Completing EAST-ADL2 with MARTE for enabling scheduling analysis for automotive applications

Saoussen Anssi, Sara Tucci-Pergiovanni, Chokri Mraidha, Arnaud Albinet, François Terrier, Sébastien Gérard

Completing EAST-ADL2 with MARTE for enabling scheduling analysis for automotive applications

Saoussen Anssi[1], Sara Tucci-Pergiovanni[2], Chokri Mraidha[2], Arnaud Albinet[1], François Terrier[2], Sébastien Gérard[2],
[1]Continental Automotive France SAS, PowerTrain E IPP
1 Avenue Paul Ourliac - BP 83649, 31036 France
{saoussen.ansi, arnaud.albinet}@continental-corporation.com
[2]CEA LIST, Laboratory of model driven engineering for embedded systems,
Point Courrier 94, Gif-sur-Yvette, F-91191 France
{sara.tucci, chokri.mraidha, sebastien.gerard, françois.terrier}@cea.fr

Abstract: Automotive software systems become more and more complex presenting tougher safety requirements and tighter timing constraints. On the other hand, today, timing verification of automotive systems is considered late in the development process, mainly at the implementation stage. Consequently, this leads generally to late detection of design mistakes involving extra costs. In this paper, we propose a model driven approach to perform real-time analysis since the earliest design phases. This approach is based on the combination of two modeling languages for system design (EAST-ADL2 and MARTE) and the integration of an open source toolset for scheduling analysis, MAST.* Benefits of the proposed approach are discussed through a practical automotive case study.

Keywords: Modeling, Scheduling analysis, EAST-ADL2, MARTE, Timing verification.

1. Introduction

Today, highly competitive industries developing real-time systems must face industry requirements both quickly and dependably. “Quickly” refers to the time-to-market issue, where delays in design or implementation incur penalties and hence reduces market profits. “Dependably” refers to the trustworthiness of the services provided by developed systems. One of the key dependability factors in real-time systems is related to system failures. Hence, whenever fault tolerance cannot be guaranteed, fault prevention is the only way to catch possible remaining system failures. Particularly, timeliness, memory constraints and other non-functional constraints belong to the set of properties that must be assured for prevention of real-time system failures.

In modern cars, more and more algorithms are implemented as distributed systems. For example, the development of an ACC-System (*Adaptive Cruise Control*) involves today a minimum of five ECUs (*Electronic Control Units*): Engine ECU, Gearbox ECU, Breaking ECU, the MMI-Interface, and an ECU operating the radar system. Handling the overall timing behavior of such a distributed system is a fundamental challenge during design. For example, on of the key issue of this context is verifying that the so-called *end-to-end timing from* a sensor to an actuator must meet a given deadline. In order to fulfill such requirements, the timing properties on the bus, the ECU timing properties, but also the timing properties of the communication controller have to be taken into account. These timing properties have to be optimized across (potentially multiple) suppliers involved in the system development, which is a challenging task for a complex distributed system.

Automotive applications development cost is particularly impacted by bad design choices made at the early development stages but only detected later, often after implementation. In particular, timing-related failures are mostly raised very late in the development process, usually during the implementation, or even later at the system integration phase. Timing behavior is mostly addressed by means of measuring and testing, rather than through formal and systematic analysis. For this reason, innovative and complex functionalities are used to be not implemented in a cost-efficient manner.

Therefore, there is a strong need for an approach that would enable performing timing prop-

* This work is performed with the support of the EDONA project of the system@tic Paris region cluster.

erties verification at very early design phases. Thus, an early prediction of the system timing behavior can be done, resolving potential weak points in the design as soon as possible. In this context, quantitative analysis techniques [1] (such as scheduling or performance analysis) are good candidates for analyzing non-functional properties at an early stage. They allow designers to detect unfeasible real-time architectures, prevent costly design mistakes, and provide an analytical basis to assess design tradeoffs associated to resource optimization.

Moreover, in order to master the system complexity and assess system-level trade-offs seeking higher quality and dependability, model-based engineering (MBE) is gaining momentum in the automotive domain [2]. One of the advantages expected from this kind of approaches is their specific ability to exploit correct-by-construction and incremental design processes, thereby relying extensively on automated, or at least assisted, model transformations and synthesis, and formalizing computer-based correctness analysis.

