MADES: A SysML/MARTE high level methodology for real-time and embedded systems

Imran Rafiq Quadri*, Andrey Sadovykh*
*Softeam, 21 Avenue Victor Hugo,
75016 Paris, France
Email:{Firstname.Lastname}@softeam.fr

Leandro Soares Indrusiak†
†University of York,
York, United Kingdom
Email:lsi@cs.york.ac.uk

***Abstract*—Rapid evolution of real-time and embedded systems (RTES) is continuing at an increasing rate, and new methodologies and design tools are needed to reduce design complexity while decreasing development costs and integrating aspects such as verification and validation. Model-Driven Engineering offers an interesting solution to the above mentioned challenges and is being widely used in various industrial and academic research projects. This paper presents the EU funded MADES project which aims to develop novel model-driven techniques to improve existing practices in development of RTES for avionics and surveillance embedded systems industries. MADES proposes a subset of existing UML profiles for embedded systems modeling: namely MARTE and SysML, and is developing new tools and technologies that support design, validation, simulation and eventual automatic code generation, while integrating aspects such as component re-use. In this paper, we first introduce the MADES language, which enables rapid system design and specification that can be then taken by underlying MADES tools for goals such as simulation or code generation. Finally, we illustrate the various concepts present in the MADES language by means of a car collision avoidance system case study.**

***Index Terms*—Real-Time and Embedded Systems, Model-Driven Engineering, SysML, MARTE, MADES language**

I. Introduction

Embedded systems have become an essential aspect of our professional and personal lives. From avionics, transport, defense, medical and telecommunication systems to general commercial appliances such as smart phones, gaming consoles; these systems with real time constraints: *Real-Time and Embedded Systems* (RTES) are now omnipresent, and it is difficult to find a domain where these miniaturized systems have not made their mark. The important characteristics of RTES include: low power consumption, reduced thermal dissipation and radiation emissions, among others; offering advantages and new opportunities to integrate more powerful, energy efficient processors, peripherals and related resources into the system.

A. Motivations

However, as computing power increases, more functionalities are expected to be realized and integrated into an embedded system. Unfortunately, the fallout of this complexity is that the system design (particularly software design) does not evolve at the same pace as that of hardware due to issues such as development budget limitations, reduction of product life cycles and design time augmentation. Additionally, development costs and time to market shoot up proportionally. Without the usage of effective design tools and methodologies, large complex RTES are becoming increasingly difficult to manage, resulting in critical issues and what has finally led to the famous *productivity gap*. The design space, representing all technical decisions that need to be elaborated by the design team is therefore, becoming difficult to explore. Similarly, manipulation of these systems at low implementation levels such as *Register Transfer Level* (RTL) can be hindered by human interventions and the subsequent errors.

Thus effective design methodologies are needed to decrease the productivity gap, while resolving issues such as related to system complexity, verification and validation, etc. Among several possibilities, elevation of design abstraction levels seems the most promising one. High abstraction level based system design approaches have been developed in this context, such as *Model-Driven Engineering* (MDE) [1] that specify the system using the UML (Unified Modeling Language) graphical language.

B. Elevating design abstraction levels

MDE enables high level system modeling of both software and hardware, with the possibility of integrating heterogeneous components into the system. It allows system level (application/architecture) modeling at a high specification level permitting several abstraction stages, each with a specific view point. This *Separation of Views* (SoV) enables a designer to focus on a domain aspect related to an abstraction stage thus permitting a transition from solution space to problem space. Using UML for system description increases the system comprehensibility as it enables designers to provide high-level descriptions of the system, that easily illustrate the internal concepts (data dependencies, hierarchy, etc.). These specifications can be reused, modified or extended due to their graphical nature. Thus, MDE offers an interesting solution to the above mentioned challenges and is being widely used in various industrial and academic research projects. It is supported by different technologies and tools such UML and related *profiles* for high level system specifications. Moreover, *Model transformations* [2] can then automatically generate executable models or code from these abstract high level design models.

C. Our contributions

Here, we first provide an overview of the MADES project followed by our contributions. MADES [3], [4] is an EU funded FP7 project which aims to develop novel model-driven techniques to improve existing practices in development of real-time and embedded systems for avionics and surveillance embedded systems industries. MADES proposes an effective subset of existing UML profiles for embedded systems modeling: namely SysML [5] and MARTE [6], and is developing new tools and technologies that support design, validation, simulation and eventual automatic code generation, while integrating aspects such as component re-use.

The contributions of this paper relate to presenting a complete methodology for the design of RTES using a combination of both SysML and MARTE, while avoiding incompatibilities resulting from simultaneous usage of both profiles. While a large number of works deal with embedded systems specifications using only either SysML or MARTE, we aim to present a combined approach and illustrate the advantages of using the two profiles. This contribution is significant in nature as while both these profiles provide numerous concepts and supporting tools, they are in turn difficult to be mastered by system designer. For this purpose, we present the MADES language, which associated with the namesake project focuses on an effective subset of SysML and MARTE profiles and proposes a specific set of unique diagrams for expressing different aspects related to a system. In the paper, an overview of the MADES language and the associated diagrams is presented, that enables rapid design and progressive composition of system specifications. The resulting models then can be taken by underlying MADES tools for goals such as simulation or automatic code generation.

Finally, we illustrate the various concepts present in the MADES language by means of an effective real life embedded systems case study: a *car collision avoidance system* that integrates the MADES language and illustrates the different phases of our developed design methodology.

The rest of this paper is organized as follows: section II illustrates some related works, while section III gives a brief overview of the MADES project, followed by a summarized description of the MADES language in section IV. Afterwards, section V presents our case study followed by a conclusion in section VI.

