MAENAD

Grant Agreement 260057

Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles

Report type Deliverable D4.2.1

Report name Profile for EAST-ADL

Dissemination level	**PU**
Status	**Final**
Version number	**2.1**
Date of preparation	**2014-02-28**

Authors

Editor	E-mail
Sara Tucci-Piergiovanni	Sara.Tucci@cea.fr

Authors	E-mail
Sara Tucci-Piergiovanni	Sara.Tucci@cea.fr
Chokri Mraidha	Chokri.Mraidha@cea.fr

Reviewers	E-mail
Hans Blom	hans.blom@volvo.com
Frank Hagl	frank.hagl@continental-corporation.com

The Consortium

Volvo Technology Corporation (S) Centro Ricerche Fiat (I)

Continental Automotive (D) Delphi/Mecel (S) 4S Group (I)

ArcCore AB (S) MetaCase (Fi) Systemite (SE) CEA LIST (F)

Kungliga Tekniska Högskolan (S) Technische Universität Berlin (D) University of Hull (GB)

Revision chart and history log

Version	Date	Reason
0.1	2010-12-06	Outline
1.0	2011-08-31	Intermediate
1.1	2011-11-16	Adding sections on timing and events
1.2	2012-07-27	Correcting and updating sections on timing and events
1.3	2013-07-23	New version released for review. Includes relationships between EAST-ADL and MARTE for functional modelling, hardware modelling, verification&validation, non-functional properties speficiation. Update to the MARTE profile for Timing.
2.0	2013-09-30	Version 2.0 based on review comments and final adjustments
2.1	2014-02-12	Version 2.1, added reference to the stand-alone UML profile for EAST-ADL 2.1.12

Approval	Date
Henrik Lönn	2014-02-12

Table of contents

1 Introduction

In the previous series of projects ATESST1&2, the EAST-ADL language was implemented as a UML profile – see [1]. During these projects a study of the convergence between EAST-ADL and OMG language for modeling real time and embedded systems, MARTE, was conducted. It resulted in an annex to the MARTE specification describing how MARTE could be used to define an EAST-ADL model – see [3]. This study focused mainly on the structural description of an EAST-ADL system, in terms of hierarchical components, connectors and ports.

In the current project, MAENAD, the WT4.2 work task continues the work done to achieve a better understanding of the relationship between EAST-ADL and MARTE. For this it was decided to design a new version of the UML profile, which implements the EAST-ADL language. This implementation had the objective of explicitly connecting the stereotypes of the EAST-ADL profile to MARTE, making EAST-ADL profile de facto a sub-profile of MARTE.

After a deep investigation (herein reported) between MARTE and EAST-ADL, the initial objective of making the whole EAST-ADL profile a sub-profile of MARTE revealed to be not totally reachable. In particular it has been clear that MARTE is at lower abstraction level than EAST-ADL for many aspects, as for instance the analysis support (Verification and Validation in EAST-ADL). For other aspects as the functional modeling (component modeling in MARTE) it has been clear that the mapping between the two languages was difficult, as a set of concepts of MARTE were more specialized than EAST-ADL concepts, while another set of concepts were more specialized in EAST-ADL.

The only aspect that easily conducted to a MARTE specialization for EAST-ADL was the timing aspect. This is not surprisingly as the same group of researchers worked at both languages for the timing aspect, at the OMG for MARTE and in the Timmo-2-Use project for EAST-ADL.

This deliverable analyzes the relationship between MARTE and EAST-ADL. Section 3 presents and compares MARTE component modeling and EAST-ADL[1] functional modeling. Section 4 presents and compares MARTE hardware resource modeling and EAST-ADL hardware modeling. Section 5 presents and compares allocation handling in both languages. Section 6 presents and compares non-functional properties handling in both languages. Section 7 focuses on languages constructs offered by the two languages to support verification and validation activities, while Section 8 covers the timing aspect, presenting a possible specialization of MARTE concepts for EAST-ADL.

Even if this deliverable is devoted to the MARTE/EAST-ADL investigation, it is important to remark that a parallel work in the MAENAD project aimed at maintaining the UML profile, developed during ATESST1&2, up-to-date with respect EAST-ADL evolutions up to EAST-ADL 2.1.12. More details on this work can be found in the deliverable D5.1.1 and in a separated document containing the profile specification [6].

[1] The MARTE/EAST-ADL investigation considers the EAST-ADL 2.1.10 specification.

2 Overall strategy for investigation of mappings between EAST-ADL and MARTE

The starting point for the investigation of the mapping between EAST-ADL and MARTE is the study done in previous projects, which resulted in the EAST-ADL annex to MARTE (see [3]) in June 2009. However several changes in the EAST-ADL language have occurred in the mean time, which makes the annex slightly outdated, although still valid in essence. The reminder of this chapter will provide an update of this mapping study.

The overall strategy for going further in the EAST-ADL/MARTE mapping study consisted in analyzing the different aspects covered by these two languages separately. A first coarse-grained mapping between EAST-ADL packages (giving an organization for EAST-ADL constructs) and MARTE sub-profiles (giving an organization for MARTE stereotypes) has been established. In this respect Table 1 presents the main EAST-ADL packages [4], and the correspondence with MARTE sub-profiles [3], if such correspondence there exists.

EAST-ADL package	MARTE sub-profile(s)
System Modeling (from Structural Constructs)	*None*
Feature Modeling (from Structural Constructs)	*None*
Vehicle Feature Modeling (from Structural Constructs)	*None*
Function Modeling (from Structural Constructs)	**Generic Component Model (GCM), Allocation Modeling (Alloc)**
Hardware Modeling (from Structural Constructs)	**Hardware Resource Modeling (HRM), Allocation Modeling (Alloc), Generic Resource Modelling (GRM)**
Environment (from Structural Constructs)	*None*
Behavior (from Behavioral Constructs)	*None*
Variability (from Variability)	*None*
Requirements (from Requirements)	*None*
Use Cases (from Requirements)	*None*
Verification Validation (from Requirements)	**Generic Quantitative Analysis Modeling (GQAM), Non Functional Properties (NFPs)**
Timing (from Timing)	**Time Modeling (Time)**
Timing Constraints (from Timing)	**Time Modeling (Time)**
Events (from Timing)	**Time Modeling (Time)**
Dependability (from Dependability)	*None*
Error Model (from Dependability)	*None*
Safety Constraints (from Dependability)	*None*
Safety Requirement (from Dependability)	*None*
Safety Case (from Dependability)	*None*

Generic Constraints	Non Functional Properties (NFPs)

Table 1 : EAST-ADL packages and corresponding MARTE sub-profiles

As can be noted, the only EAST-ADL packages that have a correspondence with MARTE sub-profiles are those packages related to logical design, platform modeling, allocation, timing and analysis. This comes from the fact that the two languages have very different scopes. EAST-ADL is intended to be a system-level language, covering aspects as features, variability, requirements and safety analysis. MARTE is intended to be used for the software design of real-time embedded systems, providing concepts for detailed software and hardware real-time architecture designs, timing and performance analysis. For sake of completeness Table 2 shows the main MARTE sub-profiles and the correspondence (if any) with EAST-ADL packages.

Let us remark that the correspondence shown in the two tables is a coarse-grained correspondence, i.e. it may be the case that some concept in an EAST—ADL package is not covered in the corresponding MARTE sub-profile and vice versa. It may be also the case that two similar concepts can be found but one concept may represent a specialization of the other one, a generalization or (even being similar) neither a generalization nor a specialization. The detailed relationships between language concepts will be presented in the next sections. More in detail, we focus on the mapping between EAST-ADL constructs to MARTE stereotypes (focusing then on correspondences shown in Table 1), studying possible mappings if similar concepts can be found and highlighting relationships (e.g. specialization, generalization, equivalence) between similar concepts. Let us note that the initial objective was to design a MARTE profile for EAST-ADL, i.e. to find a specialization of MARTE concepts to represent EAST-ADL constructs. Let us remark that this is possible only if the used MARTE concept is more general than the EAST-ADL construct. We will see that this relationship is true only for timing concepts, *keeping the idea of a MARTE profile for EAST-ADL valid for the Timing package.*

MARTE sub-profile	EAST-ADL package(s)
Non-functional Properties Modeling (NFPs)	*No specific package, only Generic Constraints in EAST-ADL but no specific modeling of NFPs*
Time Modeling	**Timing**
Generic Resource Modeling (GRM)	*None*
Allocation Modeling (Alloc)	**Functional Modeling, Hardware Modeling**
Generic Component Model (GCM)	**Functional Modeling**
High-level Application Modeling (HLAM)	*None*
Software Resource Modeling (SRM from Detailed Resource Modeling)	*None*
Hardware Resource Modeling (HRM from Detailed Resource Modeling)	**Hardware Modeling**
Generic Quantitative Analysis Modeling (GQAM)	**Verification Validation** (from Requirements)
Schedulability Analysis Modeling (SAM)	*None*
Performance Analysis Modeling (PAM)	*None*

Table 2 MARTE sub-profiles and corresponding EAST-ADL packages

3 Functional modeling and MARTE GCM

As already pointed out, the functional modeling constructs are organized in EAST-ADL in a single

Modeling purpose	MARTE subprofile
Non-functional Property Specification	Time, NFPs
Logical Design	HLAM, GCM,
Verification and Validation (Performance, Schedulability Analysis)	GQAM, SAM, PAM,
Platform Design	SRM, HRM and GRM

Table 3 MARTE sub-profiles and modeling purposes

package (in its turn sub-package of Structural Constructs). In order to find good candidates in MARTE for the mapping of functional modeling constructs, we should first inspect those sub-profiles whose modeling purpose is the modeling of logical designs. Table 3 shows modeling purposes of the different MARTE sub-profiles. From the table, only two sub-profiles pursue the logical modeling purpose, i.e. GCM and HLAM. HLAM, however, is not relevant in the context of EAST-ADL as it used to model object-oriented execution semantics. For this reason, we will select only GCM for the mapping of functional modeling constructs.

3.1 GCM in a nutshell

The Generic Component Model (GCM) of MARTE offers rich semantics for component modeling, enabling various models of computation and communication.

A MARTE Component is a simple UML Class, i.e. UML has not been specialized to represent a MARTE Component, but simple UML Classes are used.

On the other hand the UML Port has been specialized to represent two different MARTE specializations: FlowPort and ClientServerPort. Figure 1 presents the MARTE profile for ports.

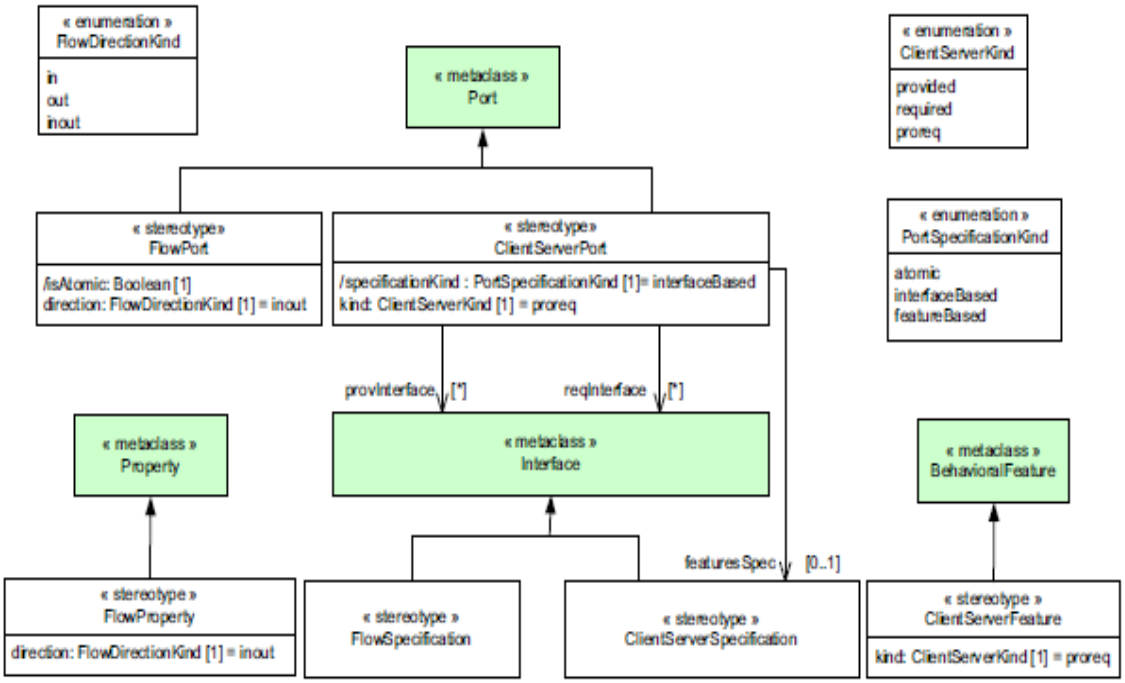

Figure 1 MARTE Profile for GCM Ports

FlowPorts

FlowPorts have been introduced to enable data flow-oriented communication between components, where messages that flow across ports represent data items. A flow port specifies the input and output items that may flow between a structured component and its environment. The specification of what can flow is achieved by typing the flow port with a specification of items that may flow along the ports and their connectors. This can include typing an atomic flow port with a single type representing the items that flow in or out as shown in Figure 2, or associating the FlowPort with a set of FlowProperties, where each FlowProperty has its own direction, which represent the properties of a FlowSpecification of an item that flows, as shown in Figure 3.

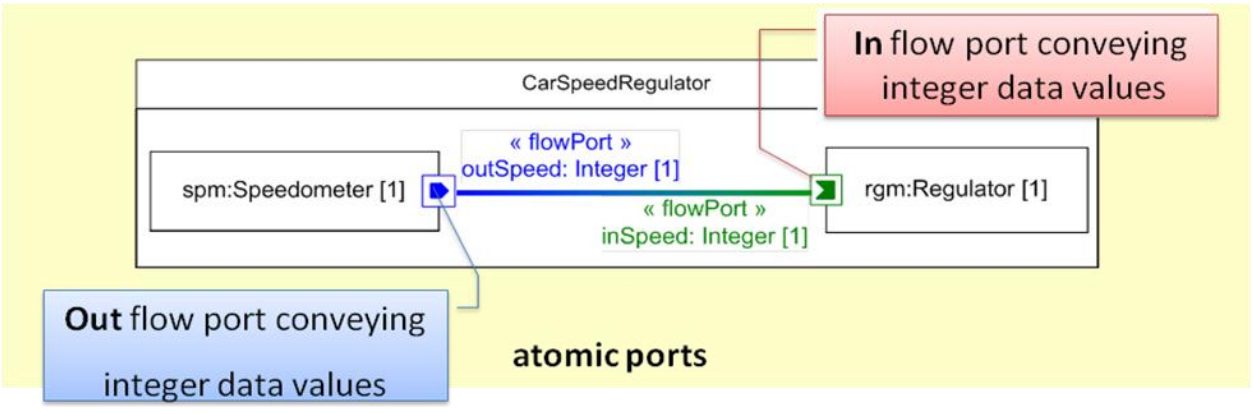

Figure 2 GCM atomic flow ports

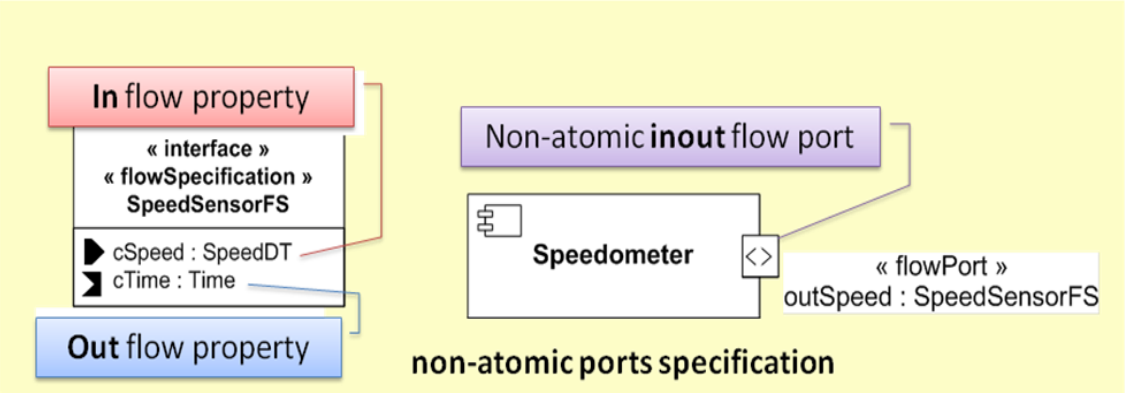

Figure 3 GCM non-atomic flow ports

There are in GCM two ways of specifying data-flow communications semantics:

- The **pull form** of the data flow semantics with the following characteristics:
  - Passive: the arrival of data in the data store does not trigger behaviors per se. It is indeed additional actions, for example time-triggered actions, that when needed pull the data from the data store.
  - Non-depleting: the use of data in the store does not remove it from the store.