In this paper, we propose a model driven approach to perform scheduling analysis for automotive systems during the early design phases. Our approach is based on the combination of two modeling languages: EAST-ADL2 [3], an architecture description language and MARTE, the OMG language for modeling and analysis of real time embedded systems [4]. This choice has particular benefits for our framework: On one hand, EAST-ADL2 is a domain specific language dedicated for the development of automotive systems. Therefore, it allows a good description of the electric/electronic architecture of automotive systems with a variety of concepts supporting most of structural features of automotive applications. On the other hand, MARTE is known by its rich expressive power to model system real time properties and constraints. This capability has been exploited to define a MARTE-based methodology whose objective is to complete application models with the information needed to perform scheduling analysis [11]. Actually, the methodology presented here, while inspired by the methodology in [11], has the particular objective of completing EAST-ADL models, to enable scheduling analysis at the design level.

The scheduling analysis is actually performed by the MAST tool [6] via an automatic transformation of MARTE models to MAST input models. This latter transformation is supported by the Optimum tool chain developed by CEA LIST.

The proposed model driven approach has been then successfully applied in the context of an automotive use case.

The paper is organized as follows: In the first section, we present the modeling process of EAST-ADL2 with respect to timing analysis needs. The second section shows how the MARTE modeling approach could contribute to complete our methodology. The third section introduces the EAST-ADL2/MARTE based methodological framework for modeling and analyzing automotive systems. Finally, we illustrate this work with an automotive use case in the fourth section.

2. EAST-ADL2 and Scheduling Analysis

EAST-ADL2 is an architecture description language defined as a domain specific language for the development of automotive electronic systems. It includes modeling entities to describe features, requirements, variability, software and hardware components.

As shown in Figure 1, the core concepts of the structural organization of EAST-ADL2 are dedicated to the description of the models at different abstraction levels: vehicle level, analysis level, design level and implementation level. The electronic functions/features are described at different levels of abstraction, reflecting the details refinement of the architecture. The different artifacts drive the functional decomposition of the functions from abstract models down to implementation in software components and hardware elements of the system architecture.

Let us notice that the implementation level is supported by AUTOSAR [8]

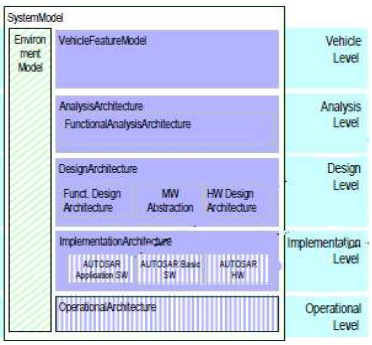

Figure1: EAST-ADL2 abstraction levels

Modeling of the electronic systems of a vehicle with EAST-ADL2 starts with capturing the functions at the Vehicle level providing this way the product line organization and description.

These functions are then realized at the analysis level by abstract entities describing models of software functions and devices that interact with the vehicle environment.
At the design level, models are refined including more implementation-oriented details that allow a subsequent software decomposition of the functional architecture. Devices are split into elements of the hardware architecture such as sensors or actuators, and the software parts for signal transformation (such as "*LocalDeviceManager*"). Middleware is modeled to project the platform specific services and functionality to the functional level. The hardware architecture that is introduced in parallel, captures the hardware entities as abstract elements (e.g. I/O, sensor, actuator, power, ECU, electrical wiring including communication bus) to describe the topology of the electronic architecture of the system. The overall structure is such that one or several entities can be later realized by AUTOSAR entities at the implementation level. More details about the modeling process of EAST-ADL2 can be found in [3].

In our approach, we suggest to perform scheduling analysis starting from the design level of EAST-ADL2. We think that this is the highest modeling level of EAST-ADL2 against which it will be possible to perform a scheduling analysis because at the previous levels, the so-called vehicle and analysis levels, there is no enough detailed information to do it. For example, there is no hardware description, no possibility to describe allocation, etc…

For timing analysis, EAST-ADL2 defines concepts in order to model timing and behavioral information of the system. All this concepts are related to a core concept for modeling the system architecture in EAST-ADL2 the "*ADLFunctionType*". This latter is used to describe system functions at the design phase.
In EAST-ADL2, behavior modeling relies on the definition of a set of elementary functions that are executed based on the assumption of synchronous run-to-completion execution (read inputs from ports, compute, and write outputs on ports).
EAST-ADL2 gives also means to define the activation patterns of an *ADLFunctionType*: the triggering of such function is defined either by time or by an event on one of its input ports. This is done through the concept "trigger" that defines the triggering parameters of ADL Functions.

As mentioned previously, the design level of EAST-ADL2 includes concepts for hardware and middleware modeling. For scheduling analysis in particular, the relative speed of the ECU and the data transmission rates of the bus are necessary to obtain absolute execution times and communication delays.