II. Related Works

While a large number of researches exist that make use of either SysML or MARTE for high level modeling of embedded systems, due to space limitations, it is not possible here to give an exhaustive description and we only provide a brief summary on some of the works that make use of SysML or MARTE based high abstraction levels and Model-Driven Engineering, for RTES design and implementation.

The MoPCoM project [7] uses MARTE profile to target modeling and code generation of reconfigurable embedded systems. While the project inspires from SysML concepts such as requirements and blocks, they are not fully integrated in the design flow. The project uses the IBM Harmony[1] process coupled with Rhapsody[2] UML modeling tool. Additionally, MoPCoM proposes two distinct flows for system modeling and schedulability analysis that increase design efforts. Similarly, eDIANA [8] is an ARTEMIS project that uses MARTE profile for RTES specification and validation. However, detailed specification of software and hardware aspects are not illustrated in the project. While TOPCASED [9] differs from MADES, as it focuses primarily on IDE infrastructure for embedded systems and not on particular implementations.

Project SATURN is [10] is another EU FP7 project that aims to use high level co-modeling approach for RTES simulation and synthesis goals. However, the project only takes SysML into account and proposes a number of UML profiles for co-simulation, synthesis and code generation purposes. The goal is to use carry out hardware/software modeling via these profiles and generate SystemC for eventual VHDL translation and FPGA implementation. Unfortunately, the project does not utilizes the MARTE profile for hardware/software co-design modeling. In [11], the authors provide a mixed modeling approach based on SysML and the MARTE profiles to address design space exploration strategies. However, the shortcomings of this approach is that they only provide implementation results by means of mathematical expressions and no actual experimental results were illustrated.

The OMEGA European project [12] is also dedicated to the development of critical real-time systems. However it uses pure UML specifications for system modeling and proposes a UML profile [13], which is a subset of the earlier UML profile for Scheduling, Performance and Time (SPT), that has been integrated in MARTE. The MARTES project emphasizes on combined usage of UML and SystemC for systematic model-based development of RTES. The results from this project in turn, have contributed to the creation of the MARTE profile. While INTERESTED [14] proposes a merged SysML/MARTE methodology where SysML is used for requirement specifications and MARTE for timing aspects, it does not proposes rules on combined usage of both profiles.

The MADES project aims to resolve this issue and thus differentiates from the above mentioned related works, as it focuses on an effective language subset combining both SysML and MARTE profiles for rapid design and specification of RTES. The two profiles have been chosen as they are both widely used in embedded systems design, and are complimentary in nature [15]. MADES proposes automatic generation of hardware descriptions and embedded software from high level models, and integrates verification of functional and non-functional properties, as illustrated in the following section.

III. MADES: goals and methodology

In this section, we provide a brief overview of the MADES design methodology, as illustrated in Fig.1. Initially, the high level system design models are carried out using the MADES

[1] http://www-01.ibm.com/software/rational/services/harmony/
[2] http://www-01.ibm.com/software/awdtools/rhapsody/

language and associated diagrams, which are represented later on in section IV. After specification of the design models that include user requirements, related hardware/software aspects and their eventual allocation along with schedulability analysis; underlying model transformations (*model-to-model* and *model-to-text* transformations) are used to bridge the gap between these abstract design models and subsequent design phases, such as verification, hardware descriptions of modeled targeted architecture and generation of platform specific embedded software from architecturally neutral software specifications. For implementing model transformations, MADES uses the Epsilon platform [16], that enables model transformations, code generation, model comparison, merging, refactoring and validation [17].

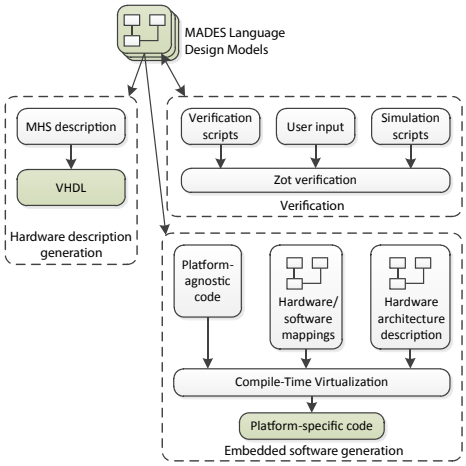

Figure.1: An overview of the global MADES methodology

Verification activities in MADES comprise of verification of key properties of designed concepts (such as meeting deadlines, etc.) and that of model transformations integrated in the design flow [18], [19]. For verification and simulation purposes, MADES uses the Zot tool [20], that permits verification of aspects, such as meeting critical deadlines, among others. While closed-loop simulation on design models enable functional testing and early validation.

Additionally, MADES employs the technique of *Compile-Time Virtualization* (CTV) [21], for targeting of non-standard hardware architectures, without requiring development of new languages or compilers. Thus a programmer can write architecturally neutral code which is automatically distributed by CTV over a complex target architecture. Finally, code generation (either in C/C++ or VHDL) can be carried out that can be eventually implemented on modern state of the art FPGAs. Currently MADES targets Xilinx FPGAs, however it is also possible to adapt to FPGAs provided by other vendors such as Altera or Atmel. Detailed description regarding the global MADES methodology can be found in [22].

IV. MADES LANGUAGE: SysML/MARTE SUBSET

Fig.2 gives an overview of the MADES language inherently present in the design flow for the initial design models. The MADES language focuses on a subset of SysML and MARTE profiles and proposes a specific set of diagrams for specifying different aspects related to a system: such as requirements, hardware/software concepts, etc. Along with these specific unique diagrams, MADES also uses classic UML diagrams such as *State* and *Activity* diagrams to model internal behavior of system components, along with *Sequence* and *Interaction Overview* diagrams to model interactions and cooperation between different system elements. Softeam's UML Modelio tool [23] enables full development of MADES diagrams and associated language, while a partial integration has been currently carried out for a MADES open source modeler, that is an extension of the Papyrus UML modeler [24]. We now provide a brief description of the MADES language and its related diagrams.