  The way of modeling the pull semantics in GCM is shown in Figure 4. A delegation connector between the port and the inner property currentSpeed of Regulator component (in the figure it is graphically represented as a property typed Regulator), establishes the link between the Regulator FlowPort and the data store currentSpeed. Let us note that the data store is typed by the type of data items that circulate through the port (Integer data items in this case). The size of the data store is specified by its multiplicity. By default a data store has size equal to one. Let us note that if multiplicity is greater than one, then it is possible to specify the order of items in the data store, thanks to the <<dataPool>> stereotype applied to the data store property.

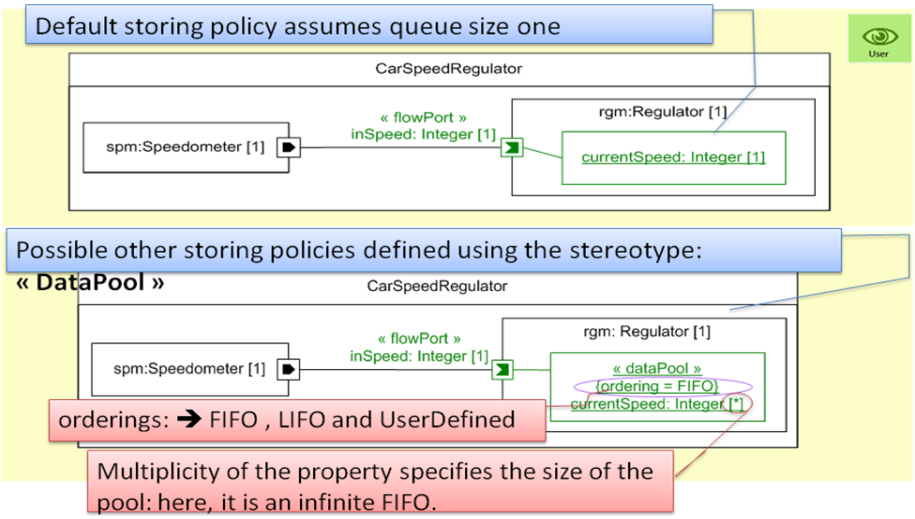

Figure 4 GCM flow ports - pull semantics

As already mentioned, for the pull semantics, the arrival of data in the store does not trigger any particular behavior per-se. Additional actions have to be explicitly modeled in order to consume data from the data store. A time-triggered activation is depicted in Figure 5. In this case the Tick signal is received at the tick port. Tick (generated by a clock not explicitly modeled in the figure) triggers the activation of the state machine Behavior. When Tick is received the Update activity is invoked. The Update activity is composed by a ReadStructuralFeatureAction with structuralFeature=currentSpeed, i.e. a read from the data store currentSpeed.

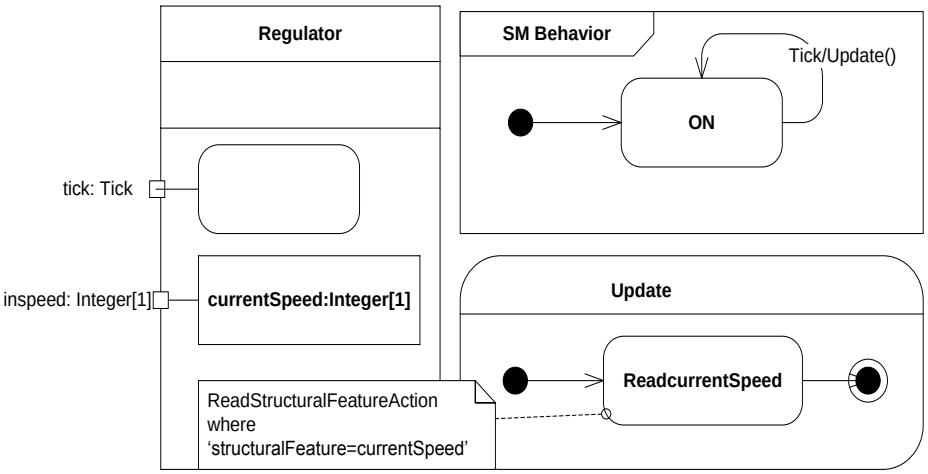

Figure 5 Time-triggered activation

- The **push form** of the data flow semantics, with the following characteristics:
  - Active: the arrival of data in the data store triggers execution of some behavior.
  - Depleting: the data arriving on the port is not stored locally. Data is indeed conveyed to the triggered behavior.

  Figure 6 shows an example of push semantics. In this case the inSpeed port is a behavioral port conveying data to the activity (Classifier behavior for Regulator) DataDrivenClassifierBehavior. Semantics of the push version of the inSpeed flow port dictates that a data event is raised each time the data is received. This event triggers the execution DataDrivenClassifierBehavior. Because the AcceptEventAction of this activity owns a trigger for the raised data event, i.e. an event stereotyped <<DataEvent>> whose 'classifier=Integer'.

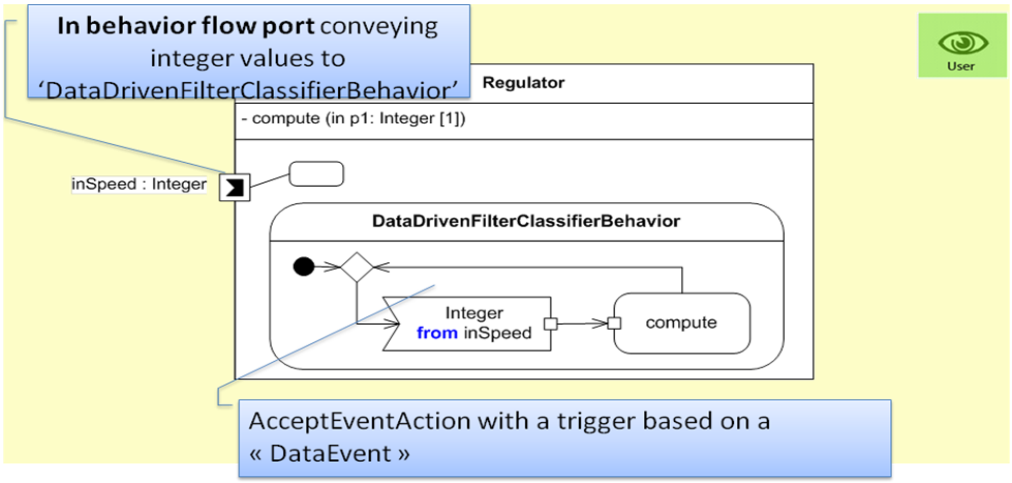

Figure 6 GCM Flow Ports - push semantics

ClientServerPorts

ClientServerPorts support a request/reply communication paradigm (also called client-server model of communication), where messages that flow across ports represent operation calls or signals. A ClientServerPort can have PortSpecificatioKind to featureBased (see Figure 1). In this case, we have a clientServerSpecification on interface (<<clientServerSpecification>> stereotype applied on an Interface). Each Operation/Signal can be either provided or required using ClientServerFeature. When the ClientServerFeature is an Operation, it represents a service that the owning structured component may provide and/or require via this port. In the case of a Reception, it represents a signal that they may publish (in this case, we consider the feature is required) and/or consume (in this case, we consider the feature is provided) via this port. Just like flow ports, a client server port can be atomic (i.e., /isAtomic = true). In this case, the ClientServerPort has no features, and the port is directly typed (via its attribute type inherited from Foundations::Property) by the signal it may produce and/or consume (with respect to its attribute kind).

3.2 Functional Modeling in a nutshell

Figure 7 shows an excerpt of the EAST-ADL domain specification for Ports. Three different ports specialize here abstract FunctionPort, namely FunctionFlowPort, FunctionClientServerPort and FuntionPowerPort. FunctionFlowPort is used for data-flow communication, while FunctionClientServerPort supports client/server communication via operation calls. FunctionPowerPort is a FunctionPort for denoting the physical interactions between environment and sensing/actuation functions.

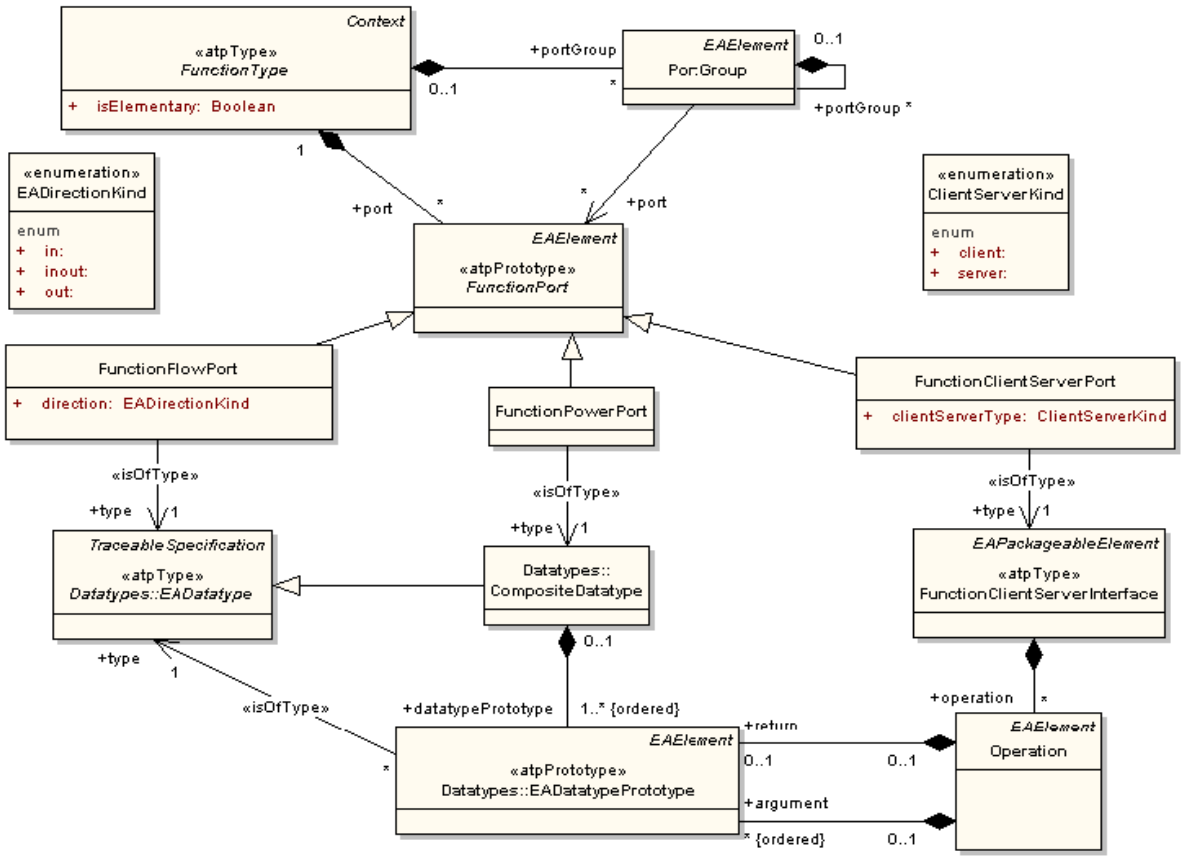

Figure 7 Ports

As for the semantics of activation through ports, EAST-ADL provides each FunctionType with a FunctionBehavior. The way in which the behavior is activated is specified through events specified directly for functions or for their ports. EventFunction is an event relevant for the activation of the function, while EventFunctionFlowPort and EvenFunctionClientServerPort specify events on ports.

These events are used in conjunction with FunctionTrigger to define the semantics of activation of the FunctionBehavior (see Figure 8 and Figure 9). A FunctionTrigger represents the triggering parameters necessary to define the execution of a FunctionType or FunctionPrototype. Triggering is either time-driven (triggering kind equal to TIME) or event-driven (triggering kind equal to EVENT). Let us note that regardless of the type of activation, once activated, the function execution follows the following semantics: read of all input ports, execute behavior with fixed inputs (run-to-completion), write on output ports.

In the case of event-triggered activation, the port association for FunctionTrigger specifies the FunctionPorts that are referred to in the FunctionTrigger (if any). EventFunctionFlowPorts and EventFunctionClientServerPorts are used in conjuction with FunctionTrigger with triggerPolicy set to Event. In this case the FunctionBehavior is executed upon the arrival of data/request on ports.

Let us note that ports for a FunctionTrigger are specified only if the triggering kind is set to EVENT.

In case of time-triggered activation FunctionTrigger points to the EventFunction of the function and defines a triggerPolicy set to TIME. The timing constraint associated to the EventFunction provides information about the period.

Let us note that the time-triggered activation in EAST-ADL can only be expressed for the function, i.e. no two different frequencies can be defined for triggering function activation in time-triggered mode.