Nevertheless, many modeling features required for scheduling analysis of automotive applications are not supported by EAST-ADL2. A comprehensive summary of those features is provided in [5].
For example, EAST-ADL2 does not model explicitly OS tasks or the allocation of functions to these tasks. On the other hand, scheduling analysis methods are typically scenarios-based. This leads to a behavior model supported by the notion of end-to-end flows. This notion is not supported in EAST-ADL2.

In conclusion, EAST-ADL2 abstracts away many timing features and relies on AUTOSAR. That is why it is necessary to complete it with more scheduling oriented concepts in order to be able to perform scheduling analysis at the design level.
For this purpose, we suggest to use MARTE, as it provides a dedicated specific framework for performing scheduling analysis aware modeling. In the next section, we describe a set of UML extensions for supporting model-based schedulability analysis as supported by MARTE. Firstly, we catalog some essential modeling features to support state-of-the-art scheduling analysis of automotive architecture description. These features suffice for the purpose of this section, which is to provide an informal review of the expressive power provided by MARTE. A more detailed description of MARTE can be found in the MARTE specification (version 1.0 [4]) and in [2].

In our approach, we propose to use MAST (Modeling and analysis Set for Real Time Applications) to perform scheduling analysis. An interesting feature in MAST is its open source aspect; this aspect is important for us, it indicates that there is potential for enriching it to support more automotive-oriented features (e.g. implementing new automotive scheduling algorithms). Due to space limitation, a detailed description of this tool is not provided here, further information on MAST can be found via its website: [6].

3. MARTE and scheduling analysis modeling

We organize the features considered in two categories, system-oriented and design-oriented modeling features.

3.1 System-oriented features

These modeling features are related to the attributes and constraints of the targeted system itself (i.e., information completeness). Figure 3 shows a simplified canonical model of the modeling features required for scheduling analysis, which is discussed in this section.

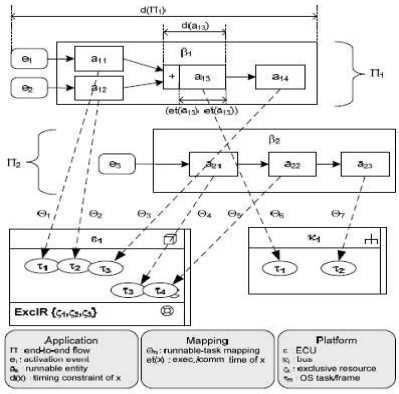

Figure 3: Simplified canonical model for scheduling analysis

Timing constraints: MARTE provides mechanisms to annotate timing requirements and constraints in models. Basic timing constraints include deadlines and maximum jitters. One key modeling feature is the concept of observation. This concept enables to mark specific points in models to anchor real-time assertions. Some common assertions have been predefined in ready-to-use patterns, such as jitters or conditional time constraints. These constraints can be for instance applied to the completion of a control/data flows of a functional chain or to arbitrary events within computation and communication chains.

End-to-end flows: Scheduling analysis methods are typically scenario-based, and consequently underlying behavior models rely mainly on the notion of *end-to-end flows* that one defines as follow: An *end-to-end flow* refers to a unique causal set of execution/communication functions triggered by an activation event (or logical combination of events).
In MARTE, *end-to-end flows* describe logical units of processing work in the system, which contend for the use of processing resources (e.g., processors and buses) [2]. Let us also notice that firstly, data and control can be part of the processing, and secondly, different kind of timing constraints can be attached to *end-to-end flows* (e.g., deadlines or output jitters).
One important feature in MARTE is that *end-to-end flows* can be represented in behavioral views (e.g., Sequence or Activity diagrams) complementing structural models. This approach allows modelers to specify multiple *end-to-end flow* configurations that could be likely related to (a) specific operational modes, (b) alternative execution chains, or (c) different quantitative scenarios of activation parameters or other non-functional annotations.

Activation events: Both event-triggered and time-triggered architectural patterns are involved in automotive applications. Event-triggered means that tasks are started, or messages are transmitted, following the occurrence of one (or a conjunction of) significant event(s): e.g., "a door has been opened"). Time-triggered architectures consist of tasks started, or messages transmitted, at predetermined points in time, usually periodically.
Whatever the architectural style used for specifying an application (often, mix approaches are applied), the activation models need to be formally specified to enable further timing analysis. In MARTE, activation models are denoted by means of *workload events*. It can be modeled under different forms: by known patterns (e.g., periodic, aperiodic, sporadic or burst), by irregular time intervals, by trace files, or by workload generator models (e.g., state machine models). *Workload events* also enable specifying additional parameters for periodic and aperiodic patterns such as jitters, burst parameters, and distribution probabilities.

