MAENAD

Grant Agreement 224442

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

Report type Deliverable D4.2.1

Report name EAST-ADL Profile for MARTE

Dissemination level	**PU**
Status	**Intermediate**
Version number	**1.2**
Date of preparation	**2012-07-27**

Authors

Editor	E-mail
David Servat	David.Servat@cea.fr
Chokri Mraidha	Chokri.Mraidha@cea.fr
Sara Tucci	Sara.Tucci@cea.fr

Authors	E-mail

The Consortium

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

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

MetaCase (Fi) Pulse-AR (Fr) 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

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 integration of 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 will explicitly connect the stereotypes of the EAST-ADL profile to MARTE, making EAST-ADL profile de facto a sub-profile of MARTE.

The foreseen advantages are clarity, and further enhancement of the standardization effort for the EAST-ADL language.

The present deliverable will first explain the overall strategy adopted for conducting this re-engineering of the existing profile, then will consist in a specification of the new profile based on MARTE.

The deliverable reviews a mapping of the core elements of the language. The engineering of the profile itself is under development and an experimental implementation of the EAST-ADL timing extension will be achieved by M24. This experimental implementation will be placed on branch separated from the current EAST-ADL profile to allow users made some experiments without breaking their existing models. Later, the profile will have to take into account the potential extensions proposed by other work tasks in the MAENAD project. New issues might appear (e.g. behavior, optimization, etc.) whose mapping to MARTE will have to be studied. It will be summarized either in intermediate releases or in the final version, planned for M36.

2 Design strategy

In this chapter the overall strategy for conducting the design of an EAST-ADL profile for MARTE will be explained. The starting point 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 subsequent sections will provide with an update of this mapping study.

2.1 Functional modeling concepts

In this section we present an updated mapping table for the main elements of the EAST-ADL language which concern functional modeling. These are the definition of function types and prototypes, connectors and ports in various flavors. It can be noted that the overall structure of this part is shared in several modeling areas of the EAST-ADL language. For instance, one finds the same composite structuring and usage of ports and connectors, for hardware architecture description, error modeling, etc. As a result the same kind of mapping can be adapted to deal with these specific areas.

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 acts as a part.	Part	None: uses the plain UML2 part concept. A FunctionPrototype will be represented as a property typed by a FunctionType.
FunctionPort	The FunctionPort is an abstract port for data-flow or client-server interaction, which has several concrete subtypes	Port	See concrete mappings for FunctionFlowPort, FunctionClientServerport, FunctionPowerPort
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	FlowPort with direction in,inout,out. FlowPort defines also an RtFeature refering to an RtSpecification to denote the possible inter arrival time between two occurrences of the data conveyed via the port.
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	Port	Either a FlowPort with dedicated FlowSpecification, or specific stereotype here

	exchange		
FunctionClientServerInterface	The FunctionClientServerInterface is used to specify the operations in FunctionClientServerPorts.	Interface	ClientServerSpecification
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	ClientServerPort, with kind set to provided or required (w.r.t to server/client type)
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 DesignFunctionType.	Part	None: uses the plain UML2 part concept. A DesignFunctionPrototype will be represented as a property typed by an 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 BasicSoftwareFunctionTyp e 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

2.2 Hardware modeling concepts

In this section we review the EAST-ADL constructs for hardware modeling and depict potential mappings with MARTE concepts.

MARTE introduces a set of constructs to depict resources, be they computing, device, or communication resources at a generic level with the GenericResourceModeling package (GRM) or in a much more detailed fashion in the HardwareResrouceModeling package (HRM).

The level of description in terms of parameter defined by EAST-ADL seems to better match the generic description (see the example of a Node and its potential counterparts ComputingResource and HW_Processor). On the other hand sometimes using such generic concepts does not allow a clear distinction between elements: for instance MARTE distinguishes between two DeviceResources, HW_Sensor and HW_Actuator, but the latter are from the HRM package.

Thus there are two alternative ways to map concepts here, see table below.

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	HW_Component
HardwareComponentPrototype	It is the equivalent of a FunctionPrototype for the Hardware level. It is typed by a HardwareComponentType.	Class	None: in fact HW_Component can own subcomponents which are then typed by a HW_Component
HardwarePin	It is the equivalent of a FunctionPort for the Hardware level. Concrete subtypes are IOHardwarepin, CommunicationHardwarepin, PowerHardwarePin. They feature a HardwarePinDirectionKind (in, inout, out) which is the equivalent of the EADirectionKind at functional level	Port	None
CommunicationHardwarePin	The CommunicationHardwarePin represents the hardware connection point of a communication bus.	Port	None: the stereotype CommunicationHardwarePin is introduced
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
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
HardwareConnector	It is the equivalent of a FunctionConnector for the Hardware level.	Connector	CommunicationMedia or HW_Media
Node	Node represents the computer	Class	HWComputingResource or