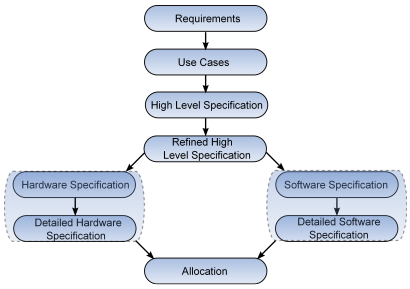

Figure.2: Overview of MADES language design flow

In the initial specification phase, a designer needs to carry out system design at high abstraction levels. This design phase consists of the following steps:

- ***System Requirements***: The user initially specifies the requirements related to the system. For this purpose, a MADES `Requirements Diagram` is utilized that integrates SysML requirements concepts.
- ***Use case Scenarios***: Afterwards, the system requirements are converted into use cases, described using MADES `Use Case Diagram` that encapsulate SysML use case concepts. This design phase is strongly related to the functional high level specification described subsequently.
- ***High Level Specification***: Each use case is converted into a SysML block by means of MADES `High Level Block` (or `Internal Block`) `Specification Diagram`, that contains SysML Block (or internal block) concepts respectively. This functionality is independent of any underlying execution platform and software details. It thus determines *what* is to be implemented, instead of *how* it is to be carried out.
- ***Refined High Level Specification***: The `Refined High Level Specification Diagram` models MARTE components, each corresponding to a SysML block. Here, MARTE's *High level Application Modeling* package is used to differentiate between active and passive components of the system.

The refined high level specification permits to link SysML and MARTE concepts while avoiding conflicts arising due

to parallel usage of both profiles [15]. The conflicts related to the two profiles are avoided as we do not mix SysML and MARTE concepts in the same diagram, but instead focus on a refinement scheme. Thus SysML is used for initial requirements and functional description, while MARTE is utilized for the enriched modeling of the global functionality and execution platform/software modeling.

Thus, once the refined functional description is completed, the designer can move onto the partitioning of the system in question: depending upon the requirements and resources in hand, he or she can determine which part of the system needs to be implemented in hardware or software. We now describe the different steps related to each design level by means of MARTE concepts.

Related to the MARTE modeling, an allocation between high level and refined high level specifications is carried out using a MADES `Allocation Diagram`. Afterwards, a co-design approach [25] is used to model the hardware and software aspects of the system. The modeling is combined with MARTE *Non Functional Properties* and *Timed Modeling* package to express aspects such as throughput, temporal constraints, etc. We now describe the hardware and software modeling, which are as follows:

- ***Hardware Specification***: The MADES `Hardware Specification Diagram` in combination with MARTE's *Generic Resource Modeling* package enables modeling of abstract hardware concepts such as computing, communication and storage resources. The design level enables a designer to describe the physical system albeit at a abstraction level higher than the detailed hardware specification level. By making use of MARTE GRM concepts, a designer can describe a physical system such as a car, a transport system, flight management system, among others.
- ***Detailed Hardware Specification***: Using the `Detailed Hardware Specification Diagram` with MARTE's *Hardware Resource Modeling* package allows extension and enrichment of concepts modeled at the hardware specification level. It also permits to model systems such as FPGA based System-on-Chips (SoCs), ASICs etc. A one-to-one correspondence usually follows here: for example, a computing resource typed as MARTE `ComputingResource` is converted into a hardware processor, such as a PowerPC or MicroBlaze [26], effectively stereotyped as MARTE `HwProcessor`. Once the detailed modeling is completed, an allocation diagram is then utilized to map the modeled hardware concepts to detailed hardware ones.
- ***Software Specification***: The MADES `Software Specification Diagram` along with MARTE's *Generic Resource Modeling* package permits modeling of software aspects of an execution platform such as schedulers and tasks; as well as their attributes and policies (e.g. priorities, possibility of preemption).
- ***Detailed Software Specification***: The MADES `Detailed Software Specification Diagram` and related MARTE's *Software Resource Modeling* are used to express aspects of the underlying *Operating System* (OS). Once this model is completed, an allocation diagram is used to map the modeled software concepts to detailed software ones: for example, allocation of tasks onto OS processes and threads. This level can express standardized or designer based RTOS APIs. Thus multi-tasking libraries and multi-tasking framework APIs can be described here.

Iteratively, several allocations can be carried out in our design methodology: a software to hardware allocation permits to associate schedulers and schedulable resources to related computing resources in the execution platform, once the initial abstract hardware/software models are completed. Subsequently this initial allocation could lead to further enriched allocation between the detailed software and hardware models (an allocation of OS to a hardware memory, for example). An allocation can also specify if the execution of a software resource onto a hardware module is carried out in a sequential or parallel manner. It should be noted that MARTE *Generic Component Modeling* package is used throughout the MADES diagrams dealing with MARTE stereotypes for describing the modeled concepts in a modular manner. Each MADES diagrams only contains commands related to that particular design phase, thus avoiding ambiguities of utilization of the various concepts present in both SysML and MARTE.

Additionally, MARTE enables analysis based on schedulability and performance criteria. Once a complete system is specified and eventually allocated; UML behavioral diagrams can be used in association with MARTE *Schedulability Analysis Modeling* package for describing the behavior of system components or the system itself. The schedulability analysis helps to determine aspects such as worst case execution times, missed deadlines etc; which can be expressed by the high level models and in turn used as input for a schedulability analysis tool, providing results that enable *Design Space Exploration* aspects by modifying in turn the high level design models.