SW and HW resources: What is needed for scheduling analyses is to take into account the impact of the computing platform of the embedded system, i.e. the operating system (OS) and the hardware resources on the software applications. For example, some scheduling analyses need to consider the overheads due to the OS and the stack of communication layers or throughputs and bandwidths of underlying networks. Among these aspects, access protocols to mutual exclusive resources are of paramount importance in scheduling analysis of modern multiprocessor architectures.
The MARTE analysis model distinguishes two kinds of processing resources: *execution hosts*, which include for example processors, coprocessors and controllers, and *communication hosts*, which include networks and buses.
Processing resources can be characterized by throughput properties such as processing rate, efficiency properties such as utilization, and over-

head properties such as blocking times and clock overheads.
The system model shown in Figure 3 thereby captures information about the applications and the available resources provided by the platform of the system. It defines also the mapping of application functions to OS resources, ECUs and buses.

3.2 Design-oriented features

The design-oriented features category relates to the modeling constructs and styles that serve to organize models and to improve the designer decision-making capability.

Application vs. Platform: In a typical automotive development process, application and platform descriptions evolve separately. Application artifacts center on functionality and control logic, while platform artifacts focus on ECU/bus selection, middleware layers, and OS services.
MARTE supports this separation of views at different abstraction levels. For the particular case of scheduling analysis as denoted in Figure 4, the modeling concepts are organized into a *workload behavior* model and a *resources platform* model. The former is dealing with application-specific annotations, whereas the second is dedicated to computing and communication annotations.

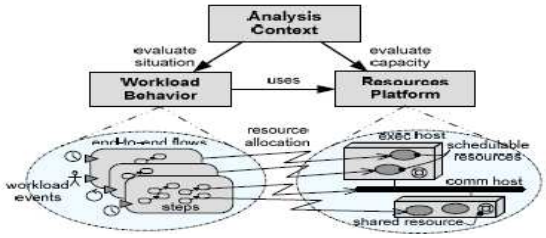

Figure 4. Expected organization of analysis-specific model elements in MARTE

Analysis scope: Due to the specific tools targeted by scheduling analysis, it is important to bind system model elements to a particular analysis or evaluation scope that represents a real time situation to be analyzed and for which the system timing constraints or properties are valid. As automotive electronic functions become more and more complex, there is often the need to represent a system by multiple analysis models, corresponding to different application-platform allocations, abstraction levels, operational modes, or different quantitative values of non-functional parameters.

For that purpose, MARTE defined the notion of *analysis context* (see Figure 4). An *analysis context* stands for the root concept used to collect relevant quantitative information for performing a specific analysis scenario. Starting with the analysis context and its parameters, a tool can follow the links of the model to extract the information that it needs to perform scheduling analysis. Analysis results can also be annotated back in application models to be taken into account, for instance, for architecture optimization or refinement.

Allocation: Performing system-level analysis requires taking into account the influence of underlying platforms in order to provide accurate results. To enable that, we will build an integrated global model where application and platform models are associated.
In MARTE, the allocation profile must be used for supporting this activity. By using this profile, it is then possible to denote a so-called *allocation model* that will define how the application model is mapped to the platform model. The allocation model is built orthogonally to the mapped models (the application and the platform models) and allows describing several possible mappings in order to explore different architecture options with respect to a set of functionalities and thereby reusing an architecture platform with different functions.
This allocation model may also include the associated timing attributes resulting from the allocation. For example, when allocating a runnable to a given OS task and ECU, one needs to specify its execution time (e.g. by calculated or measured). In MARTE, it is achieved by specifying *non-functional constraints* attached to allocations.

4. EAST-ADL2/MARTE based methodology for scheduling analysis aware modeling

In this section, we describe the EAST-ADL2/MARTE modeling framework for scheduling analysis aware modeling with respect to the abovementioned requirements.