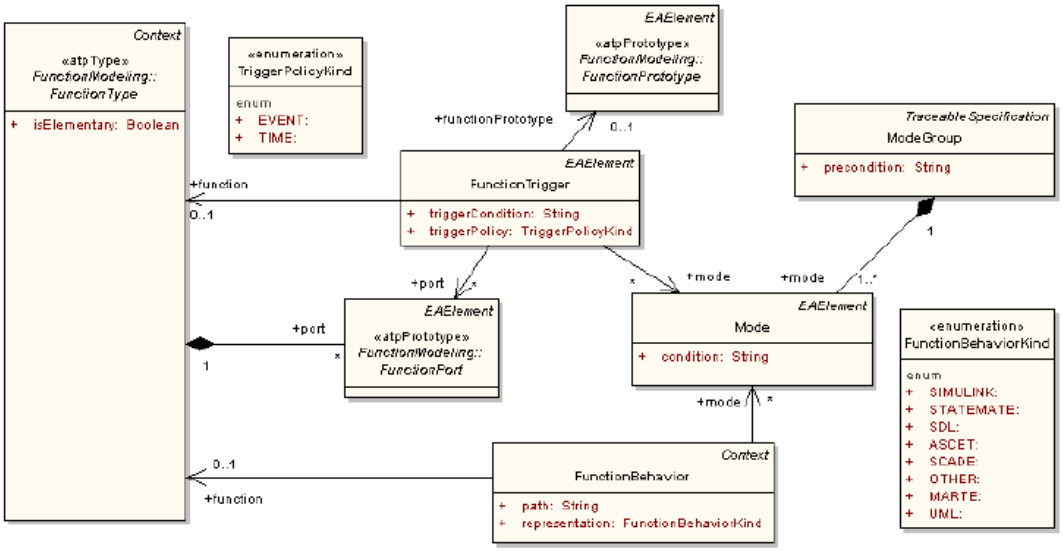

Figure 8 FunctionTrigger and FunctionBehavior

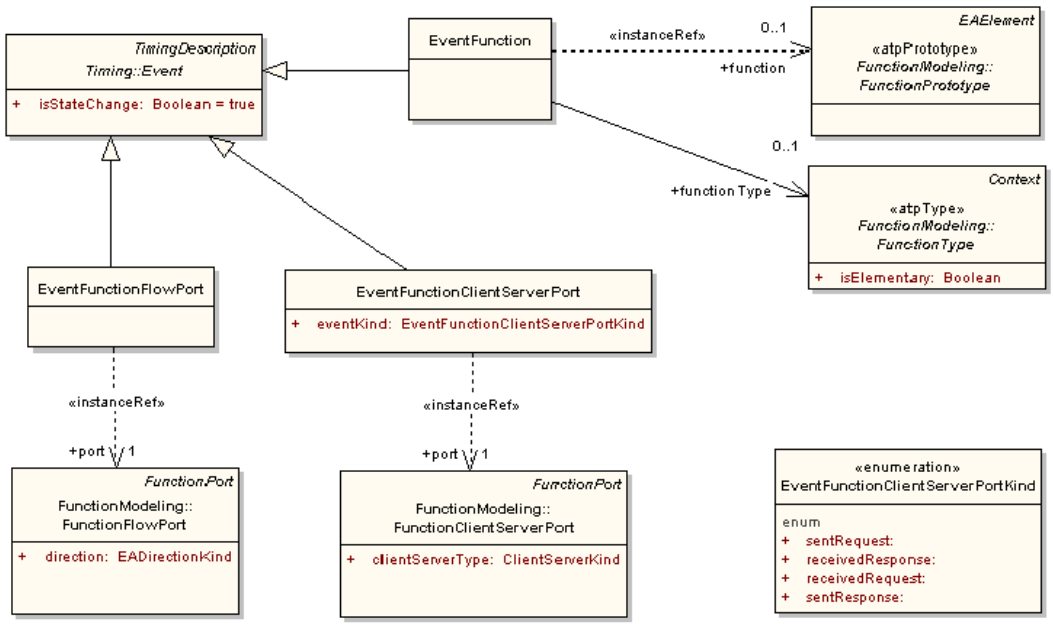

Figure 9 Events for Ports

3.3 GCM and Functional modeling comparison

In this section we discuss similarities and differences between EAST-ADL and GCM MARTE.

Inspecting Figure 1 and Figure 7 we observe that FunctionFlowPort could have, as natural corresponding concept in GCM, the FlowPort stereotype. However, FlowPort of GCM supports non-atomic flow ports, in which a flow specification can define multiple flows with possible different directions. This is not possible for FunctionFlowPort. This fact implies that FunctionFlowPort cannot be seen as a specialization of FlowPort. In practical terms, the isAtomic attribute will be inherited by the FunctionFlowPort if represented as specialization of FlowPort, which will violate the EAST-ADL domain model specification.

A similar reasoning can be applied to FunctionClientServerPort and GCM ClientServerPort. In this case ClientServerPort supports feature-based specification that (as in the case of FlowSpecification) can define multiple Operation/Signal of different kind (provided and required).

As for FunctionPowerPort, no correspondent concept in GCM can be found. FunctionFlowPort can then only see as specialization of UML Port.

Another interesting point is the comparison of activation semantics. In MARTE pull and push semantics for ports are very flexible but pull semantics have to be explicitly modeled (e.g. time-triggered pulling of the data store). In EAST-ADL push semantics can be expressed with FunctionFlowPorts and FunctionClientServerPort, while pull-semantics is only restricted to time-triggered activation of the whole function.

In summary, both languages provide constructs able to model a functional architecture with flow and service oriented communication, with both time and event-triggered activation patterns for function/component execution. In this respect EAST-ADL is more restrictive as it limits the execution semantics to synchronous and run-to-completion executions. In MARTE execution semantics is a variation point and needs to be explicitly modeled.

3.4 Functional modeling as specialization (profile) of plain UML

In this section we present EAST-ADL specializations for the main elements of the EAST-ADL language which concern functional modeling. As stated in Section 3.3, even if the two languages provide similar concepts, EAST-ADL concepts cannot be seen as specializations of MARTE ones. For this reason EAST-ADL concepts can only be specialized from UML (higher-level) concepts. The following table presents UML specializations for EAST-ADL Functional modeling.

EAST-ADL concept	Description	UML concept	MARTE stereotype
FunctionType	It is an abstract concept, with concrete subtypes appearing in various levels (e.g. AnalysisFunctionType, DesignFunctionType etc.) It is the functionality provided by a car on that level.	Class	None: The stereotype FunctionType is introduced.
FunctionPrototype	It is an abstract concept, with concrete subtypes appearing in various levels (e.g. AnalysisFunctionPrototype, etc.) Appear as parts of FunctionTypes and are typed by a FunctionType. This allows for a reference to the occurrence of a FunctionType when it	Part	None: uses the plain UML2 part concept. A FunctionPrototype will be represented as a property typed by a FunctionType.

	acts as a part.		
FunctionPort	The FunctionPort is an abstract port for data-flow or client-server interaction, which has several concrete subtypes	Port	None : uses UML2 port concept
FunctionFlowPort	The FunctionFlowPort represents a port that exchanges data. An EADirectionKind attribute specifies the direction of the flow (in, inout, out). The associated EADatatype specifies the type of data. FunctionFlowPorts are single buffer overwrite and non-consumable.	Port	None : uses UML2 port concept.
FunctionPowerPort	The FunctionPowerPort is a concrete port for denoting the physical interactions between environment and sensing/actuation functions, it essentially features CompositeDatatype as type in which two variables (across and through) represent the physical variables exchange	Port	None : uses UML2 port concept
FunctionClientServerInterface	The FunctionClientServerInterface is used to specify the operations in FunctionClientServerPorts.	Interface	None : uses UML Interface concpet
Operation	Operation features a list of EADatatypePrototype for arguments and one optional additional return parameter.	Operation	None: uses the plain UML2 operation concept
FunctionClientServerPort	FunctionClientServerPort is a port for client-server interaction. An attribute clientServerType:ClientServerKind defines the type of exchange (client or server). The port is typed by a FunctionClientServerInterface, which provides the signature of the operations available or requested by the port.	Port	None : uses UML2 port concept
FunctionConnector	The FunctionConnector connects a pair of FunctionFlowPorts with matching types and opposite directions or a pair of FunctionClientSeverPorts, with matching FunctionClientServerInterfaces and opposite directions	Connector	None: uses the plain UML2 Connector.
AnalysisFunctionPrototype	A concrete FunctionPrototype to model the internal structure of a composite AnalysisFunctionType at Analysis level. It is typed by an AnalysisFunctionType.	Part	None: uses the plain UML2 part concept. An AnalysisFunctionPrototype will be represented as a property typed by an AnalysisFunctionType.
AnalysisFunctionType	A concrete FunctionType at Analysis level, which can be decomposed with several AnalysisFunctionPrototypes.	Class	None: The stereotype AnalysisFunctionType is introduced.
DesignFunctionPrototype	A concrete FunctionPrototype to model the internal structure of a composite DesignFunctionType at Design level. It is typed by a	Part	None: uses the plain UML2 part concept. A DesignFunctionPrototype will be represented as a property typed by an

	DesignFunctionType.		DesignFunctionType.
DesignFunctionType	A concrete FunctionType at Design level, which can be decomposed with several DesignFunctionPrototypes.	Class	None: The stereotype DesignFunctionType is introduced.
BasicSoftwareFunctionType	A subtype of DesignFunctionType to represent a middleware functionality at Design level.	Class	None: The stereotype BasicSoftwareFunctionType is introduced.
HardwareFunctionType	A subtype of DesignFunctionType to represent the transfer function for the identified HardwareComponentType or a specification of an intended transfer function.	Class	None: The stereotype HardwareFunctionType is introduced.
LocalDeviceManager	A subtype of DesignFunctionType to represent the functional interface to sensors.actuators and other devices.	Class	None: The stereotype LocalDeviceManager is introduced.
PortGroup	The PortGroup is used to collapse several ports to one. All ports that are part of a port group are graphically represented as a single graphically collapsed to a single line.	None	None

4 Hardware modeling and MARTE HRM

As already pointed out, the hardware modeling constructs are organized in EAST-ADL in a single package (in its turn sub-package of Structural Constructs). In order to find good candidates in MARTE for the mapping of hardware modeling constructs, we should first inspect those sub-profiles whose modeling purpose is the modeling of platform designs. Table 3 shows that three sub-profiles pursue the platform design modeling purpose, i.e. SRM, HRM and GRM. SRM, however, is not relevant in the context of EAST-ADL as it used to model real-time operating systems. GRM is more general and does not distinguish from software and hardware resource. For this reason, we will select only HRM for the mapping of hardware modeling constructs and we will explore GRM only for its possible specialization towards EAST-ADL.

4.1 HRM in a nutshell

The hardware resource modelling sub-profile provides constructs to describe the structure of hardware platforms. The Deployment package of UML specifies constructs like DeploymentTarget, Node, or Device, which can be used to define roughly a hardware architecture that is to serve as the target of software artifacts. MARTE scope is larger, as MARTE aims at covering many aspects such as:

- Software design and allocation using a high level hardware description model of the targeted hardware architecture, with some details about available resources, instruction set family, memory size. Such model is a formal alternative to block diagrams.
- Analysis and simulation of a specialized hardware description model. Let us note that the nature of details depends on the analysis focus and the simulated resources. For example, schedulability analysis requires details on the processor throughput, memory organization, and communication bandwidth; whereas, power analysis will focus on power consumption, heat dissipation, and the layout of the hardware components. Beside the nature of models (targeting schedulability or power consumption analysis), their level of details depends on the analysis and simulation accuracy. The performance simulation needs a fine description of the processor micro-architecture and memory timings; whereas, many functional simulators simply require entering the instruction set family.

As shown in Figure 10, the Hardware Resource Model is composed of two views: a logical view that classifies hardware resources depending on their functional properties, and a physical view that concentrates on their physical properties.

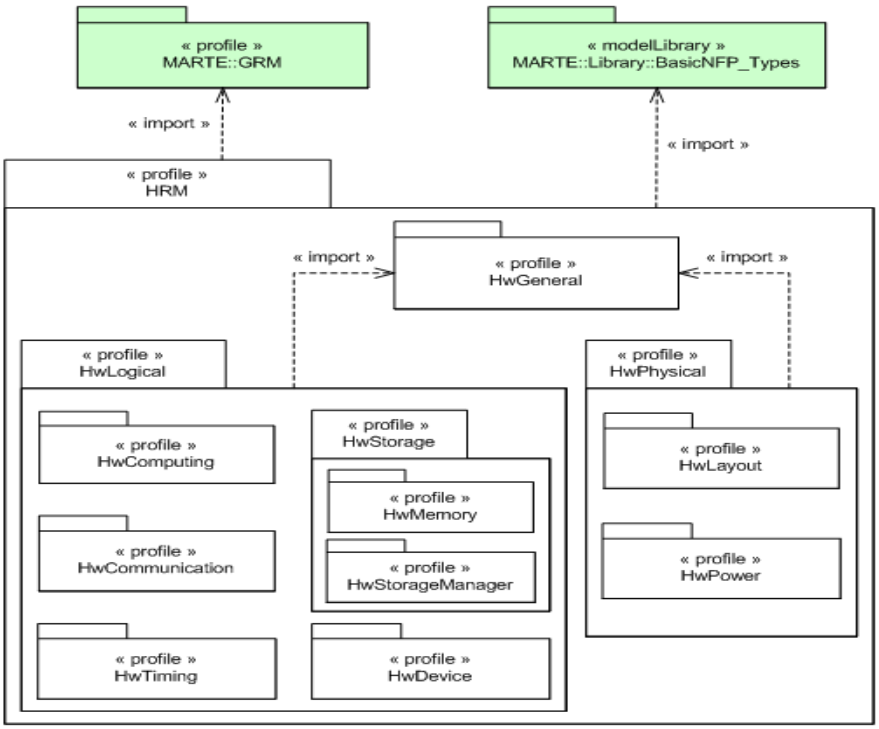

Figure 10 Hardware Resource Model profile structure

More in details,

The objective of the **logical modeling** is to provide a functional classification of hardware entities, whether they are computing, storage, communication, timing, or device resources. Such a classification is mainly based on services that each resource offers and optionally influenced by the resources nature. Figure 11, Figure 12,

- Figure 13 present respectively profile details for computing, communication resources and devices.

    1. HwComputing profile defines a set of active processing resources that are central to execution platforms (Figure 11). HwComputingResource is a generic resource. It could be specialized (HwASIC), such resources are known to be efficient but not flexible. It could be configurable (HwPLD), there are many technologies that have different capabilities like dynamic reconfiguration (SRAM). And it could be programmable (HwProcessor), which typically implements some instruction sets, owns caches, corresponding memory management units, and adopts branch prediction policies.

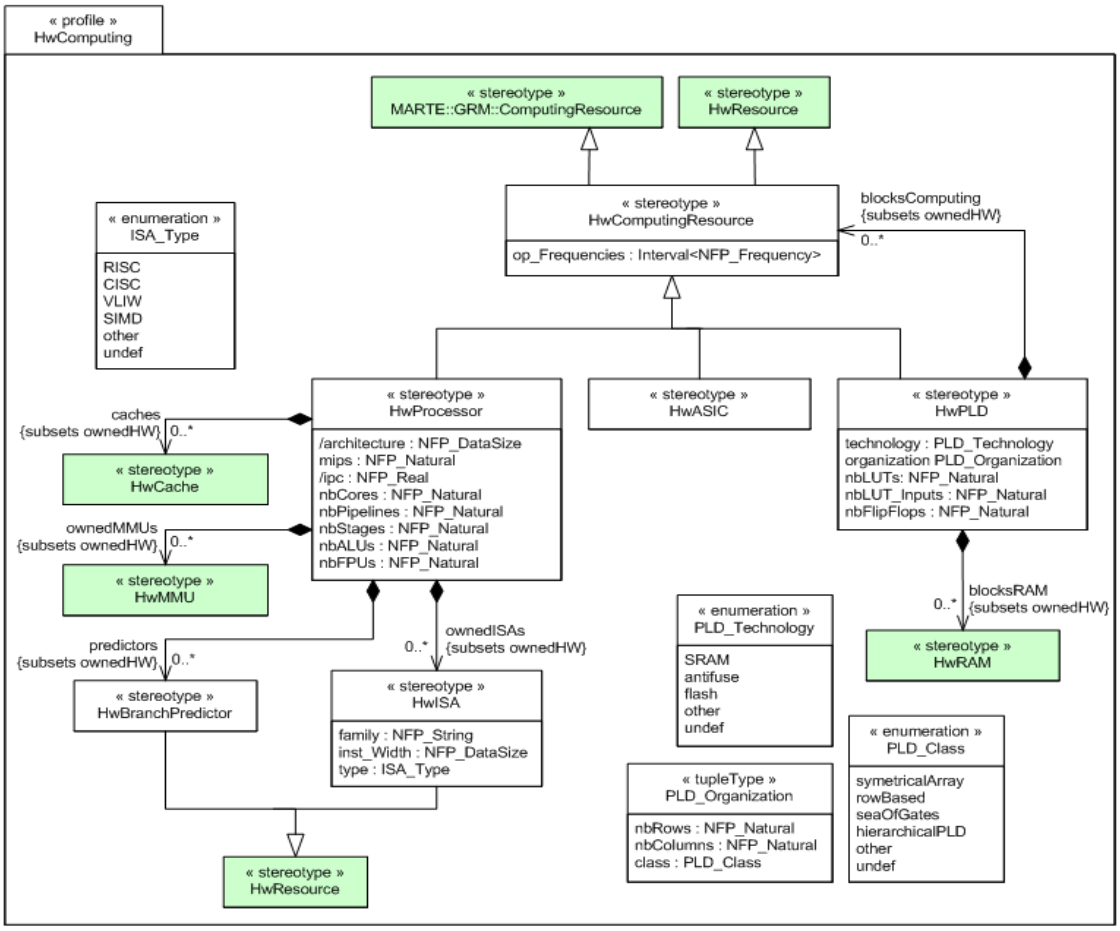

Figure 11 HwComputing profile details

2. HwCommunication profile groups all communication participants within a functional taxonomy (Figure 12). The HwMedia is a central concept that denotes a communication resource able to transfer data with a theoretical bandwidth. It may link many HwEndPoint(s). It could be controlled by many HwArbiters and it may be connected to other HwMedias by means of HwBridges. An HwEndPoint is an identified connection point of an HwResource (e.g., pin, port, or slot). If HwMedia is generic and symbolizes any kind of connections, HwBus is a particular wired channel with specific functional properties.