Once the modeling aspects are completed, the high level models can be utilized by model transformations to produce intermediate or executable enriched models as well as code for eventual implementation. Afterwards, simulation can be carried out using third party tools to verify the correctness and functionality of the generated code before moving onto synthesis which enables to create actual implementation of a RTES, for example a prototype in case of an FPGA. For MADES, all these aspects related to verification, automatic code generation and hardware implementation have been detailed in [27]. We now present our case study that illustrates our combined SysML and MARTE design methodology.

V. Car Collision Avoidance System Case Study

We now provide the car collision avoidance system (CCAS) case study that is modeled in Modelio using the MADES language and underlying methodology. Verification aspects, automatic code generation and hardware implementation re-

lated to the CCAS have been illustrated in [27], and are not the scope of this paper.

The car collision avoidance system or CCAS for short, when installed in a vehicle, permits to detect and prevent collisions with incoming objects such as cars and pedestrians. The CCAS as shown in Fig.3 contains two types of detection modules. The first one is a radar detection module that emits continuous waves. A transmitted wave when collides with an incoming object, is reflected and received by the radar itself. The radar sends this data to an obstacle detection module, which in turn removes the noise from the incoming signal along with other tasks such as a correlation algorithm. The distance of the incoming object is then calculated and sent to a primary controller for appropriate actions.

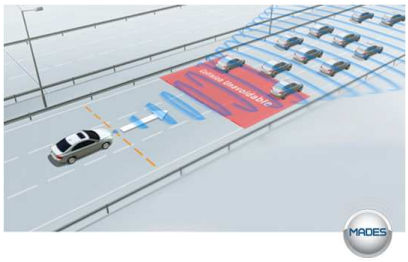

Figure.3: The CCAS installed on a car to avoid collisions with incoming objects

The image tracking module is the second detection module installed in the CCAS. It permits to determine the distance of the car from an object by means of image computation. The camera takes pictures of incoming objects and sends the data to a secondary controller, which executes a distance algorithm. If the results of the computation indicate that the object is closer to the car then a specified default value that means a collision can occur. The result of this data is then sent to the primary controller of the CCAS.

The primary controller when receives the data, acts accordingly to the situation at hand. In case of an imminent collision, it can carry out some emergency actions, such as stopping the engine, applying emergency brakes; otherwise if the collision is not imminent, it can decrease the speed of the car and can apply normal brakes. We now describe the CCAS system in detail subsequently.

A. SysML based modeling of CCAS

The CCAS design specifications start with SysML based modeling, which involves the initial design decisions such as system requirements and functionality description.

1) Requirement Specifications: We first move on to the system requirements that basically describe the overall needs, constraints and limitations of the system. Using the SysML inspired MADES *Requirements Diagram*, it is possible to describe the system requirements at the initial phase of system conception. In Fig.4, we illustrate the different requirements of the CCAS system. It should be mentioned that only the *functional* requirements of a system are described at this level.

Here, the different requirements for the CCAS are described: the `Global Collision Avoidance Strategy` determines the global requirement for the system which is to detect incoming objects by means of either the radar or the image tracking system. Additional requirements can be derived from this global requirement, such as the `Imminent Collision Strategy`, `Near Collision Avoidance Strategy`, `Additional Timing requirements` and `Changing Lanes Strategy`. It should be noted that this requirement specification is not complete in nature and has a strong relation with other MADES diagrams. More specifically, these specifications rely on use case scenarios and the functional blocks, described in turn in *High level Block Specification Diagram* for their completion, as illustrated later on. We thus will revisit this requirement diagram once we specify these other aspects.

Initially, the user describes the system requirements which state that if the distance from an object is less than 3 meters than the CCAS should enter in a warning state. If it remains in that state for 300 ms and distance is still less than 3 meters, then the CCAS should decrease car speed and alert the driver by means of an alarm and a Heads-up-display (HUD). If the distance falls to 2 meters, the CCAS should enter in a critical warning state. If it remains in that state for 300 ms and distance is still less than 2 meters, then CCAS should apply emergency brakes, deploy airbags and alert the driver as well.

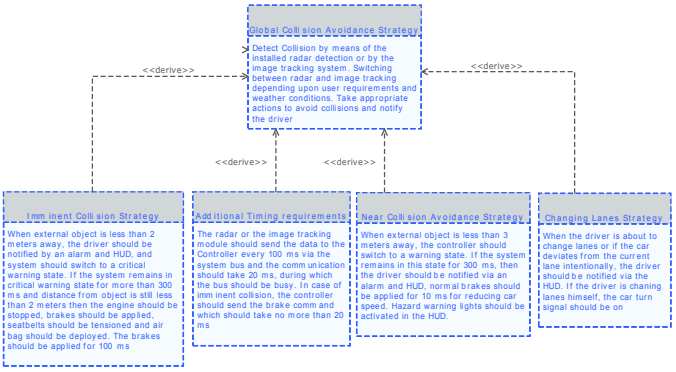

Figure.4: Initial requirements specifications for the CCAS

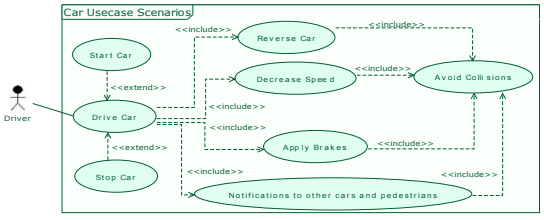

Figure.5: The different case scenarios related to the CCAS

2) Use case scenarios: Once the requirement phase is partially completed, we move onto describing the use case scenarios associated with our system via the MADES *Use Case Diagram*. Here in Fig.5, we describe the different scenarios associated with the car on which the CCAS is installed. The

figure illustrates some basic scenarios, such as the car being started, stopped or being driven. The `Avoid Collisions` scenario makes use of other specified scenarios and is the one that is related to the system requirements described in the section.V-A1.

