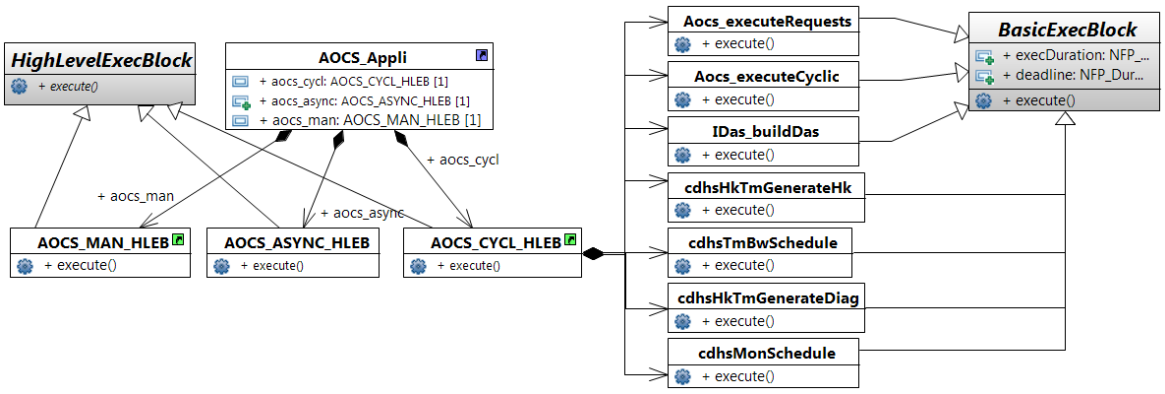

Figure 5 : Execution Blocks Definition (UML Class Diagram)

4.1.2 Instantiation

The second step is to associate a UML Composite diagram to the class which represents the considered functional chain. Figure 6 shows the composite diagram associated to the *AOCS_Appli* class.

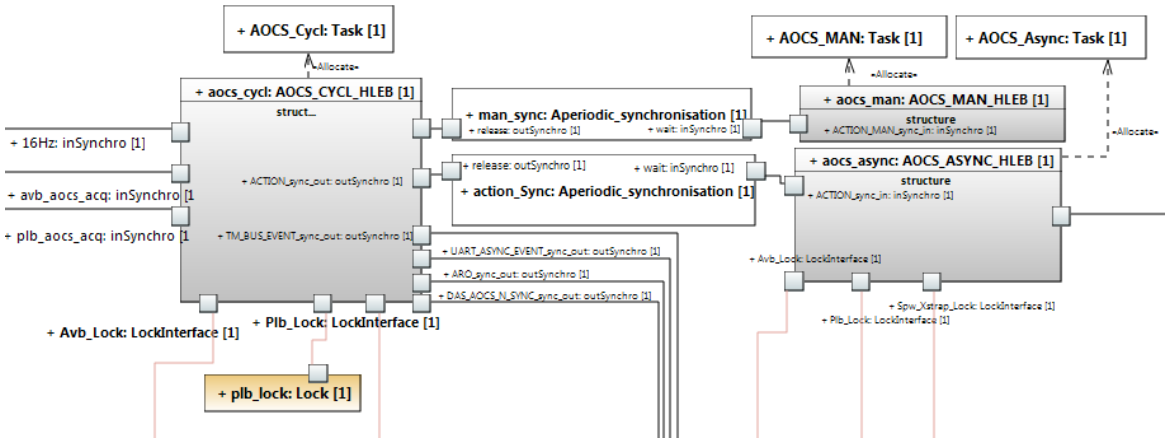

Figure 6 : Dynamic Architecture of *AOCS_Appli* (UML Composite Diagram)

At this stage synchronisations between execution blocks are defined. A synchronisation is represented as a 'part' of the composite diagram and is typed by one of the synchronization types defined in the computational model. For instance, the AOCS application chain defines two aperiodic synchronisations *man_sync* and *action_sync*. A synchronisation is composed of two client server ports, the first one is typed by the *OutSynchro* interface which provides the *release()* operation and the second one is typed by the *InSynchro* interface which provides the *wait*() operation. The use of these synchronisations by the execution blocks is represented by connectors between a client server port of the execution block and a client server port of the synchronisation instance.

Lock instances which models mutual exclusion constraints are represented in the same way.