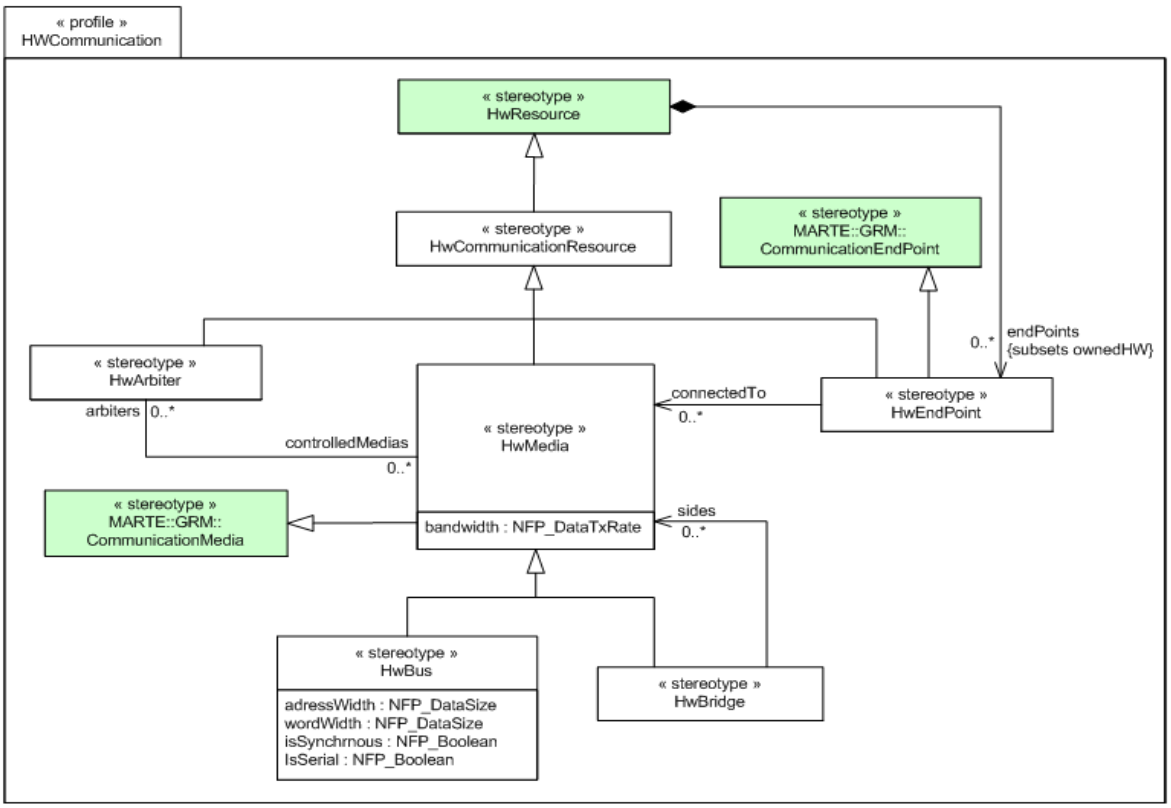

Figure 12 HwCommunication profile Details

3. HwDevice profile groups auxiliary resources that are not as fundamental as computing, storage,and communication resources are, but they expand the functionality of the hardware. It has two subcategories as shown in
4. Figure 13. The HwI/O denotes resources that interact with the environment, like sensors, actuators, peripherals, displays, external port, and so on (these specializations are not shown in the figure). Whereas, the HwSupport is a support resource like power suppliers (batteries), power regulators, cooling fans, or miscellaneous electronic devices. Because of their nature, some support devices are detailed in the physical model.

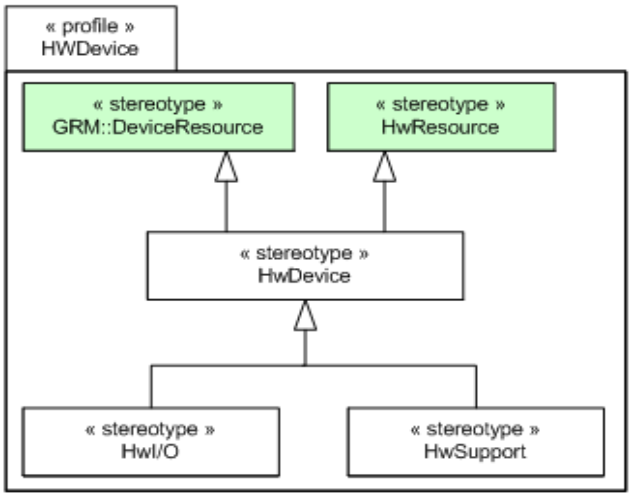

Figure 13 HwDevice profile details

- The objective of **physical modeling** is to represent hardware resources as physical components with details on their shape, size, position (within platform), power consumption, heat dissipation, and many other physical properties. It is organized in two sub- profiles (see Figure 14 and Figure 15):

    1. HwLayout provides mechanisms to make UML graphical diagrams as close as possible to the real hardware platform layout. It classifies hardware components depending on their forms and offers arrangement constructs using rectilinear grids. HwComponent denotes a generic physical component that can be refined into a grid of subcomponents. It has dimensions, a resulting area, a particular weight, and optionally a number of pins and a position within a potential container. Each HWComponent requires some environmental conditions whether if it is in use or not.

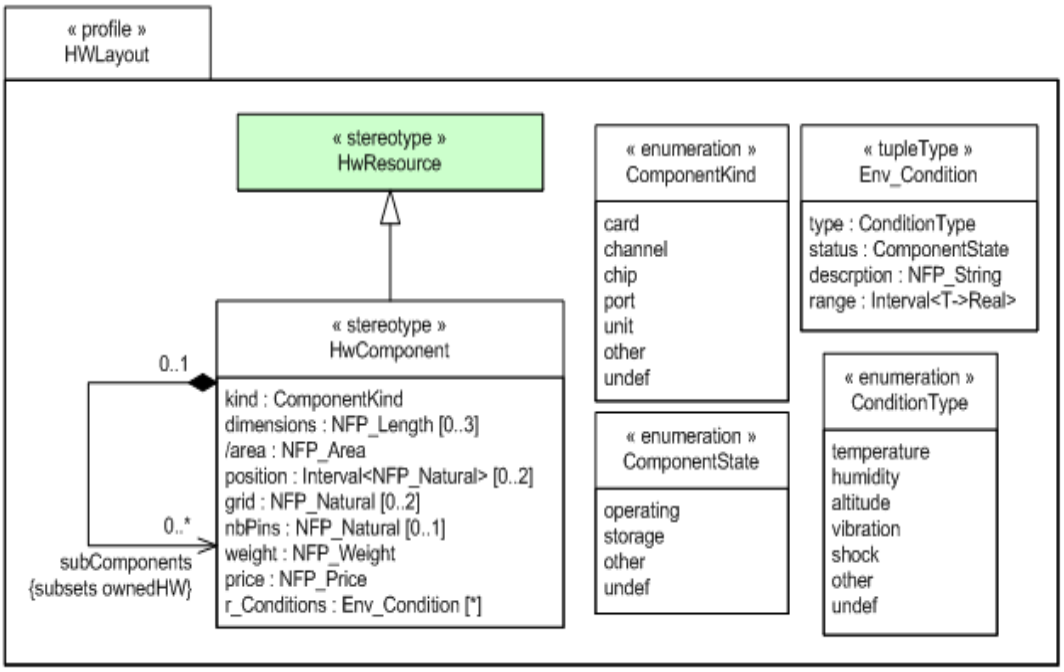

Figure 14 HwLayout profile details

    2. HwPower comes with a detailed description of HwComponent power consumption and heat dissipation. It enables advanced power analysis and autonomy optimization that are crucial for embedded systems. Notice that the HwLayout may also influence the power analysis. HwResourceService is a key stereotype that provides instantaneous power descriptions: consumption and leakage at non-operating time. HwPowerSupply is an energy suppliers, whereas HwCoolingSupply is a heat reducer.

3.

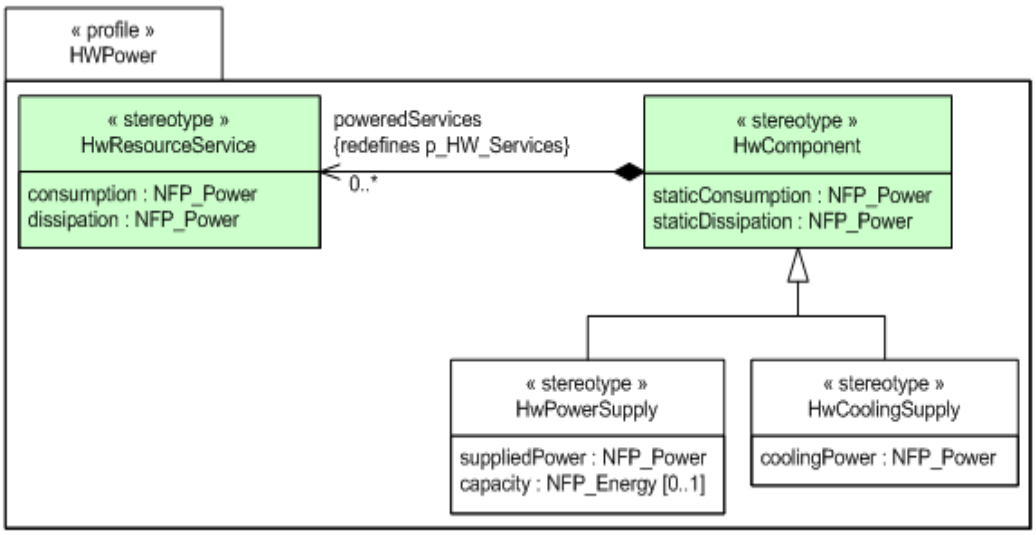

Figure 15 HwPower profile details

4.2 Hardware Modeling in a nutshell

Figure 16 shows the organization of hardware modeling concepts in EAST-ADL. These concepts allow the hardware to be captured in sufficient detail to allow preliminary allocation decisions.

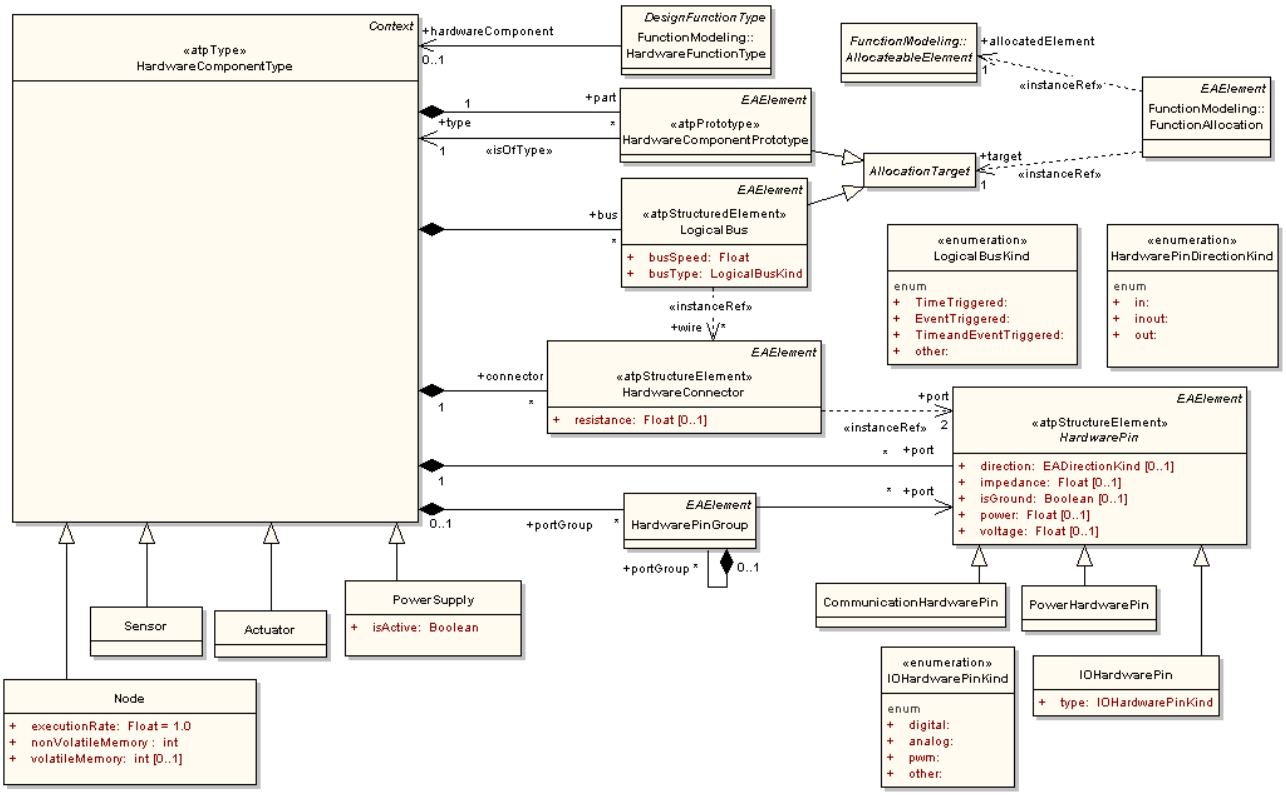

Figure 16 Hardware Modelling

4.3 HRM and Hardware modeling comparison

Common objectives have led to the definition of hardware modeling constructs in MARTE and EAST-ADL. However, MARTE comes with a lower level of detail and presents a different organization of concepts. Concepts are in fact separated in the logical and physical view, while the two views are not separated in EAST-ADL. For instance, the logical bus and the hardware connector are put together in the same EAST-ADL Hardware Modelling package, while representing a logical and physical element, respectively. In MARTE, communication media represents a logical bus in the logical view; while a connector connecting HwComponents represent a physical connector in the physical view. A different example is represented by the HardwareComponentType that in EAST-ADL can be viewed as a logical and physical component at the same time. In MARTE HwComponent is only used for physical representations and can be used to capture the physical aspect of the hardware component. Note that HwComponent is further specialized in MARTE to represent HwPowerSupply, as in EAST-ADL the HardwareComponentType is specialized to PowerSupply.

A MARTE concept for mapping HardwareComponentType in its logical flavor could be HwResource that is in fact further specialized to represent computing, communication and device resources as in EAST-ADL HardwareComponentType is further specialized to represent Node, Sensor and Actuators.