4.1 General overview

UML2 encompasses many concepts of non-object-oriented design. For instance, activity and composite structure diagrams strive to support procedural and component-based design approaches. In particular, the UML capability for modeling component-based architectures is very

convenient since the automotive domain follows this approach.
System models, as described by component-based approaches, often do not fully match the models used as input in the state-of-the-art of scheduling analyses. Beyond syntactical mismatches (which can be solved by model transformation techniques), semantics shall be preserved in order to enable reliable and consistent analysis with respect to other activities of the development process such as code generation or simulation. In particular, the semantics of port communication and internal behavior of software components need to be conciliated with the causal model supported by scheduling analysis techniques. The modeling framework defined in this section aims at answering to this issue.
Our modeling architectural framework [9] is organized into a set of views, each providing a modeling concern of the system under study:

- *Application Components View*: This view describes the application structure organized in components, ports, interfaces and connectors.
- *System End-to-end Flow View*: In this view, we highlight the system-level behavior as organized in end-to-end data and control flows.
- *Platform View (SW & HW)*. It describes the software (which includes an OS task model) and hardware resources (which includes the processors, buses, devices, etc.), into specific configurations of allocation.
- *Allocation view*. This view describes the allocation of functions to software resources and the allocation of software resources to hardware resources.

The first view, *application component view*, describes the application logic. Although this model view is important for the initial design scope, system-level behavior, platform models and allocation are the most important for scheduling analysis.

4.2 Application component view

In this view, components and data/event flows between components are described with UML composite structure diagrams using EAST-ADL2 concepts for structure modeling. In this view, a component defines a self-contained entity of a system, which may encapsulate structured data and behaviors. The composite structure diagram consists of a main component (called "*ADLFunctionType*") that is internally structured as a set of components (UML parts) communicating between them (called "*ADLFunctionPrototypes*"). The concept of "*ADLPort*" defines an explicit interaction point through which components may be connected through a connector ("*ADLConnectorPrototype*"), and through which they can communicate.
In this view, both data-based and service-based communication are supported. This is done through "*ADLFlowPort*" and "*ADL-Clien/ServerPort*" of EAST-ADL2.

4.3 System end-to-end flows view

Starting with the application components view, we should be able to model explicitly component interaction, event/data flow, and activation events. The granularity of the entities involved in a model for scheduling analysis is often related to the choice of black or gray-box component modeling. For the first case, port-to-port delays should be considered, while for the second scenario internal "functions" may be considered. Whatever the granularity, scheduling analysis require to model the ordering of these functions. For that purpose, on applies the predecessor-successor patterns, with the possibility of multiple concurrent successors and predecessors, stemming from concurrent function joins and forks respectively.
To model this behavior information, we use the *end-to-end flows* of MARTE. However, design models are not always constructed to show *end-to-end flows* explicitly. Instead, they are implied by the presence of component interactions and internal behavior models. Therefore, there is a need to derive the *end-to-end flow* models from design models used e.g. for code generation.
All possible *end-to-end flows* in a given system can be generated by starting from the activation events, and then forming event sequences by recursively considering the output event set for the functions producing the events. Of course, many of the possible *end-to-end flows* may be meaningless since the appropriate triggering conditions may not hold.
As mentioned previously, MARTE *end-end-flows* can be represented in behavioral views complementing component models. According to the UML 2.0 specification [7], seven UML diagrams can be used to specify the behavior of a system: Activity, Sequence, Communication, Interaction Overview, Timing, Use Case and State Machine diagrams. In this modeling framework, we propose to use activity diagrams as modeling views for behavior scenarios. This choice has some particular benefits for our method:

- Activity diagrams are very intuitive representations of processing chains. They give a full support of the different precedence relationships between actions (joins, forks, etc.), that are not easily represented in sequence diagrams for example. In addition, activity diagrams allow modeling both service-based and data based communication while sequence diagrams allow only modeling service-base communication. Furthermore, unlike state machine, activity diagrams can explicitly represent end-to-end processing scenarios. These aspects qualify activity diagrams as the more suitable UML behavioral diagrams for developing scheduling analysis aware models.
- Structural elements, such as objects, components and actors, can be consistently modeled with UML::ActivityPartition in an activity diagram.
- Activity diagrams are also good candidates to model *workload behavior* annotations. An *UML::AcceptEventAction* can define a workload with the corresponding arrival pattern and related parameters (period, jitter, etc.).

In order to avoid ambiguities in the use of activity diagrams, we adopted a set of well-formed interpretations for MARTE annotations, which are described below. We will limit our description to the set of annotations that makes the model complete and consistent from the point of view of the schedulability analysis performed by MAST. Further details on the MAST system model may be found in [10]

- The activity diagram for representing a single system *end-to-end flow* is stereotyped as «saEndtoEndFlow». Deadlines on the *end-to-end flow* can be specified through the attribute "*end2endD*".
- The activation event that triggers the behavior of the system in the considered scenario needs to be specified and stereotyped as «gaWorkloadEvent». This way the arrival pattern of the event can be specified thanks to the attribute "*ArrivalPattern*". The value of this attribute priority is in form of a VSL expression (Value Specification Language [2]). For instance, a periodic event with inter-arrival period of 100ms will be annotated with an attribute *arrivalPattern* whose value is equal to *periodic (value=100, unit=ms)*.