3) High Level Specification: Once the requirements and use case scenarios of our system are specified; we move onto the functional block description of the CCAS system as described in Fig.6. For this, MADES *High Level Block Specification* or *High Level Internal Block Specification* diagrams are used. This conception phase enables a designer to describe the system functionality without going into details how the functionality is to be eventually implemented. Here the functional specification is described using SysML block definition diagram concepts. The functional description can be specified by means of UML concepts such as aggregation, inheritance, composition etc. Equally, hierarchical composition of functional blocks can be specified by means of internal blocks, ports and connectors. Here we use these concepts for describing the global composition of the CCAS. The `Car` block is composed of some system blocks such as a `Ignition System`, `Charging System`, `Starting System`, `Engine Component Parts`, `Transmission System` and finally the `Car Collision Avoidance Module` which is the main component related to our case study. Each functional blocks can be composed of internal blocks, however, this step has not been illustrated in the paper.

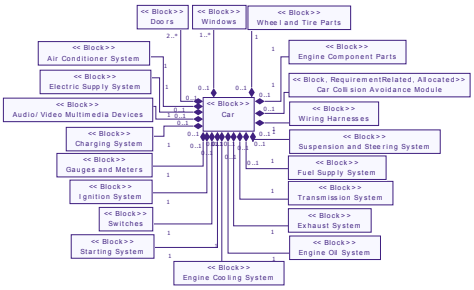

Figure.6: High level specification using SysML block concepts

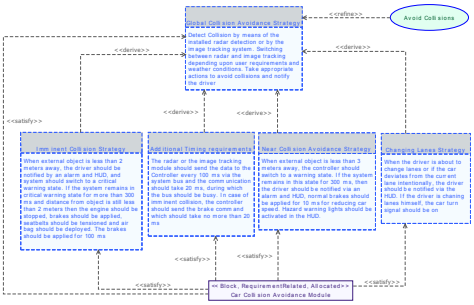

Figure.7: Completing the functional specifications of the CCAS

4) Completing the Requirement Specifications: Once we have finished the use case scenarios and functional block specification of our system, it is possible for us to complete the requirement specifications, as described in Fig.7.

As seen here, a related use case scenario and a functional block has been added to the figure, which helps to complete and satisfy the high level system requirements. It should be noted here, that as seen in Fig.7, only the `Car Collision Avoidance Module` block specified in the diagram is utilized to satisfy the global system requirements. Thus, these system requirements can be refined once an initial functional specification of system is completed. Since we are only interested in the `Car Collision Avoidance Module` and not other functional blocks of the car, it is this module that is the focus of the subsequent design phases.

B. MARTE based modeling of CCAS

Once the initial design descriptions have been specified, it is possible to partition and enrich the high level functionalities. For this, MARTE concepts are used to determine which parts of the system are implemented in software or hardware along with their eventual allocation. Additionally, MARTE profile enables to express non-functional properties related to a system, such as throughput, worst case execution times, etc. We now describe the MARTE based design phases in the subsequent sections.

1) Refined High Level Specification: We now turn towards the MARTE based modeling of the CCAS. All necessary concepts present at the *High Level Specification Diagram* correspond to an equivalent concept at the *Refined High Level Specification Diagram*. Since we are only interested in the `Car Collision Avoidance Module` at the higher level specifications, an equivalent MARTE component is created. The `RH_Car Collision Avoidance Module` is stereotyped as a MARTE `RtUnit` that determines the active nature of the component. Fig.8 shows the related modeling of this concept. The `RtUnit` modeling element is the basic building block that permits to handle concurrency in RTES applications [6]. It should be mentioned that component structure and hierarchy should be preserved between the two high level specification diagrams. As in this particular example, no hierarchical compositions are present at the high level specifications for `Car Collision Avoidance Module`, they are equally not present in the underlying refined high level specifications.

Figure.8: Refined high level specification of the CCAS

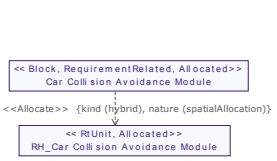

Figure.9: Allocation between high level/refined high level specifications

2) Allocating High Level and Refined High Level Specifications: Afterwards, an allocation using the MADES *Allocation*

Diagram is used to map the high level specification concepts to the refined high level specification concepts. This aspect is represented in Fig.9. Using the MARTE *allocation* mechanism, we express that the allocation is *hybrid* (both structural and behavioral aspects are thus related from source to target) and *spatial* in nature.

3) Clock Specification: Once the initial specification has been carried out, modeling of hardware and software aspects of the required functionality is possible in a parallel manner. For that purpose, we first model the different clock types which are used by the execution platform of the CCAS, as illustrated in Fig.10. Here, an initial ideal clock type serves as the basis for the `Hardware` and `System` clock types. All the clocks types are discrete in nature using the MARTE *Time* package.

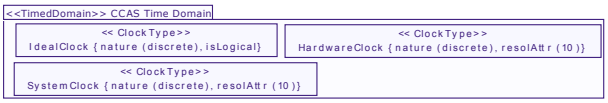

Figure.10: Specification of clock types related to CCAS

We first define time domain to contain the clock types related to the system. For this, a package `CCAS Time Domain` stereotyped as `TimedDomain` is created, respecting the timing notions in MARTE. Afterwards, we specify the clock types present in our system as illustrated in the figure. We specify three clock types: `IdealClock`, `SystemClock` and `HardwareClock`; all appropriately stereotyped as `ClockType`. The `IdealClock` is the basic clock type present in the system referencing a certain frequency, while the other two reference this basic clock and run at much higher frequencies.