Even if similarities can be found between the above mentioned concepts, it is very difficult to obtain a perfect mapping between EAST-ADL concepts and MARTE HRM ones. The two packages follow very different designs resulting in not only different (but maybe similar) concepts but a very different characterization of these concepts in terms of properties and their relations. Just to make an example, let us take the HwResource stereotype from MARTE. As already said it seems similar to the logical aspect of HardwareComponentType. Nonetheless, let us have a look at the HardwareResource properties in terms of generalization, associations and attributes:

HwResource (from HwLogical)

Generalizations • MARTE::GRM::Resource

Associations

- ownedHW: HwResource[0..*] Specifies the owned sub-HwResources. Subsets Resource.ownedElement.
- p_HW_Services: HwResourceService[0..*] Specifies the provided services. Subsets Resource.pServices.
- r_HW_Services: HwResourceService[0..*] Specifies the required services.
- endPoints: HwEndPoint[0..*] Specifies the connection points of the HwReource. Subsets ownedHW

Attributes

- description: NFP_String Specifies a textual description of the HwResource.
- frequency: NFP_Frequency[0..1] Specifies the clock frequency of the HwResource

Now let us focus on the ‘frequency’ attribute. This attribute will be inherited in MARTE by all the stereotypes specializing HwResource, i.e. computation, communication and devices, including sensors and actuators. Note that in EAST-ADL, even if this ‘frequency’ is conceptually equivalent to ‘executionRate’ of Node, no executionRate is present in Sensors and Actuators, which are a little more abstract than the MARTE counterparts of sensors and actuators (both specializations of HwI/O in HwDevice).

4.4 Hardware Modeling specialization (profile) of plain UML

As already pointed out, to obtain EAST-ADL elements as specialization of MARTE concepts, those MARTE concepts must be enough general to be specialized without violating the EAST-ADL specification, e.g. by introducing an attribute not present in the EAST-ADL specification.

In general, HRM stereotypes result 'too' specialized to be used as generalization of the Hardware Modelling constructs of EAST-ADL. To address this problem we need then to resort to plain UML while inspecting the MARTE Generic Resource Modelling (GRM) package to see if some concept can be used for specialization. GRM in fact provides very abstract resources only characterized by the service they provide, disregarding if this service will be implemented in software or hardware. Figure 17 shows an example of platform architecture at GRM level, where vey high-level resources are represented. At a first look the GRM stereotypes ComputingResource, CommunicationMedia and DeviceResource seem better suited to be further specialized in the EAST-ADL constructs Node (from ComputingResource), LogicalBus (from Communication Media), Sensor and Actuator (from DeviceResource). However, GRM profile details (see Figure 18) show that all these MARTE stereotypes inherits from <<Resource>> that has three attributes (result, isProtected, isActive) not present in EAST-ADL Node, Sensor, Actuator and LogicalBus.

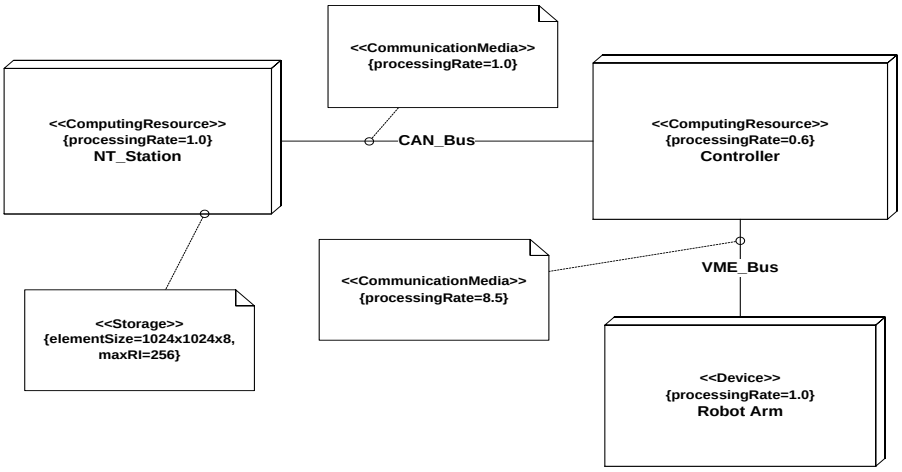

Figure 17 example of usage of GRM

The logical consequence of this fact is that for the specialization we need to resort to plain UML only (as it can be observed the resource stereotype inherits directly from UML), as shown in the next table.

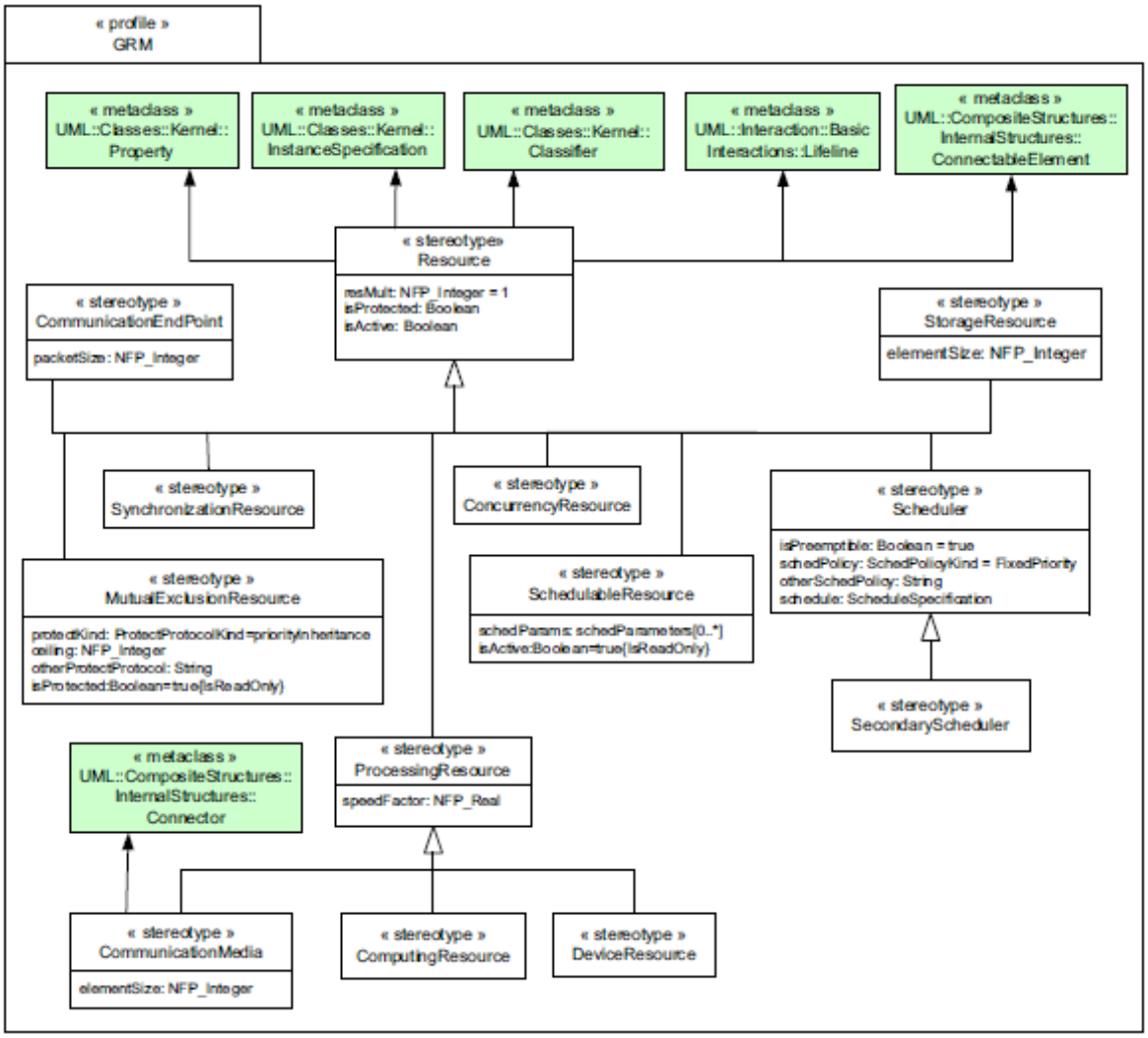

Figure 18 GRM profile details

EAST-ADL concept	Description	UML concept	MARTE stereotype
HardwareComponentType	It is the equivalent of a FunctionType for the Hardware level. It can be decomposed using several HardwareComponentPrototype and feature a set of connectors, ports (called pins at hardware level) and buses. Concrete subtypes are Nodes, Sensors, Actuators, PowerSupplies	Class	None. The HardwareComponentType stereotype is introduced by inheriting from Class
HardwareComponentPrototype	It is the equivalent of a FunctionPrototype for the Hardware level. It is typed by a HardwareComponentType.	Property	None: The HArdwareComopnentPrototype is introduced by inheriting from Property
HardwarePin	It is the equivalent of a FunctionPort for the Hardware level. Concrete	Port	None. The HardwarePin stereotype is introduced

	subtypes are IOHardwarepin, CommunicationHardwarepin, PowerHardwarePin. They feature a HardwarePinDirectionKind (in, inout, out) which is the equivalent of the EADirectionKind at functional level		inheriting from Port
CommunicationHardwarePin	The CommunicationHardwarePin represents the hardware connection point of a communication bus.	Port	None: the stereotype CommunicationHardwarePin is introduced inheriting from Port
IOHardwarePin	The IOHardwarePin represents an electrical pin or connection point. It features an IOHardwarePinKind (analog, digital, pwm – pulse width modulated, other)	Port	None: the stereotype IOHardwarePin is introduced inheriting from Port
PowerHardwarePin	A PowerHardwarePin is primarily intended to be a power supply. The direction attribute of the pin defines whether it is providing or consuming energy.	Port	None: the stereotype PowerHardwarePin is introduced inheriting from Port
HardwareConnector	It is the equivalent of a FunctionConnector for the Hardware level.	Connector	None. The stereotype HardwareConnector is introduced inheriting from Connector
Node	Node represents the computer nodes of the embedded electrical/electronic system. Nodes consist of processor(s) and may be connected to sensors, actuators and other ECUs via a BusConnector. Node denotes an electronic control unit that acts as a computing element executing Functions. In case a single CPU-single core ECU is represented, it is sufficient to have a single, non-hierarchical Node. They are characterized by an executionRate as float, which is the ratio compared to nominal execution (i.e. a 25% faster CPU would have an executionRate of 1.25), volatileMemory and nonVolatileMemory size in bytes	Class	None. The stereotype Node is introduced inheriting from Class
Sensor	A concrete HardwareComponentType representing a Sensor	Class	None. The stereotype Sensor is introduced inheriting from Class
Actuator	The Actuator is the element that represents electrical actuators, such as valves, motors, lamps, brake units, etc. Non electrical actuators fall outside the hardware modeling: they are part of the plant model	Class	None. The stereotype Actuator is introduced inheriting from Class
PowerSupply	PowerSupply denotes a power source that may be active (e.g., a battery) or passive (main relay). A boolean isActive indicates whether the source is active or passive.	Class	None. The stereotype PowerSupply is introduced inheriting from Class

LogicalBus	The LogicalBus represents logical communication channels. It serves as an allocation target for connectors, i.e. the data exchanged between functions in the FunctionalDesignArchitecture. It features a busSpeed as a float, which is in bits per second. Used to assess communication delay and schedulability on the bus. Note that scheduling details are not represented in the model. A LogicalBusKind describeds the type of bus scheduling assumed (EventTriggered, TimeTriggered, TimeandEventTriggered, other)	Class	None. The stereotype LogicalBus is introduced inheriting from Class
HardwarePinGroup	Equivalent of PortGroup for the Hardware level	Port and Class	None. The stereotype HardwarePinGroup is introduced inheriting from Class and Port

5 Allocation modeling in EAST-ADL and MARTE

Allocation modeling provides concepts for the allocation of platform-independent elements to platform resources in both languages. MARTE Allocation (Alloc) is the sub-profile dedicated to allocation modeling, while EAST-ADL provide allocation constructs in FunctionModeling and HardwareModeling

5.1 Alloc in a Nutshell

A MARTE allocation is an association between a MARTE application and a MARTE execution platform. Application elements may be any UML element suitable for modeling an application, with structural and behavioral aspects. An execution platform is represented as a set of connected resources, where each resource provides services to support the execution of the application. So resources are basically structural elements, while services are rather behavioral elements. Note that the MARTE allocation does not use the UML notion of Deployment. MARTE specification is indeed close to SysML approach, where allocation though relating a functional to execution platform mapping, allows that the execution platform is still in an abstract form.

The first step is to identify what can be allocated, the logical view (behavior or structure), and what can serve as a target of an allocation, the physical view (a resource or a service). The stereotype Allocated (Figure 19) is used for this matter

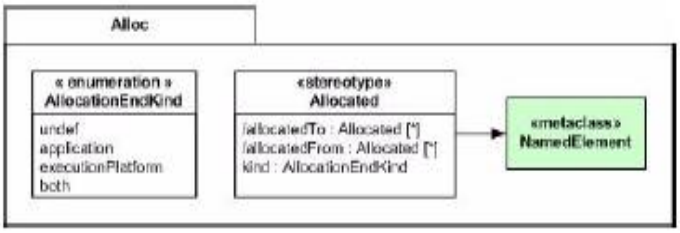

Figure 19 Allocated Stereotype

The second step is to identify what is allocated onto what and what are the reasons for such an allocation and what are the constraints implied by this allocation, hence the definition of the stereotype Allocate (Figure 20). Note that allocation can be specified in different kinds: structural, behavioral, or hybrid. Structural allocation is an association between a group of structural elements and a group of resources. Behavioral allocation is an association between a set of behavioral elements and a service provided by the execution platform. When clear from context, hybrid allocations can also be allowed (for instance when an implicit service is uniquely defined for a resource). At the finer level of detail, behavioral allocation deals with the mapping of UML actions to resources and services. Allocation can also have different nature: it results in both spatial distribution and temporal scheduling. Spatial distribution is the allocation of computations to processing elements, of data to memories, and of data/control dependencies to communication resources. Scheduling is the temporal/behavioral ordering of the activities (computations, data storage movements or communication) allocated to each resource. Scheduling is represented as a relation between the respective time bases of application and platform elements.

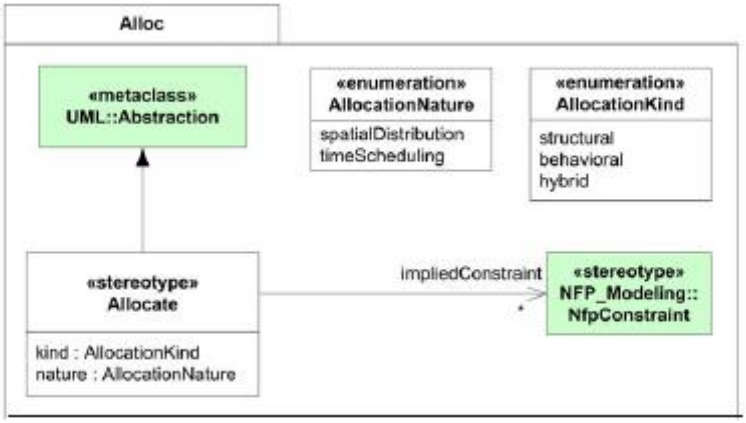

Figure 20 Allocate Stereotype

Alternately an allocation can be specified using the Assign stereotype. The Assign stereotype extends a UML metaclass: Comment with neutral semantics (instead of leveraging the semantics of Abstraction). It defines “from” / “to” attributes to indicate the ends of the assignment. Like an allocation, an assignment can be characterized by its “nature” (spatial or time distribution) and its “kind” (structural, behavioral, or hybrid). The optional body property of the Comment meta-class can be used to provide the justification of the assignment.