Tasks. As stated in section 3.2, a task is linked to a high level execution block which catches its execution flow. This link is represented in the same composite diagram. In the example provided in Figure 6 three parts typed by the *Task* entity of the computational model are defined: *AOCS_cycl*, *AOCS_MAN* and *AOCS_Async*. Links between execution blocks and tasks is formally defined using an <<*Allocate*>> dependency.

Instance diagram for value specification of NFP. Although the concept of 'parts' in composite diagram maps with the concepts of instances, it is not possible to specify the value of the non-functional properties defined in the computational model. For instance, when defining *task instances* in a composite diagram we are not able to specify the value of its *priority* attribute. This UML issue in the definition of the composite diagram forced us to define an Instance Diagram to specify the values of these non-functional properties. Figure 7 presents a part of the composite diagram associated to the AOCS application chain to illustrate our concern.

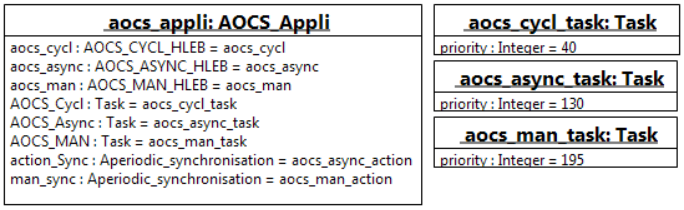

Figure 7 : Instance Diagram to specify the values of NFP

4.1.3 Execution Blocks Behaviour & link with static architecture

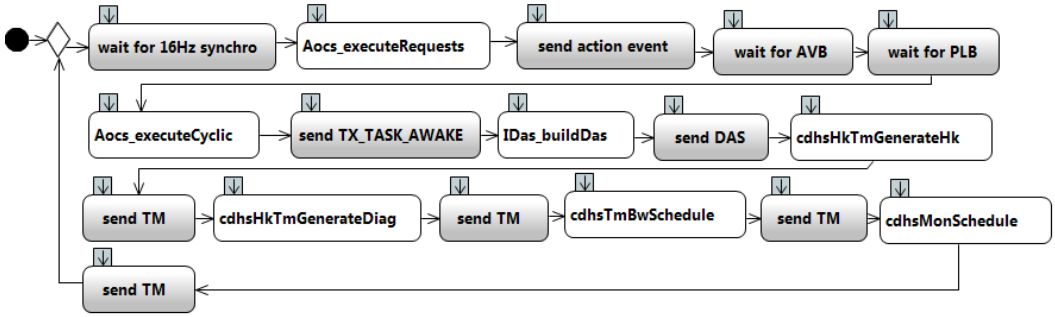

Figure 8 : Control Flow of AOCS_CYCL High Level Execution Block

UML composite diagrams specify the structure of the application in terms of execution blocks and the set of synchronizations and locks used by each of these execution blocks. The last step in the definition of the dynamic architecture is to define the concrete execution flow of each execution block. UML Activity diagrams are used for this purpose. Each execution block owned an activity which is composed of a sequence of call operation actions. A call operation in such activity diagram is either:

- a call to the *wait()* or *release()* operation of a synchronisation instance if the activity diagram stands for the execution sequence of a High Level Execution Block;
- a call to the *releaseLock()* or *requestLock()* of a lock instance if the activity diagram stands for the execution sequence of a High Level Execution Block;
- a call to the *execute*() method of an internal execution block if the activity diagram stands for the execution sequence of a High Level Execution Block;

- a call to an operation defined in the static architecture of the application. Notice that such a call is only accept in an activity diagram which stands for the execution behaviour of a Basic Execution Block.

OCL structural constraints have to be defined on activity diagrams to ensure that called operations are in the scope of visibility of the execution block as specified by the composite diagram and the structural rules previously defined. Figure 8 shows the activity diagram associated to the AOCS_CYCL High Level Execution Block.

5. SCHEDULABILITY ANALYSIS

The model presented in this article describes a dynamic architecture and temporal constraints. We want to check that these constraints can be matched by the architecture. To achieve this validation, we want to use the model as an entry point for a scheduling analysis tool. As it is a common approach different studies have proposed a mapping between MARTE and a scheduling analysis tool: from the GRM to Cheddar [4], from SRM to Cheddar [5] and from SAM to MAST [6].

In this experimentation, the starting point was an existing project with its own design rules. Some of these rules can be different from the traditional concepts of the scheduling theory. It implies specific translation rules to a scheduling analysis tool (Cheddar). Our model is has too much details, the translation act as an automatic abstraction.