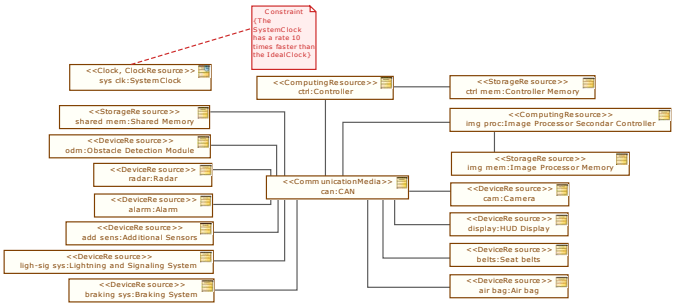

Figure.11: Abstract hardware specification of CCAS

4) Hardware Specification: At the hardware specification level, we first model the abstract hardware concepts of the execution platform in Fig.11. The abstract hardware modeling of the execution platform of the CCAS contains primary and secondary controllers for radar/image tracking modules, their respective local and shared memories, system clock and additional hardware resources (radar and camera modules, braking system, etc.); all of which communicate via a CAN bus. The MARTE *GRM* package stereotypes are applied onto the different hardware resources: for example `ComputingResource` for the primary and secondary controllers, `StorageResource` for the local and shared memories, `CommunicationMedia` for the `CAN` bus, while `DeviceResource` is used for the other hardware components. Here, the hardware specification also contains a clock `sysclk` of the type `SystemClock` specified earlier in Fig.10. Here using the MARTE *Time* package, we add a clock constraint onto the clock, specifying that this clock (and related clock type) runs at a rate 10 times faster than that of the ideal clock.

The hardware specification contains different hardware components which themselves are either further composed of sub components, or have internal behaviors, expressed by means of classic UML behavioral diagrams. We now describe the internal behavior of three hardware components:

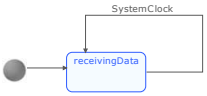

Figure.12: Behavior of the radar module present in the CCAS

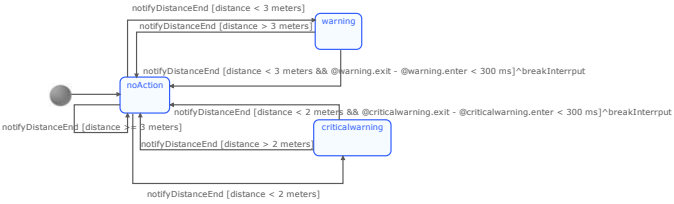

Figure.13: Behavior of the primary controller of the CCAS

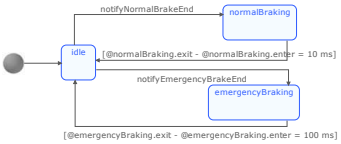

Figure.14: Internal behavior of the braking system

In Fig.12, we describe the internal behavior of the `Radar` component by means of a state machine diagram. The `RadarBehavior` state machine is stereotyped as `TimedProcessing` (not shown in the Figure). This permits to bind the processing of this behavior to time by means of a clock. Here the Radar remains in a single `receivingData` state and continues to send data to the primary controller at each tick of the `SystemClock`, every 100 ms. While Fig.13 illustrates the internal behavior of the primary controller. The controller contains three states, `noAction`, `warning` and `criticalwarning`. The controller initially remains in the `noAction` state when distance from incoming objects is greater than 3 meters. If the distance decreases to less than 3 meters, the controller switches to either `warning` or `criticalwarning` state (depending upon the the related condition) and sends a braking command to the `Braking System`. Fig.14 illustrates behavior of the `Braking System`. It switches to either the `normalBraking` or the `emergencyBraking` state, depending upon receiving a particular command from the primary controller.

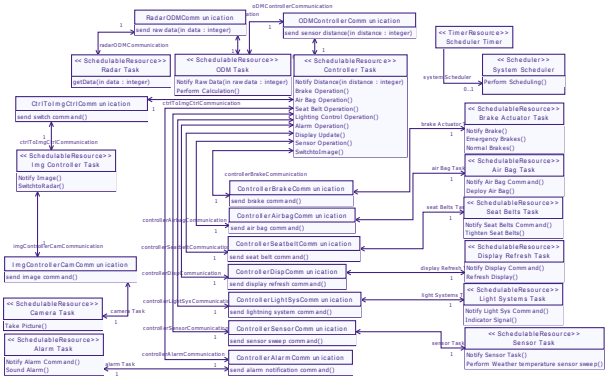

Figure.15: Software specification of the CCAS

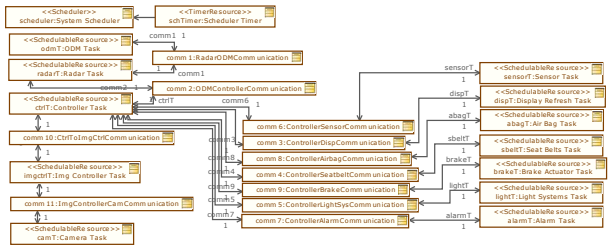

Figure.16: Software specification of the CCAS (Instance level)

5) Software Specification: We now turn towards modeling of the software specification of the execution platform of the CCAS, as displayed in Fig.15. Here, schedulable tasks related to the hardware modules are modeled along with their communications. A scheduler is also present that manages the overall scheduling based on a fixed priority algorithm. The different tasks are stereotyped as `SchedulableResource`, indicating that they are scheduled by means of a `System Scheduler`, itself appropriately stereotyped as a `Scheduler`. Each task contains a number of operations, indicating the functionality related to that particular task.