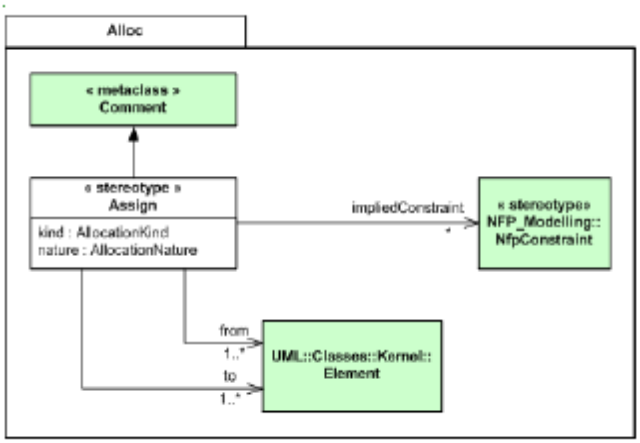

Figure 21 Assign Stereotype

5.2 Allocation constructs in EAST-ADL

In EAST-ADL concepts for modeling allocation are spread over FunctionalModelling and HardwareModelling. More in detail, Allocateable, FunctionalAllocation, Allocation belong to FunctionalModeling, while AllocationTarget belongs to HardwareModelling. FunctionAllocation establishes allocations between elements using an instanceRef mechanism. InstanceRef allows linking two allocatable elements and allocation targets explicitly using a particular *context.* To make an example let us consider the case of two vehicles vehicle1 and vehicle2, both of a common type Vehicle that is made of four wheels: frontleftwheel, frontrightwheel, rearleftwheel,

rearrightwheel. Let us consider now the case that we want to allocate a particular sensing function s1 to the front left wheel of vehicle 1 while another sensing function s2 must be allocated to frontleftwheel of vehicle2. Note that the target in both cases is the frontleftwheel prototype part of the Vehicle type. Thus, to distinguish between the two cases an explicit context must be represented for each case. The context will be of the form {*vehicle1}* for the AllocationTarget of S1 and of the form *{vehicle2}* for the AllocationTarget of s2.

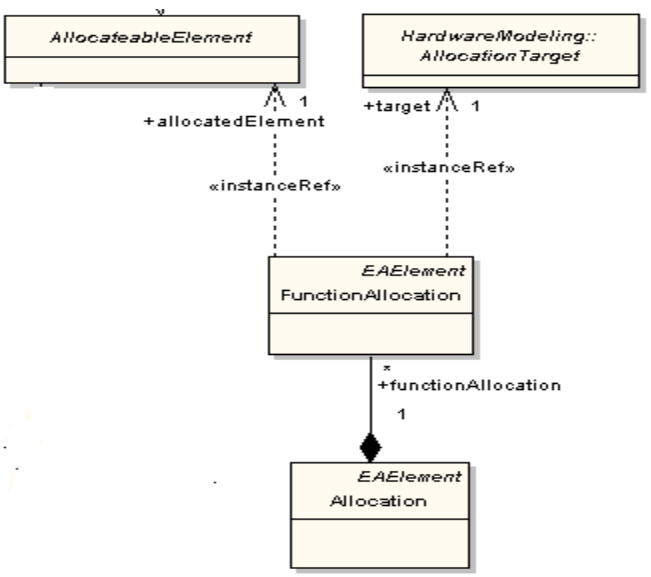

Figure 22 EAST-ADL allocation constructs

5.3 Alloc and EAST-ADL Allocation comparison

While both languages provide constructs for allocation modeling, many differences can be found. First of all in MARTE each element the user wants to allocate must be marked with an <<allocated>> stereotype. Note that every NamedElement can be potentially used in allocations. It is always the user that must specify if the <<allocated>> element is an application or execution platform element (see Figure 19). Note in MARTE allocations are very flexible as they are not restricted to be vertical (from an application to an execution platform element).

In EAST-ADL, allocation can only be established between FunctionalModelling elements and HardwareModelling elements (see Figure 22). As already discussed allocations use in EAST-ADL the instanceRef mechanism, which is not supported by MARTE allocations.

Another point is about the EAST-ADL allocation concept that owns a number of functional allocations. As MARTE <<allocate>> stereotype specializes UML abstraction, and it is not possible for an Abstraction to contain other elements, the two concepts cannot be mapped. Moreover, it is also impossible to map the EAST-ADL functionalAllocation concept to MARTE <<allocate>> since abstractions can only be contained in UML Packages, while functionalAllocation cannot be contained in a package.

To overcome these problems, EAST-ADL functional allocation could be mapped to the <<assign>> stereotype, where EAST-ADL allocation will map to a new stereotype <<allocation>> specializing UML Class. Functional allocation will be owned by the class that contains the UML Comment stereotyped <<FunctionalAlocation>>. Even if functionalAllocation may conceptually

map to the <<assign>> stereotype, the <<assign>> stereotype cannot be used to derive the functionalAllocation stereotype has the ‘kind’ and ‘nature’ attribute will appear.

5.4 EAST-ADL Allocation constructs as specialization (profile) of plain UML

From the considerations made at the end of the previous section, it is again the case in which concepts found in MARTE for allocation are not well-suited to be specialized in EAST-ADL ones.

The following table shows a plain UML specialization for EAST-ADL allocation language constructs.

EAST-ADL concept	Description	UML concept	MARTE stereotype
AllocateableElement	The AllocateableElement abstracts all elements that are allocateable.	NamedElement	None, a new stereotype AllocatableElement is introduced by specializing NamedElement
AllocationTarget	An abstract concept representing the potential target of an allocation.	NamedElement	None, a new stereotype AllocatableElement is introduced by specializing NamedElement
FunctionAllocation	FunctionAllocation represents an allocation constraint binding an AllocateableElement (computation functions or communication connectors) on an AllocationTarget (computation or communication resource). It uses the inst	Comment	None, a new sterotype FunctionAllocation is introduced by specializing Comment

6 Generic Constraints and MARTE NFPs

This section discusses relationships between the Generic Constraints package and MARTE Non-Functional Properties (NFPs).

6.1 NFPs in a nutshell

MARTE modeling of non-functional properties provides detailed non-functional properties descriptions able to represent: system properties, constraints and relationships among them.

Three main stereotypes are defined for this purpose as shown in Figure 23 : Nfp (non-functional property), Nfptype, NfpConstraint, and Unit.

MARTE offers as well a predefined library of Units and NfpTypes as Power, Frequency, DataSize, DataTxRate, Duration, BoundDuration, etc. Figure 24 shows the library. Note that each Nfttype inherits from a NFPCommonType that has the following attributes:

- expr :VSL_Expression. It is a placeholder for mathematical expressions in addition to the actual value.
- source : SourceKind[0..1]. It specifies the origin of the specification (estimated, calculated, required and measured).
- staQ :StatisticalQualifierKind[0.. 1]. It specifies the type of statistical measure of a given property (maximum, minimum, mean, percentile, distribution).
- dir:DirectionKind[0..1]. It defines the type of the quality order relation (increasing or decreasing) in the value domain of nfp type. This allows multiple instances of nfp values to be compared with the relation ‘higher-quality-then’ in order to indentify what values represents the higher quality or importance.

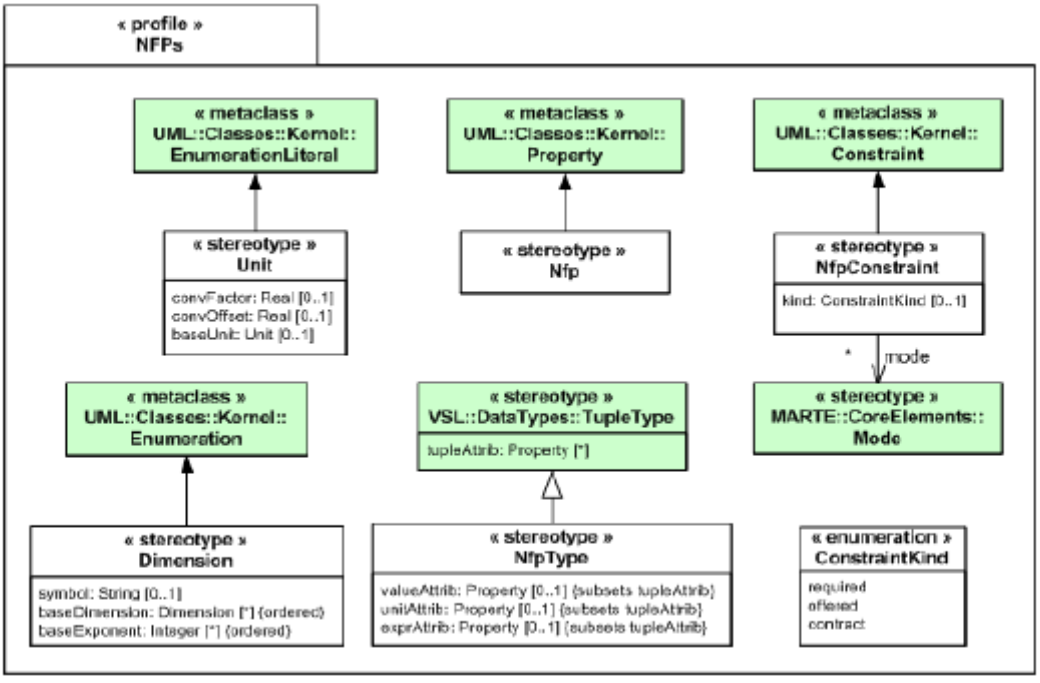

Figure 23 NFP profile details

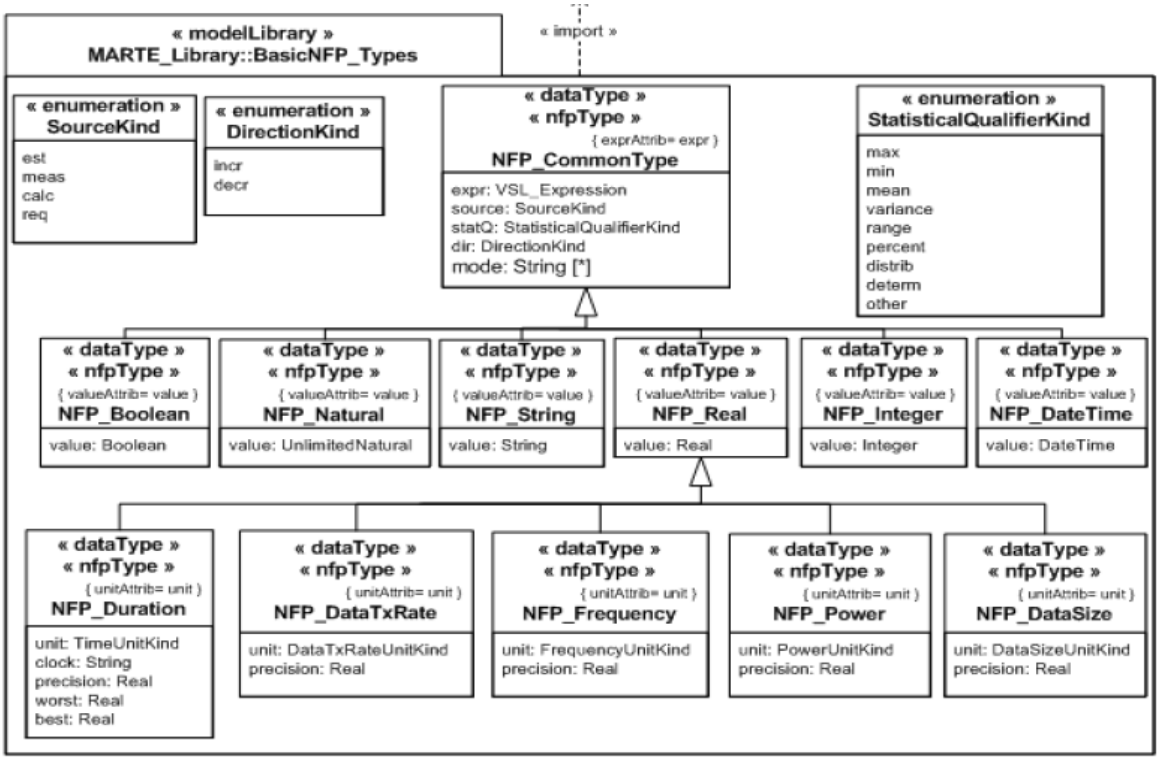

Figure 24 Nfp Type Library

Figure 25 shows and compare the use of nfps against plain UML. More in details the NFPs counterpart of the UML CAN_Bus shows a number of attributes typed by NFPs types (NFP_Real, NFP_DataTxRate, NFP_Duration) instead of UML primitive types (Real, Integer,Real). This allows describing rich expressions for speedFactor, capacity and packetIT as shown in the MARTE can1:CAN_Bus.

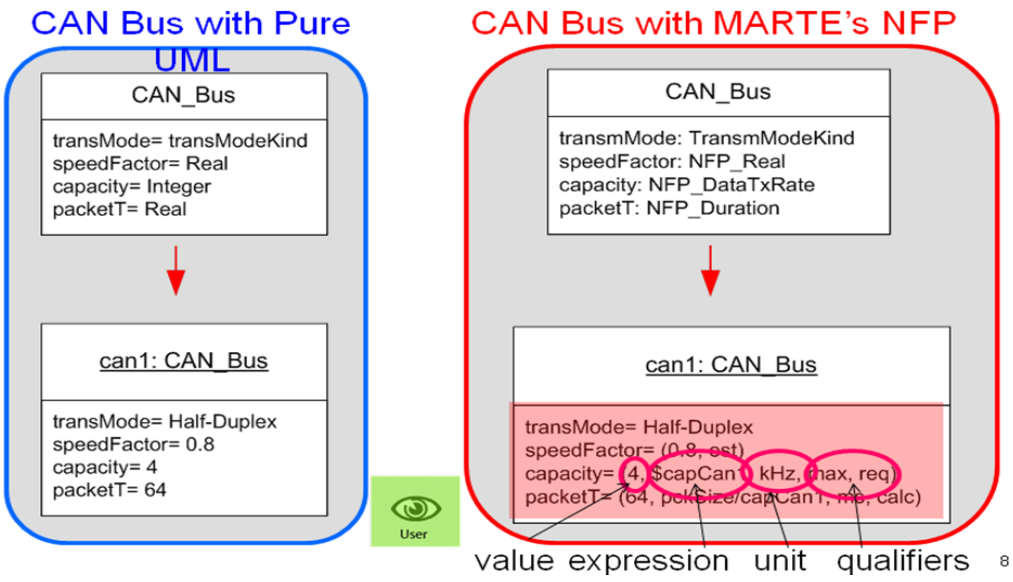

Figure 25 NFPs usage example

6.2 Generic Constraints in a nutshell

Generic constraint denotes a property, a requirement or a validation result. It is a requirement if it refines a Requirement, it is a validation result if it realizes a VVActualOutcome (see Section 7). Figure 26 shows the GenericConstraint package. Note that some kind of constraints are pre-defined (GenericConstraintKind enumeration) and that a value of the generic constraint is always of type String.