Each activity representing the execution of an operation will be stereotyped as a «saStep» Each step is indeed an operation call described in the application component view. If the activity represents the sending of a message, it will be stereotyped as «saCommunicationStep». For both types of steps, it is required to specify the execution time of each step through the attribute "*execTime*". This execution time represents the host demand for executing the called function. Deadlines can be applied at the step level by assigning values to the deadline attribute. The *execTime* attribute value should be specified with the following syntax: *(min=value1,max=value2,unit=ms)*.

4.4 Platform view

As mentioned previously, MARTE allows for separating application and platform views, as well as describing a separated view for system allocations.

In order to comply with the MAST model, we propose to use the following set of annotations:

- The elements composing the software platform need to be stereotyped as «SchedulableResource» (such as tasks, threads, etc...) and «gaCommChannel» if the element represents a piece of logical execution or a sending of a message, respectively. Priorities of those software resources are specified through the "*schedParam*" attribute. The value of a priority is in form of a VSL expression [2]. It is also possible to specify the name of the scheduler the resource is assigned to, through the attribute "*host*". If no value is assigned to the attribute, it is assumed that the scheduler for this resource will be the scheduler assigned to the hardware resource (e.g. processor) where the software resource will be allocated. The allocation is done in the allocation model (see next section).
- The elements composing the hardware platform need to be stereotyped as «saExecHost» (if they represent resources with processing capacity) and «saCommHost» (if they are able to transmit/receive messages). These stereotypes let specify the scheduling parameters for the resource. For saExecHost is it possible to specify, for example, the *scheduling policy* (fixed priority, earliest deadline first, etc.) that is implemented by the scheduler assigned to the hardware resource, context switch overheads, the range of priorities accepted, the level of preemptability, clock overhead, etc... For saCommHost it is

also possible to specify the transmission mode (simplex, duplex, half-duplex).

4.5 Allocation

Due to the specific tools targeted by scheduling analysis, it is important to bind system model elements to a particular analysis scope, since automotive applications become more complex. MARTE supports this need via the concept of *analysis context*, which helps binding model elements to a particular evaluation scope. At the heart of this binding there is the allocation of functional steps executed in the considered scenarios to software resources (software resource platform) and the allocation of software resources to hardware resources (hardware resource platform). The allocation is carried out by specifying a composite diagram stereotyped as «saAnalysisContext».
Even if the *saAnalysisContext* pertains to the set of *end-to-end flows* the designer want to analyze, the allocation is not done per-flow basis. The intent, in fact, is to factorize the different steps to allocate in a unique view. As already said in the *end-to-end flow* section, each step is actually the call of a function operation owned by an application component. This relationship allows the direct allocation of functions on software resources. This way, the composite diagram contains three main components representing respectively: the application component views with functions to be allocated, the software platform and the hardware platform.
Each function of the application view (that has been called at least once in one of the *end-to-end flows*) will be associated to only one schedulable resource (e.g. OS task) and each connector in the application component view will be associated to a communication channel software resource. Moreover, each schedulable resource needs to be associated to a processing unit (*SaExecHost*) and each channel need to be associated to a bus (*saCommHost*). To carry out this association the software resource will be stereotyped as <<allocated>> and a dependency connector (type abstraction) between the function/connector and the resource will be drawn. The connector needs to be stereotyped as «allocate».

The allocation stereotypes allow the allocation definition through the attribute: *allocatedTo*. It lists the resources the element is allocated to. The stereotype «allocate» lets also specify constraints on the allocation and the type of allocation.

5. Illustration: the cruise control system

In this section, we illustrate our methodology with a complete EAST-ADL2/MARTE model of the cruise control and the scheduling analysis performed with MAST.
When activated, the cruise control system allows maintaining the vehicle speed to a set point value specified by the driver. The cruise control system is connected to a switch sensor that acquires the driver inputs (activate cruise control, cancel cruise control, set vehicle speed, etc…). The input is then processed in order to decide about the action to be taken by the system (authorize cruise activation, set speed, etc…)

5.1 The application component view

The cruise control system is modeled as a composite diagram as shown in the figure 5:

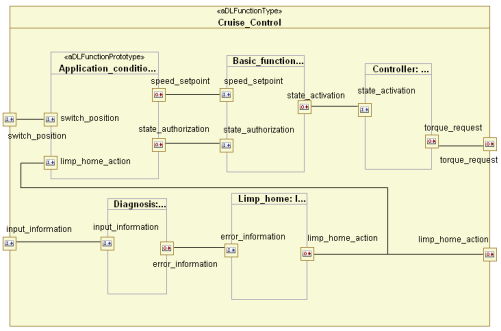

Figure 5: Component view of the cruise control system

The cruise control is composed of five basic modules: the application condition, the basic function and the controller module that are responsible for the control (performed every 40ms). The diagnosis and the limp home module are responsible for the failure detection and management that is performed each 10ms.
As shown in the previous figure, the cruise control is modeled as an "*ADLFunctionType*"; Its sub-functions are modeled as "*ADLFunctionPrototypes*". Each sub-function is an instance of an *ADLFunctionType* for which we defined an operation. (For example for the *controller* sub-function, we define an operation called "*call_controller*").

This operation will be used after when defining the *end-to-end flows* with MARTE.

5.2 The end-to-end flow view

In figure 6, we show the *end-to-end flows view*. Here we have identified two *end-to-end flows* that we called *control* and *failure management*.

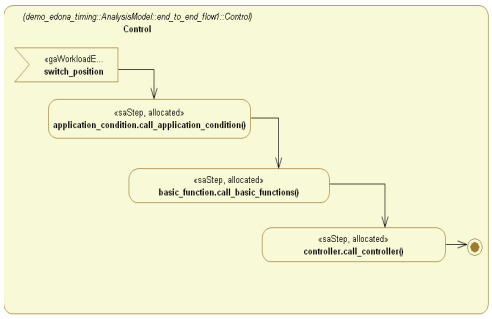

Figure 6: Control end-to-end flow

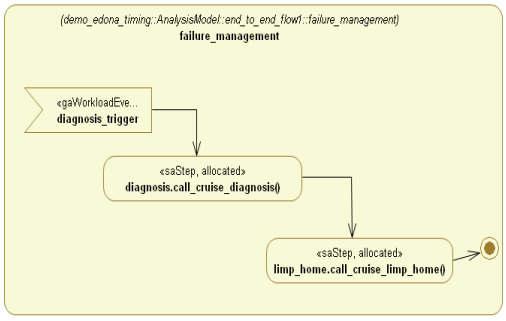

Figure 7: Failure management-end-to-end flow

Figure 6 shows the control *end-to-end flow*: The *PeriodicEvent* that triggers this end-to-end flow is stereotyped as «GaWorkloadEvent» and the corresponding annotation specifies the arrival pattern (periodic) and the period value (40 ms). This *end-to-end flow* then shows that the periodic event triggers a sequential flow of three steps which are operation calls, namely *call_application_condition*, *call_basic_function* and *call_controller*. These operation calls represent the execution of the logic described in the previous section for the corresponding functions *application_condition*, *basic_function* and *controller* (note that we did not specify the internal behavior of components; this is why we assume they have a unique operation whose semantic has been described in terms of functions in the previous section).
The *application_condition* takes as input all the inputs described on input ports in Figure 5. At the end of each period, its output triggers the execution of *basic_function* whose output triggers in turn *controller*. For each operation of the *end-to-end flows*, we specified its execution time. For example for the *controller* operation, we have specified *(min=1.0,max=3.0,unit=ms)*.

Let us remark how the *end-to-end flow* gives an immediate insight on the system behavior with respect to real-time constraints. The successor/predecessor relation and the activation event show that the deadline to be respected for each *end-to-end flow* is the cycle duration (event period) and that the *sum of all* execution times in each *end-to-end flow* needs to be lower than or equal to the cycle duration in order to have a schedulable system.

5.3 The allocation view

Figure 8 shows the binding between the application view and the software/ hardware platform of the cruise control. The diagram shows how the cruise control functions are allocated to two tasks (*fourty_ms_task* and *ten_ms_task*), which in turn are allocated on the ECU. Proper stereotypes are applied: «saAnalysisContext» for the whole diagram, «ShedulableResource» for Tasks and «SaExecHost» for the ECU.
Note that the allocation does not show tasks dependencies, which are instead represented by the *end-to-end flows*. Those task dependencies will be taken into account by the MAST analysis tool.