The software specification is also modeled at the instance level as illustrated in Fig.16, for an eventual allocation between the software/hardware specifications, and also for schedulability analysis modeling using UML sequence diagrams, as illustrated later on in section V-B12.

6) Allocating Software to Hardware: Once the hardware and software specifications have been carried out, we carry out an allocation between the two using the MADES *Allocation Diagram*. Here in Fig.17, the majority of the tasks (such as `Brake Actuator Task`, `Air Bag Task`) are allocated to the primary controller by means of a *temporal* allocation. Similarly, the `Radar` and `ODM` tasks are allocated to their respective hardware modules by means of *spatial* allocations. While tasks related to the secondary controller such as `Camera Task` are mapped on to it by means of a *temporal* allocation. Finally, all the communications are allocated to the `CAN` bus. It should be noted that while the *Allocated* stereotype on the software and hardware concepts has been applied similarly to the concepts illustrated in Fig.9, they have not been displayed here for a better visualization.

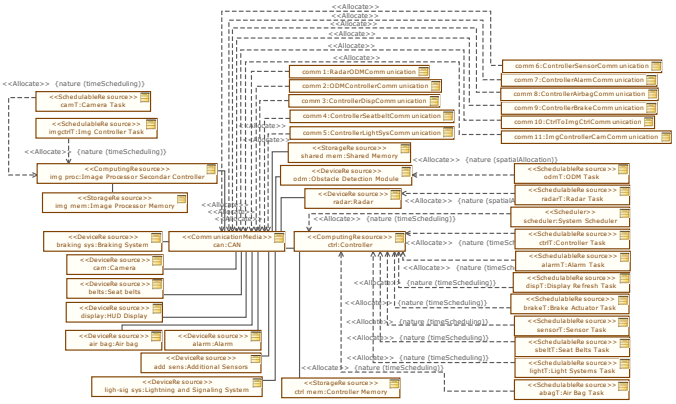

Figure.17: Mapping software resources to the hardware modules of CCAS

7) Detailed Hardware Specification: Once the initial abstract hardware specification has been modeled, the designer can move on to modeling of the detailed hardware specification which corresponds more closely to the actual implementation details of the execution platform. The structure of the detailed hardware specifications corresponds to the abstract specifications specified in Fig.11, but have been enriched with additional details: such as operating frequencies of the primary and secondary controllers, for example. All these aspects have been represented in Fig.18. Here, the modeled computing resources are stereotyped as `HwProcessor`, while memories are typed as `HwRAM`. The sensors, radar and display modules as typed as `HwI_O`, while remaining hardware concepts are stereotyped as `HwDevice`. Additionally the communication module, the `HW_ChannelBox` has been typed as a `HwMedia`. Finally, the execution platform also contains a `hwclk` clock of type `HardwareClock` with a related clock constraint.

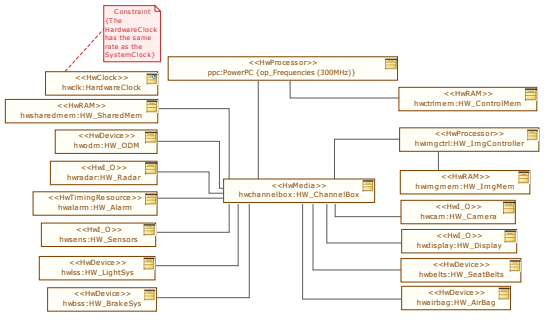

Figure.18: Detailed hardware specification of the execution platform

8) Detailed Software Specification: In parallel, a designer can model the detailed software specification, which basically correspond to expressing the concepts related to the underlying operating system of the CCAS. While it is possible to carry out detailed modeling of the operating system using MARTE *SRM* concepts, by adding relevant details such as semaphores, deadlocks, memory brokers etc; we chose to only illustrate some basic concepts related to a generic operating system.

Here as seen in Fig.19, the operating system has processes, each of which contains a number of threads. The `Operating System` is stereotyped according to *SRM* concepts as a `SwResource`, while the `Process` concept is typed as a `SwSchedulableResource` and a `MemoryPartition`. The latter stereotype indicates an address space which will be shared by the different threads associated with a process. Also, the `Thread` is itself typed as `SwSchedulableResource`.

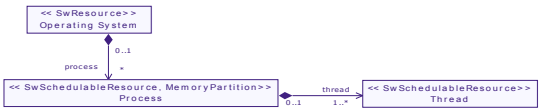

Figure.19: Detailed software specification of the execution platform

9) Allocating Hardware to Detailed Hardware Specifications: Once the detailed hardware specification has been modeled, we can carry out an allocation linking hardware to the detailed hardware specifications. This allocation corresponds to a one to one mapping between the two specifications, and thus determines the refinement of the execution platform. Thus, it enables to move from abstract classifications to details corresponding closely to RTL implementation. For example a `Controller` (and its related instance) is mapped to a `PowerPC` processor (and its instance), as shown in Fig.20. It should be mentioned that the *Allocated* stereotype applied on the allocated source/target concepts is not illustrated in the figure.