Identification of tasks. In our model, tasks consist of different HLEB. HLEBs have their own activation properties and consequently our tasks can have suspension points due to a wait on event synchronization. A task in the model cannot be mapped directly on a Cheddar task. In the most general case the HLEB is the closest concept from a Cheddar task. Some specific HLEB have a synchronisation with a periodic event during their execution. In this case the block must be divided in two tasks.

Properties of Cheddar tasks. The *priority* is defined by the priority of the MARTE task on which the HLEB is allocated. The *capacity* corresponds to the sum of the WCET property of the BEB executed by the HLEB. The *deadline* is given by the deadline NFP of the last BEB executed by the HLEB. The *activation* is defined by the type of synchronisation resource that triggers the activation of an HLEB.

Shared resource. Each lock instance corresponds to a shared resource in Cheddar. The access to shared resource is represented in the model by the connectors between an HLEB and a lock instance. In Cheddar we have to set the date of begin and end of the critical section. This information is given by the activity diagram of the HLEB. The begin date correspond to the sum of each BEB WCET called before the call of the "*requestLock*" service of the lock instance. The end date is the sum of the WCET steps until the "*releaseLock*".

Processor. The different properties used to describe a processor in Cheddar (*scheduling policy, preemption*) are defined by attributes of a MARTE scheduler. The association between a task and a processor in Cheddar corresponds to the property "*schedulers*" of the *swSchedulableResource* MARTE stereotype.

6. CONCLUSIONS

Results. This paper has presented an experimentation which proposed a specific use of the UML-MARTE language to model dynamic architectures of space On-Board Software. This modeling

strategy has been successfully applied on an operational use case using the Papyrus Editor on an Eclipse Platform. The resulting use case model has been exploited for schedulability analysis purpose using the Cheddar formal schedulability analysis tool. The proposed modelling strategy relies on a selected sub-set of UML-MARTE. The cornerstone of this methodology is the use of the SRM profile of MARTE to formally specify the computational model upon which is constructed the dynamic architecture. The deep encoding of this computational model allows defining a dynamic model very closed to the actual implementation of the application software. Such detailed dynamic model can be productively used for documentation generation, schedulability analysis and we hope for code generation purpose. Even if the gain is marginal in terms of coding effort, using automatic code generation could be a real benefit in terms of quality as it ensures a strong coherency between analysis results and actual implementation.

Using UML & MARTE in Operational Context. This study shows that MARTE metamodel provides all the necessary concepts to model space OBSW dynamic architecture. However its adoption in an operational design process requires defining home guidelines to be respect by dynamic architecture designers. From Astrium point of view, the use of a home specific editor which enforces these guidelines is required to improve usability of MARTE and consequently reduce the cost of modelling activities. For instance, editors should not allow the user to use MARTE stereotypes which don't belong to the MARTE sub-set defined by the modelling guidelines and mask the access to stereotype properties which are not relevant for its concern. Notice that Papyrus provides broad capabilities of customisation which could be sufficient to design such domain specific editor.

Perspectives. The design of dynamic architecture is an iterative process. The SRM profile is probably not adequate in first stages of the design process. The use of the HLAM sub-profile for these first stages and links between such abstract model and the detailed models proposed in this paper have to be investigated. Automatic code generation using the SRM profile also needs investigation. The last perspective is to improve the tooling environment to be able to capture in the same modelling environment results coming from the formal analysis tools used for schedulability analysis.

Reference documents

[1] UML Profile For MARTE: Modeling And Analysis Of Real-Time Embedded Systems, http://www.omg.org/spec/MARTE/

[2] A. Burns, B. Dobbing, and G. Romanski. *'The Ravenscar tasking profile for high integrity real-time programs* '. Reliable Software Technologies, Proceedings of the Ada Europe, Conference, Uppsala, pages 263–275. Springer Verlag, 1998.

[3] http://beru.univ-brest.fr/~singhoff/cheddar/

[4] Eric Maes. *MARTE to Cheddar transformation using ATL.* THALES Technical Report

[5] Frédéric Thomas. *Contribution à la prise en compte des plates-formes logicielles d'exécution dans une ingénierie générative dirigée par les modèles.* Phd Thesis 2008

[6] Julio L. Medina , Álvaro García Cuesta. *From composable design models to schedulability analysis with UML and UML profile for MARTE.* Proc. of CRTS 2010. 3 rd Workshop on pompositional Theory and Technology for Real-time Embedded Systems