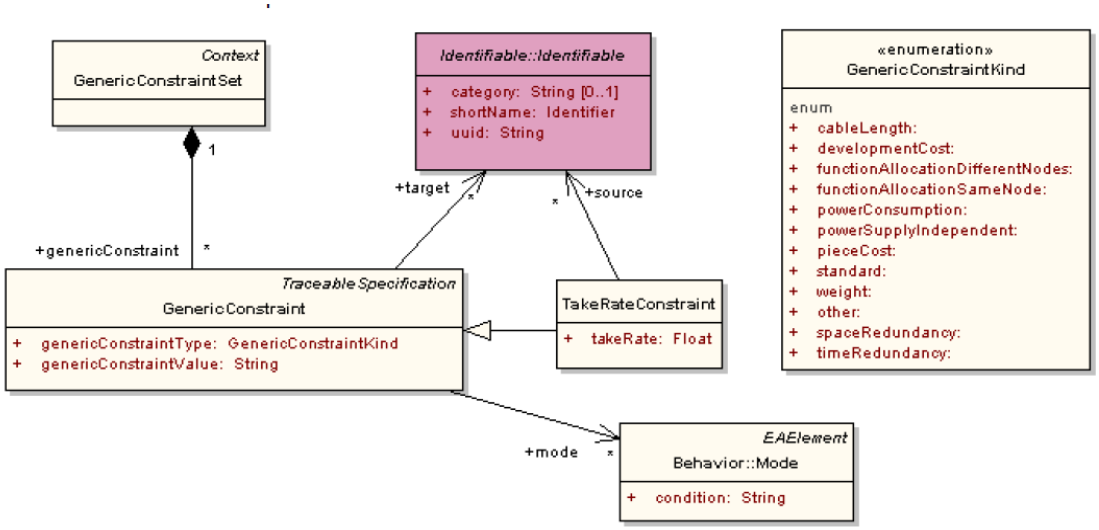

Figure 26 Generic Constraint package

6.3 NFPs and Generic Constraints comparison

The way in which the two languages support the description of non-functional properties is very different. In EAST-ADL non-functional properties are expressed through constraints. The package GenericConstraint offers a way of defines user-constraints, but a set of pre-defined constraints there exists not included in the Generic Constraint package. For instance timing constraints, included in the Timing package, denote a set of non-functional properties capturing timing aspects as execution time constraints or delays constraints. Note that these constraints represented measured/computed properties when attached to a VVactualOutcome. Besides constraints some non-functional properties as power, bus speed, execution rate, can be found as attributes of type Float belonging to hardware modeling elements.

With respect to MARTE, EAST-ADL presents a non-homogenous way of modeling NFPs, as some non-functional properties with Float values are already present in the characterization of hardware elements, some constraints are present in the Timing package, while the general mechanism to enrich elements with non-functional properties offers only the possibility of attaching non-functional properties with string values. Note that in MARTE the NFPs sub-profile offers a powerful and extendible framework to express complex NFP types, which enjoy a set of useful qualifiers and VSL expressions.

6.4 Generic Constraints as specialization (profile) of plain UML

As pointed out in the previous section, the Generic Constraint package shares with NFPs sub-profile of MARTE the objective of enriching model element with annotations for non-functional properties, but besides that, NFPs offers a richer framework than Generic Constraint package does. For this reason, Generic Constraint package elements can be obtained only through specialization of plain UML as the following table shows.

EAST-ADL concept	Description	UML concept	MARTE stereotype
GenericConstraint	Generic constraint denotes a property, a requirement or a validation result	Class	None, a new stereotype GenericConstraint is introduced by extending Class
GenericConstraintSet	Collection of generic constraints	Package	None, a new stereotype GenericConstraintSet is introduced by extending Package
TakeRateConstraint	Defines the ratio between the number of configurations that includes the target elements and the number of configurations that include the source		None, a new stereotype TakeRateConstraint is introduced by specializing the stereotype GenericConstraint

7 Verification and Validation and MARTE GQAM

As already pointed out, the verification and validation (v/v) modeling constructs are organized in EAST-ADL in a single package. In order to find good candidates in MARTE for the mapping of v/v modeling constructs, we should first inspect those sub-profiles whose modeling purpose is the verification and validation of designs. Table 3 shows that three sub-profiles pursue v/v purposes, i.e. GQAM, SAM and PAM. SAM and PAM, however, are not relevant in the context of EAST-ADL as they are used to carry out schedulability analysis and performance analysis of logical designs mapped into real-time operating systems, which is out of the EAST-ADL scope. GQAM, however, is more general than SAM and PAM, as it provides a genral framework for system analysis, without targeting a particular analysis yet. For this reason, we will select GQAM for the mapping of v/v constructs.

7.1 GQAM in a nutshell

The Generic Quantitative Analysis Modelling (GQAM) sub-profile supports predictive or model-based quantitative analysis to detect potentially unfeasible real-time architectures and/or implementations before the realization phase and to validate non functional requirements on the final system. GQAM supports as well architectural exploration and sensitivity analysis, to explore different architecture alternatives. At the heart of GQAM resides the 'AnalysisContext' concept (Figure 27), which defines the context for the analysis: the logical application, workload for the analysis, the target platform and the parameters for the analysis, e.g. optimization criteria, constraints.

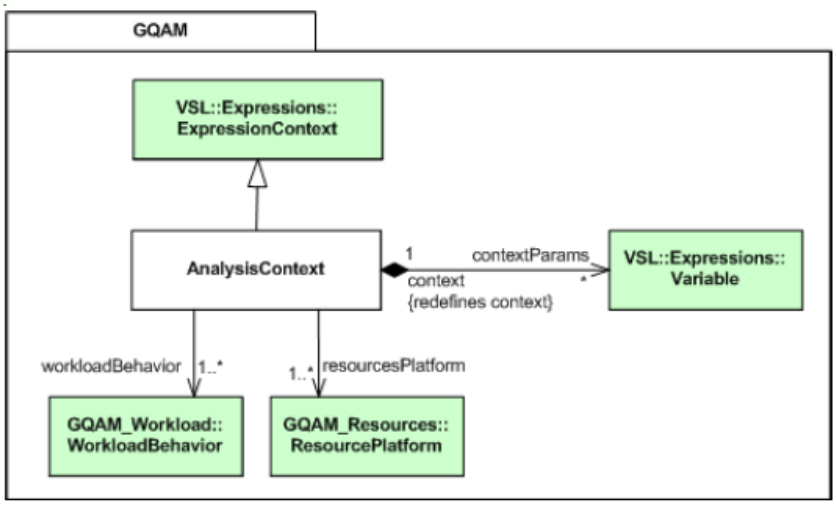

Figure 27 AnalysisContext details

More in detail, a workload is composed by behavior scenarios: selected runs for the applications (particular chains of function activations), stressed by a workload event (a set of stimuli). The behavior scenarios selected for the analysis are modeled as a sequence of atomic steps, with a start and a finish time. These steps are characterized by a set of non-functional

properties as host demand, throughput, response time, utilization, etc. Note that these properties can be input for the analysis or outputs from the analysis (Figure 28).

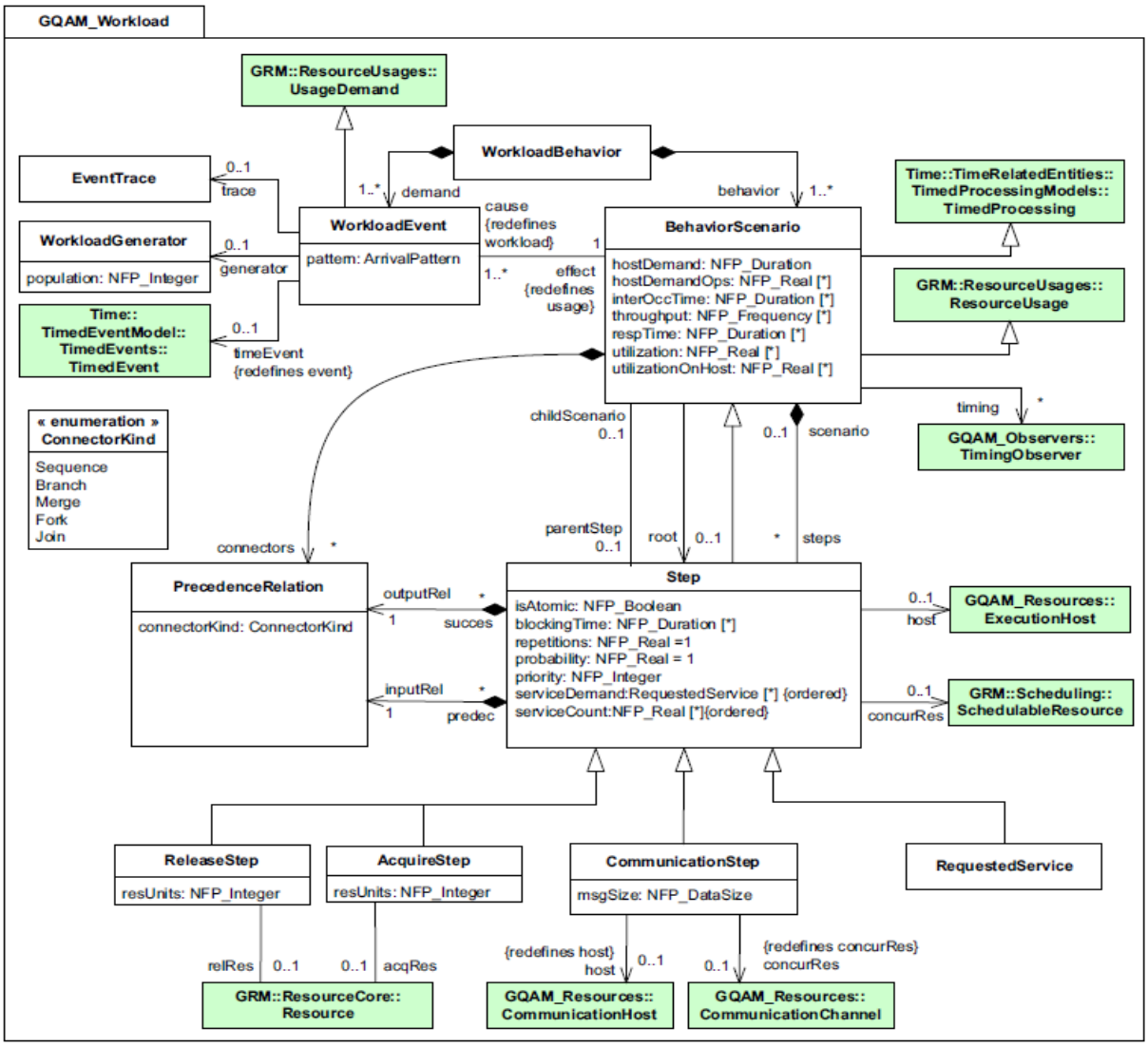

Figure 28 Workload details

7.2 Verification and Validation in a nutshell

The Verification and Validation package of EAST-ADL does not target a particular analysis, method or verification and validation activity, because at EAST-ADL level many different verification and validation (v/v) techniques, methods and tools can be applied. The goal is to provide instead means for planning, organizing and describing v/v activities and to define the links between v/v activities, the satisfied and verified requirements and the objects modeling the system. Information that is specific to an individual technique is not described in EAST-ADL but a place for storing this information is provided. The most important language constructs (shown in Figure 29) are:

- VVCase. It represents a v/v effort
- VVProcedure. It represents a task in a v/v effort

- VVTarget. It represents a testing environment in which a v/v effort is performed
- VVLog. It represents the execution of a v/v effort
- VVActualOutcome. It represents the actual outcome of a performed v/v effort
- VVIntendedOutcome. It represents the expected outcome of a v/v effort
- VVStimuli. It represents input values for a VVProcedure

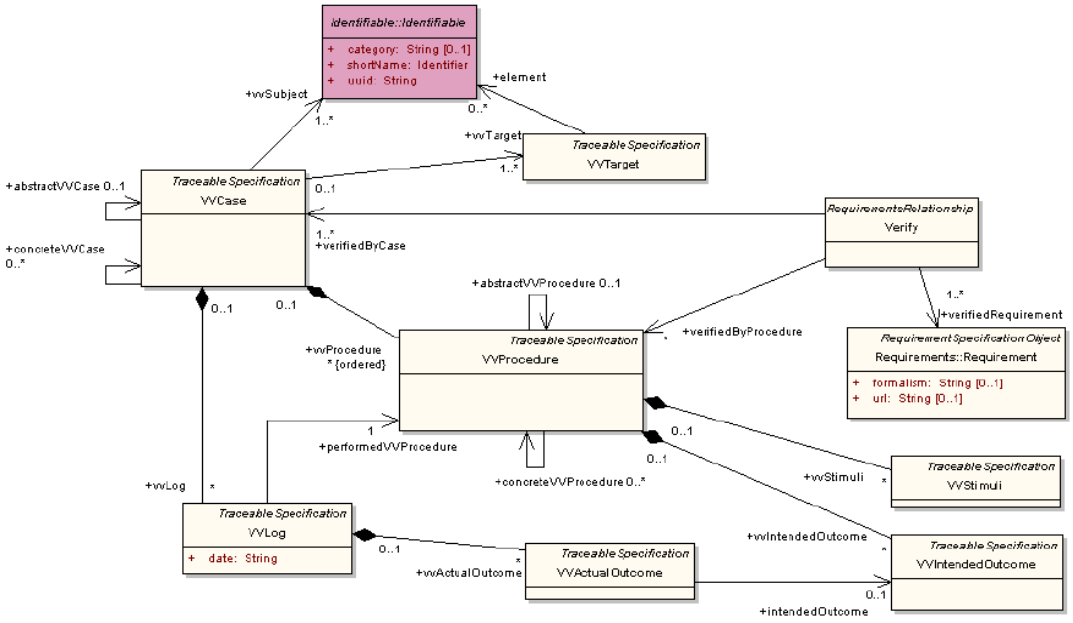

Figure 29 Verification and Validation package details

7.3 GQAM and Verification and Validation comparison

The EAST-ADL Verification and Validation package is at higher-level of abstraction than the GQAM level. Nicely, GQAM concepts can be viewed as refinements of EAST-ADL ones. For instance, let us take the case of the VVStimuli in EAST-ADL. In this case the GQAM WorkloadEvent stereotype can be viewed as a refinement of VVStimuli in which the stimuli is a stream of triggering occurrences. The stream may be generated by a Timed Event, have a stated arrival pattern, may have a generator (e.g. state machine) or generated by a trace stored in a file. In the same line, VVTarget could be refined by the WorkloadBehavior stereotype. In this case the VVTarget is a set of selected critical activation paths subjected to the stimuli.

7.4 Verification and Validation as specialization (profile) of plain UML

Since the Verification and Validation package is at higher abstraction level than the GQAM package, once again we need to resort to plain UML to define EAST-ADL verification/validation constructs.

EAST-ADL concept	UML concept	MARTE stereotype
VVCase	Class	None, a new stereotype VVCase is introduced by extending Class
VVTarget	Class	None, a new stereotype VVTarget is introduced by extending Class
VVIntendedOutcome	Class	None, a new stereotype VVIntendedOutcome is introduced by extending Class
VVStimuli	Class	None, a new stereotype VVStimuli is introduced by extending Class
VVProcedure	Class	None, a new stereotype VVProcedure is introduced by extending Class
VVActualOutcome	Class	None, a new stereotype VVActualOutcome is introduced by extending Class
VVLog	Class	None, a new stereotype VVLog is introduced by extending Class

8 Timing Modeling and MARTE Time

As already pointed out, the timing modeling constructs are organized in EAST-ADL in a single package, called Timing, which includes Timing, Timing Constraints and Events. Good candidates in MARTE for the mapping of timing modeling constructs can be found in the Time (sub-) profile. In the following main concepts from MARTE Time and the MARTE profile for EAST-ADL Timing are presented.

8.1 MARTE Time in a nutshell

MARTE Time describes a general framework for representing time and time-related concepts and mechanisms that are appropriate for modeling real-time and embedded systems.

At the heart of the time modeling resides the concept of time structure. A time structure is defined by a time base, in basic timing models, and by time structure relations in multiple timing models. A timing base can be discrete or dense and it is a container of instants.

The access to time is via clocks. A clock is related to a time base. Clocks can be logical or physical. Any clock can be associated to model elements, making the model element a timed element as shown in Figure 30.