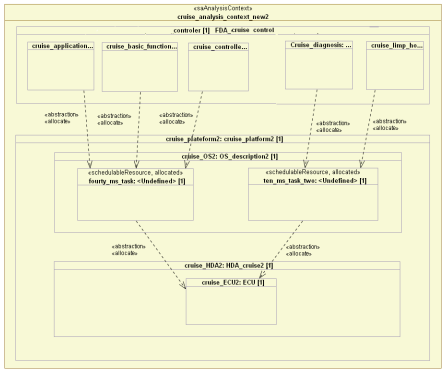

Figure 8: Allocation model for the cruise control

5.4 Schedulability analysis for the cruise control

As mentioned previously, in our framework, we propose to use MAST for schedulability analysis. Each *end-to end flow* modeled with MARTE is converted to a *transactio*n in MAST (a transaction represents interrelated activities that are executed in the system).

Figure 9 shows the MAST result. This result tells that the system is schedulable with a null slack. This means that the system is just schedulable and any change in its operation execution times will affect its schedulability (a system slack in MAST is the percentage by which we can increase all the execution times of all the operations in the system without jeopardizing its schedulability).

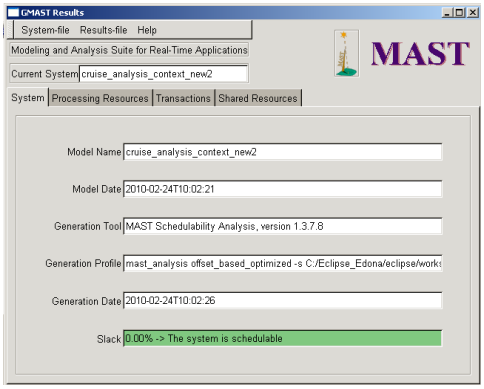

Figure 9: MAST analysis results

The table below shows the timing results calculated by MAST for the *control* and the *failure management* flows:

	Best response time (ms)	Worst response time (ms)
control	4	39.5
Failure management	2	7

Table 1: response times

As the table shows, the worst response times are lower than the periods. That is why the system remains schedulable.

6. Conclusion

In this paper, we presented an approach for modeling and analyzing automotive systems at the early design phase by combining the EAST-ADL2 and MARTE languages for modeling and the MAST tool for analysis.

The originality of this approach consists in performing scheduling analysis, usually used at the implementation phase, at a high abstraction level. This will allow designers to detect early unfeasible real-time architectures and hence prevent costly design mistakes that are currently detected later in the development process.

To be efficient, such approach has to be coherent with the analysis that must be done along all the development process. For example, performing scheduling analysis at the design level will increase the confidence degree in the architecture design and the allocation chosen. Hence, this will help him to move to the next design step. However, “design level analysis” must be coherent with “implementation level analysis”. This coherence requirement raises the question of which kind of implementation oriented information may be abstracted at the design level to enable early timing analysis but without redundancy with the implementation level.

7. References

[1] B. Selic, A Generic Framework for Modeling Resources with UML, IEEE Computer vol. 33 no.6, pp.64-69, June 2000.

[2] H. Espinoza. An Integrated Model-Driven Framework for Specifying and Analyzing Non-Functional Properties of Real-Time Systems. Ph.D. Thesis, University of Evry; 2007.

[3] P. Cuenot, P. Frey, R. Johansson, H. Lonn, M. –O. Reiser, D. Servat, R. Tavakoli Kolagari, D. J. Chen. Developing Automotive Products Using the EAST-ADL2, an AUTOSAR Compliant Architecture Description Language.

[4] Object Management Group, UML Profile for MARTE: Modelling and Analysis of Real-Time and Embedded systems, June 2008. OMG document: ptc/08-06-09

[5] S. Anssi, H. Espinoza, A. Albinet, S. Gérard, F. Terrier. On the Expressive Power of Modelling Languages for Enabling Scheduling Analysis of Automotive Applications.

[6] MAST website (http://mast.unican.es).

[7] Object Management Group. Unified Modeling Language: Superstructure Version 2.1.1 formal/2007-02-03.

[8] AUTOSAR Partnership (www.autosar.org).

[9] System and software engineering- architecture description. ISO/IEC WD4 42010. IEEE P42010/D6

[10] J. M. Drake, M. G. Harbour, J. J. Gutiérrez, P. L. Martinez, J. L. Medina, J. C. Palencia. “Modeling and Analysis Suite for Real Time Applications (MAST 1.3.7), Description of the MAST Model,” Report, Universidad De Cantabria, SPAIN, 2008

[11] H. Espinoza, S. Tucci-Piergiovanni, C. Mraidha, S. Gerard, "MARTE-Based Modeling Framework for Scheduling Analysis of Automotive Applications – Bridge to MAST Schedulability Analysis Tool –", DRT.LIST/SOL/09-280/CM, CEA LIST, December 2009.