	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		HW_Processor from the HRM (HardwareResourceModeling, but both features a lot of parameters (such number of cores, etc.), which go beyond the EAST-ADL description. An alternative is to use **ComputingResource** from the GRM (GenericResourceModeling package) which only introduces a speedFactor, equivalent of executionRate
Sensor	A concrete HardwareComponentType representing a Sensor	Class	HW_Sensor or DeviceResource
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	HW_Actuator or DeviceResource
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	HW_PowerSupply or HW_Battery from the HRM (HardwareResourceModeling), providing a suppliedPower: NFP_Power and capacity:NFP_Ernergy Alternatively DeviceResource
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	HW_Bus or CommunicationResource
HardwarePinGroup	Equivalent of PortGroup for the Hardware level	None	None

2.3 Allocation modeling concepts

In this section we review the EAST-ADL constructs used to model allocation of functions to hardware. We briefly present MARTE constructs for this issue. It is worth noting that MARTE notion of allocation differs from the concept of deployment introduced by UML. The idea as stated in the specification is to be 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. This essentially fits with EAST-ADL position on the matter.

EAST-ADL concept	Description	UML concept	MARTE stereotype
AllocateableElement	The AllocateableElement abstracts all elements that are allocateable. There is no way for an allocated element to have access to the other end of an allocation it is taking part in, contrary to MARTE, with its Allocated stereotype.	NamedElement	Yet MARTE defines an additional stereotype: Allocated which is applied to an allocated element, ie. An element part of an Allocate relationship (see below). It features derived property list of the sources and targets allocatedTo, allocatedFrom and an attribute AllocationEndKind (undef, application, executionPlatform, both) to distinguish the role played by the allocated element.
AllocationTarget	An abstract concept representing the potential target of an allocation. Concrete subtypes are LogicalBus and HardwareComponentPrototype, whose type can be Node, Sensor, Actuator, PowerSupply (all being concrete subtypes of HardwareComponentType)	None	In fact no restriction are made any NamedElement can serve either as source or target
FunctionAllocation	FunctionAllocation represents an allocation constraint binding an AllocateableElement (computation functions or communication connectors) on an AllocationTarget (computation or communication resource).	Abstraction	Allocate features an optional set of implied constraints as NfpConstraint, for instance to depict the cost of a particular allocation. Also an AllocationKind (structural, behavioral or hybridand AllocationNature (spatialDistribution to assign computations to execution resources or timeScheduling when describing a temporal/behavioral ordering – the order being that of targets) complement the description.

2.4 Behavior, Timing and Events

In this section we review the EAST-ADL constructs used to model behavior, event occurrence and timing constraints attached to these events.

EAST-ADL concept	Description	UML concept	MARTE stereotype
Behavior	A placeholder to regroup FunctionBehavior, FunctionTrigger, and ModeGroup.	Class, Package	MARTE does not add any specificity to the Class or Package UML concepts.
FunctionBehavior	Represents the behavior of a particular FunctionType. They may refer to execution modes. An enumeration defines the type of representation used to model the behavior, e.g. Simulink, UML, etc.	Behavior	Appropriate UML behavior can be used. No MARTE specificitiy is required for FunctionBehavior modeling.
FunctionTrigger	Defines the activation of a function behavior either in general (when refereeing to a type) or in context (when referring to a prototype). This activation is either event-based (in this case, a set of referred ports must be provided) or time-based (referred ports must be empty). This is defined by the triggerPolicy property. In case it is time-based an extra condition Represents the behavior of a particular FunctionType. They may refer to execution modes.	Class	Specializes MARTE RtFeature stereotype.
Mode, ModeGroup	Defines activation modes for function with conditions. ModeGroup regroup such Modes.	Class	In MARTE Modes are modeled with dedicated state machines made of Mode states and Mode transitions allowing mode changes. Mode Groups are represented with composite mode states. Such a state machine could be attached to a function to define its activation mode.
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	Class	Specializes by inheritance from TimedEvent with additional EAST-ADL properties: functionType

	output data is produced.		and functionPrototype
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 TimedEvent 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 TimedEvent 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 TimedProcessing stereotype and adds EAST-ADL properties: stimulus and response
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	Class, Constraint	Specializes by inheritance EAST-ADL TimingConstraint stereotype. Added EAST-ADL properties are: designFunctionType, designPrototype, variation

	starts execution some time after activation, depending on e.g. interference and blocking from other functions on the same 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

2.5 Summary

The review of core concepts has focused on functional elements, hardware elements, and the way to allocate one on the other. This part covers the needs for the models that are dealt with the Autosar gateway (see D5.2.1 [2]), which takes as input a functional and hardware description of a

system and generates a potential Autosar architecture, where software components and runnables are guessed from the allocation pattern in the EAST-ADL model.

A description of a MARTE profile for EAST-ADL covering the timing aspect has been presented. This profile allows for modeling time using EAST-ADL concepts while being compatible with MARTE way of modeling time. Once EAST-ADL timing extension will be aligned with TADL2 timing description language in the future versions of EAST-ADL, some adjustments will be made on that timing part.

3 EAST-ADL profile for MARTE specification

In this chapter the specification of the new profile will be presented, following the recommendations of the OMG document format once EAST-ADL timing extension will be aligned with TADL2 timing description language.

For the moment, the specification of EAST-ADL profile is found in a separate document – see [4].

4 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 M2.1.9, august 2011