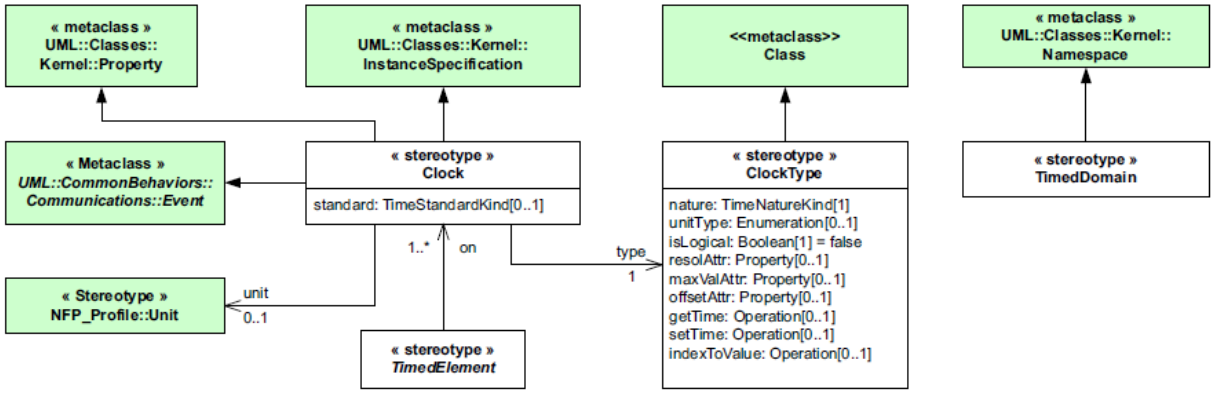

Figure 30 Clocks and Timed Elements

More in details by associating a clock to behavioral elements, we obtain TimedEvent(s), and TimedProcessing (s). The association of a clock to a constraint, results in a TimedConstraint, while the association of a clock to a Data Type (or Value) results in a TimedValue.

A timed event (shown in Figure 31) specifies event whose occurrences are bound to a single clock

A timed processing (shown in Figure 32) represents executions that have known start and finish times OR a known duration.

A TimedConstraint (Figure 33) imposes constraints on either instant value or duration value associated with model elements bound to clocks.

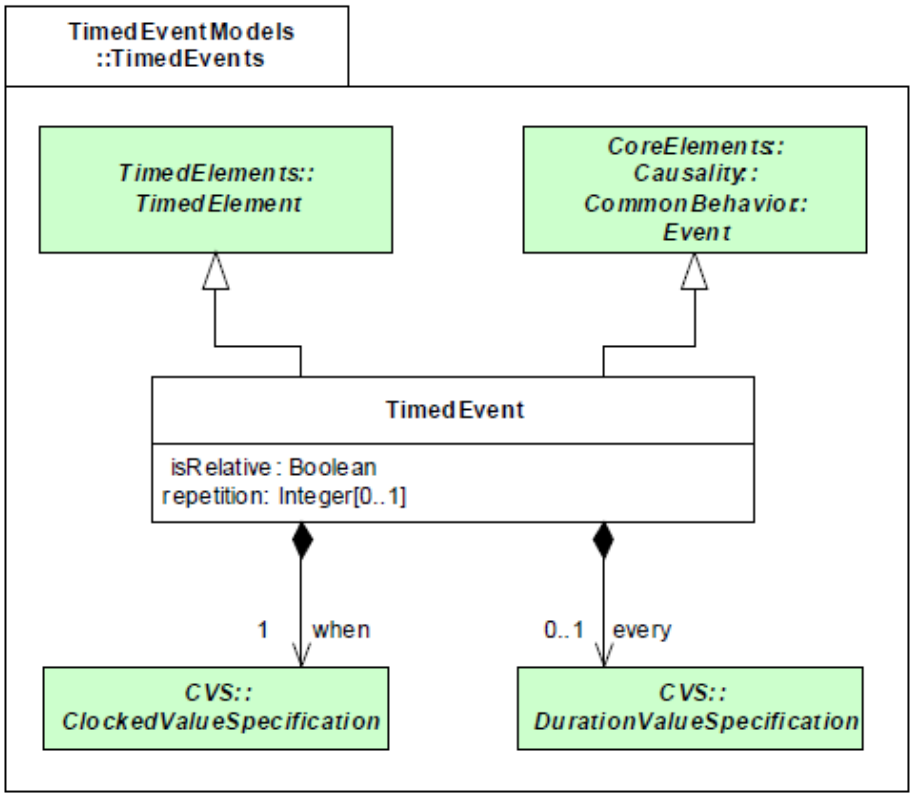

Figure 31 TimedEvent

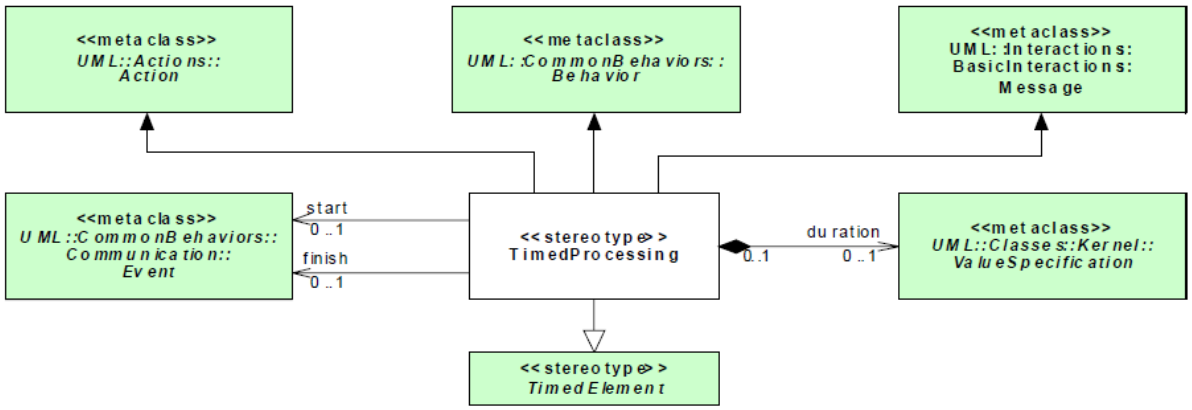

Figure 32 UML profile for TimedProcessing

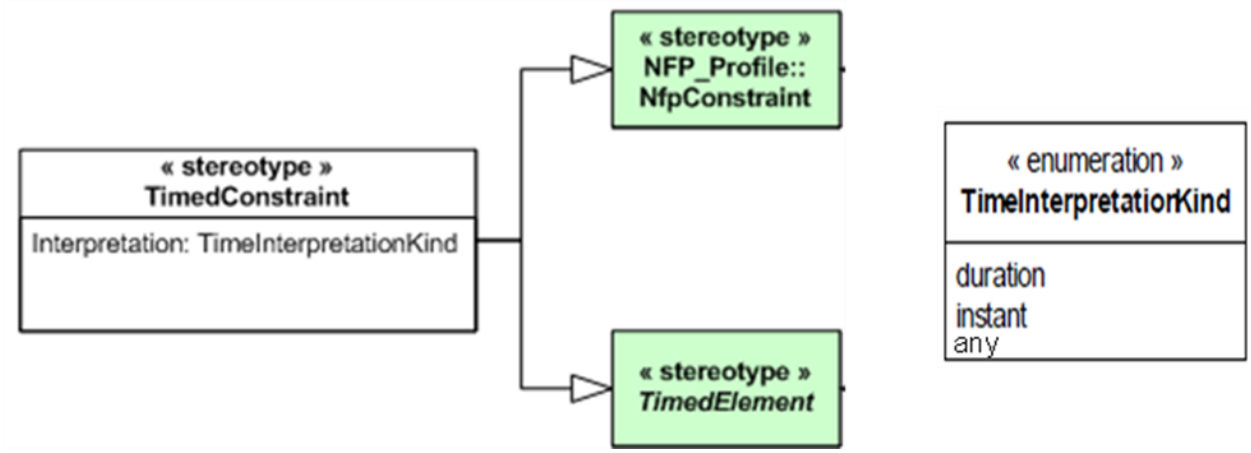

Figure 33 Timed constraint

8.2 The MARTE (Time) Profile for EAST-ADL Timing Modeling

In this section we review the EAST-ADL constructs used to model timing aspects and how these concepts are specialized from MARTE stereotypes.

EAST-ADL concept	Description	UML concept	MARTE stereotype
Event	An Event represents a distinct form of state change in a running system, occurring at different time instants – in this case the Event.isStateChanged is set to true. Or it is a periodical report of the current state of the system (the same Boolean property is false). It is assumed one can observe such events and tell the time instants at which they occur. An Event can either be a stimulus, which causes another Event or a response to another Event. These roles are assigned in EventChains.	Event	The UML Event metaclass is extended and the EAST-ADL property isStateChange shall be added to this extension.
EventFunction	An event of a Function refers to the triggering of the Function, i.e., when the input data is consumed, data transformation is performed on that input data by the function, and output data is produced.	Class	Specializes by inheritance from TimedElement with additional EAST-ADL properties: functionType and functionPrototype Note that the most similar concept in MARTE is TimedEvent, but TimedEvent enjoys the 'repetition' and the 'isRelative' which are absent in EventFunction.
EventFunctionFlowPort	Event that refers to the triggering of the Function at a flow port, i.e., when data is sent or received.	Class	Specializes by inheritance from TimedElement with additional EAST-ADL property: port

EventFunctionClientServerPort	Event that refers to the triggering of the Function at a client/server port, i.e., when the input data is sent / received, or when the output data is produced / received.	Class	Specializes by inheritance from TimedElement with additional EAST-ADL properties: port and eventKind
EventChain	EventChains depict temporal sequences of Events occurring in response or causing other Events. Constraints may be attached to such chains. EventChains can refer to other EventChains: the referred chains refine the top chain, either as an ordered sequence (they are referred as segments) or parallel chains (they are referred as strands).	Class	Specializes by inheritance TimedElement stereotype and adds EAST-ADL properties: stimulus and response Note that the most similar concept is TimedProcessing stereotype which includes start and finish that could correspond to stimulus and response of EventChain
TimingConstraint	TimingConstraint regroups a lower and upper TimeDuration, which serve as bounds to a certain Event or EvenChain. The link to Events or EventChains is managed by a Timing construct (see this). The bounds can be either requirements, or a validation result or an intended validation result, depending on what the TimingConstraint refines, resp. a Requirement, a VVActualOutcome or a VVIntendedOutcom (through a Refine relationship). A mode property specify the modes in which the constraint is valid. Concrete subconstructs are ExecutionTimeConstraint, PrecedenceConstraint or various subtypes, which define specific time responses (DelayConstraints) or event models (EventConstraints).	Class, Constraint	Specializes by inheritance TimedConstraint stereotype and adds EAST-ADL properties: upper and lower
Timing	Regroups and links TimingConstraints to either Events or EventChains (both are TimingDescriptions).	Class, Package	None
TimeDuration	Defines a duration value as a Float, a code (cseCode) provides an integer value which defines either the time unit (ms, etc.) or angular or combustion step. See specification for a detailed explanation.	DataType	Specializes by inheritance TimedValueType stereotype
ExecutionTimeConstraint	ExecutionTimeConstraint expresses the execution time of a function under the assumption of a nominal CPU that executes 1 "function second" per second. Function allocation will decide the actual execution time by multiplication with the relative speed of the host CPU. The function is activated by a time trigger or a port trigger. The function starts execution some time after activation, depending on e.g. interference and blocking from other functions on the same	Class, Constraint	Specializes by inheritance EAST-ADL TimingConstraint stereotype. Added EAST-ADL properties are: designFunctionType, designPrototype, variation

	resource. Immediately on start, the function reads input data on all ports. Functions write data at the latest when the execution time has elapsed (which is after the execution time plus any blocking and interference time). A variation property (TimeDuration) defines the allowed variation between worst and best execution time. The target of this constraint is either a DesignFunctionType or a DesignFucntionPrototype		
PrecedenceConstraint	The PrecedenceConstraint represents a particular constraint applied on the execution sequence of functions, such that all predecessors have completed before the successors are started. FunctionPrototypes are referred to, paths enable to reference particular function prototypes in the context of a composite. Note: without a precedence relation, Functions are executed according to their data dependencies, if these are uni-directional. For bi-directional data dependencies, execution order is not defined unless the PrecedenceDependency relationship is used.	Class, Constraint, Dependency	Specializes by inheritance EAST-ADL TimingConstraint stereotype. Added EAST-ADL properties are: successive and preceeding FunctionPrototypes.
DelayConstraint	The DelayConstraint provides additional parameters to define a bound, aside from the upper and lower values inherited from TimingConstraints. The additional properties are jitter and nominal. Variation around the nominal value can be expressed by means of an upper and lower bound, or by means of a jitter value. For example, [lower=10, upper=20, nominal=15] is equal to [nominal=15, jitter=10]. A scope property refers to the EventChain on which the constraint is applied.	Class, Constraint	Specializes by inheritance EAST-ADL TimingConstraint stereotype. Added EAST-ADL properties are: scope event chain, nominal time duration and jitter.
ReactionConstraint	ReactionConstraint is used to impose a timing constraint on an event chain in order to specify bounds for reacting on the occurrence of a stimulus or stimuli. The intention of this constraint is to look forward in time.	Class, Constraint	Specializes by inheritance EAST-ADL DelayConstraint stereotype.
AgeConstraint	In case of over- or undersampling, a one-to-one relation is not possible between the occurrences of stimuli and responses of the associated event chain. Thus, the age constraint defines the semantic of which delay must be constrained.	Class, Constraint	Specializes by inheritance EAST-ADL DelayConstraint stereotype.
OutputSynchronizationConstraint	OutputSynchronizationConstraint expresses a timing constraint on the output synchronization among the set of response events.	Class, Constraint	Specializes by inheritance EAST-ADL DelayConstraint stereotype. Added EAST-

			ADL property is: width (time duration).
InputSynchronizationConstraint	InputSynchronizationConstraint expresses a timing constraint on the input synchronization among the set of stimulus events.	Class, Constraint	Specializes by inheritance EAST-ADL DelayConstraint stereotype. Added EAST-ADL property is: width (time duration)
EventConstraint	The EventConstraint describes the basic characteristics of the way an event occurs over time. In addition an event model may specify an offset, which delays the start of the first period - the occurrence of the very first event - by the given amount of time.	Class, Constraint	Specializes by inheritance EAST-ADL TimingConstraint stereotype. Added EAST-ADL properties are: event and offset.
ArbitraryEventConstraint	The ArbitraryEventConstraint describes whether an event occurs occasionally, singly, irregularly or randomly.	Class, Constraint	Specializes by inheritance EAST-ADL EventConstraint stereotype. Added EAST-ADL properties are: minArrivalTime and maxArrivalTime.
PatternEventConstraint	PatternEventConstraint describes that an event occurs following a known pattern.	Class, Constraint	Specializes by inheritance EAST-ADL EventConstraint stereotype. Added EAST-ADL properties are: minimumInterraArrivalTime and maximumInterraArrivalTime
PeriodicEventConstraint	The PeriodicEventConstraint describes that an event occurs periodically.	Class, Constraint	Specializes by inheritance EAST-ADL EventConstraint stereotype. Added EAST-ADL properties are: period, minimumInterrArrivalTime and jitter
SporadicEventConstraint	The SporadicEventConstraint describes that an event occurs occasionally.	Class, Constraint	Specializes by inheritance EAST-ADL EventConstraint stereotype. Added EAST-ADL properties are: period, minimumInterrArrivalTime, maximumInterrArrivalTime and jitter

8.3 Summary

The review of core concepts has focused on functional elements, hardware elements, and the way to allocate one on the other, verification and validation, non-functional properties modeling and timing. A description of a MARTE profile for EAST-ADL covering the timing aspect has been presented.

9 References

[1] ATESST2 Deliverable D4.1.1 EAST-ADL Profile Specification, June 2010.

[2] MAENAD Deliverable D5.2.1 MAENAD Analysis workbench, June 2011

[3] OMG: UML Profile for MARTE, Version 1.0, June, 2009.

[4] EAST-ADL profile specification M2.1.10, March 2012

[5] EAST-ADL specification M2.1.11, June 2013

[6] UML Profile Specification for M2.1.12 http://maenad.eu/publications.html