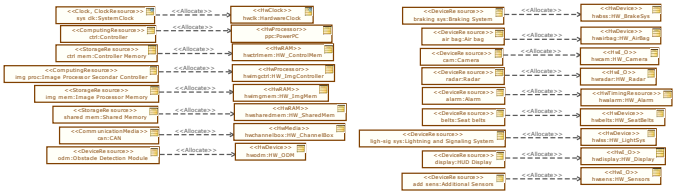

Figure.20: Allocating hardware/detailed hardware specifications

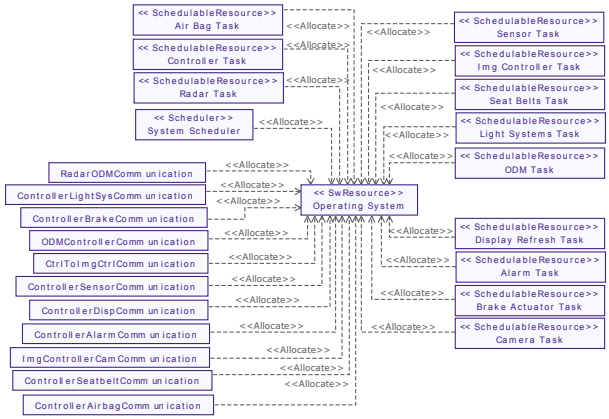

Figure.21: Allocating software/detailed software specifications

10) Allocation Software to Detailed Software Specifications: Similarly, the software specification is allocated onto the detailed software specification (the operating system in this case), in Fig.21. All the tasks and communications are allocated onto the operating system. While it is also possible to model the detailed software modeling in a detailed manner (different threads corresponding to the different tasks at software specifications) and then carry a one to one allocation, this aspect has not been carried out in the paper.

11) Allocating Detailed Software to Detailed Hardware Specifications: Finally, once all the detailed specifications related to the software and hardware aspects of the execution platform have been modeled, it is possible to carry out a final allocation from the detailed software to the detailed hardware specifications. Here, as seen in Fig.22, the operating system is allocated onto the local memory of the primary controller, by means of a spatial allocation.

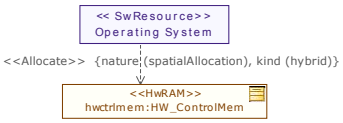

Figure.22: Allocating the detailed software and hardware specifications

12) Schedulability Analysis Specifications: Once the software and hardware modeling have been carried out, using the MADES language, it is possible to carry out schedulability analysis at the high level models. Here, in the following figures, we illustrate schedulability analysis aspects related to some modules of the execution platform. However, it is equally possible to analysis the whole system in question.

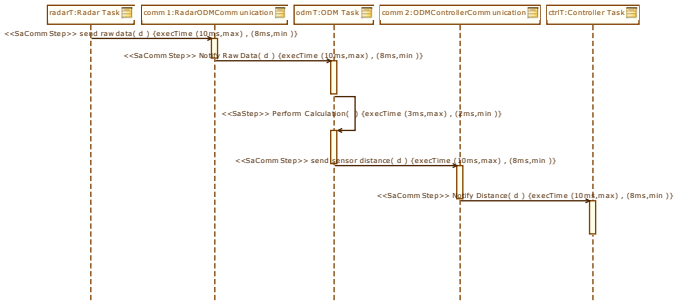

Figure.23: Sequence `SendSensorDistanceToContrl` for radar/controller communication

In Fig.23, the UML sequence diagram illustrates the communication flows between the different tasks of the execution platform. The `radarT` instance of the `Radar Task` sends data to the `odmT` instance of the `ODM Task` by means of a communication: `comm1` instance of `RadarODMCommunication`. The instance of the `ODM Task` after carrying out a noise reduction algorithm, sends the distance to the instance `ctrlT` of the `Controller Task` by means of the `comm2` communication instance of `ODMControllerCommunication`. Using appropriate MARTE packages, it is possible to carry out schedulability analysis: such as determination of average and worst case execution times for these flows. For example, the `SaStep`

stereotype helps to express worst and best case execution times, while `SaCommStep` can additionally express the size of the message transmitted or received during the communication flow. Here, the sequence `SendSensorDistanceToContrl` is itself stereotyped as a `GaScenario` (not shown in the figure), indicating it is one of the possible scenarios related to the system.

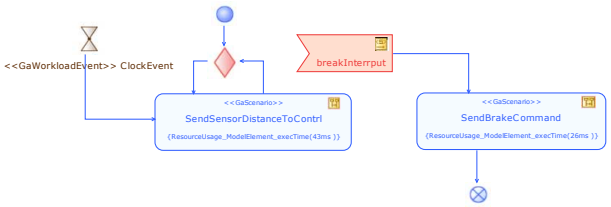

Figure.24: The `CCAS InteractionOverview` interaction

Finally, an interaction overview diagram as shown in Fig.24, illustrates the overall behavior related to two possible scenarios. The second scenario `SendBrakeCommand` has not illustrated due to space limitations. While it is possible to include other possible scenarios related to the CCAS (such as switching to camera based mode, applying normal brakes) and carry out a schedulability analysis of the whole CCAS execution platform, here only a partial analysis has been shown for clarification purposes. Here the figure indicates that once the CCAS system starts, it enters into a loop. It remains in the first scenario while keeps getting clock ticks at regular time intervals, as displayed by the `ClockEvent` typed as `GaWorkloadEvent`. It keeps executing the first sequence until a `brakeInterrupt` event occurs, causing the system to execute the sequence related to the second scenario concurrently. As the system does not stop its execution, no final node has been represented in the figure. Finally the interaction `CCAS InteractionOverview` is stereotyped as a `GaWorkloadBehavior` (not shown in the figure). This stereotype is used to specify a set of related system-level operations, each associated with its respective behavior and executed on the basis of associated (workload) events.

VI. Conclusions

This article aims to present a complete methodology integrated in the EU MADES project, for the design and development of real-time and embedded systems using an effective subset of UML profiles: SysML and MARTE. The paper presents its contributions by proposing an effective subset of the two profiles, forming the basis of MADES language and proposes unique set of diagrams to increase design productivity, decrease production cycles and promote synergy between the different designers/teams working at different domain aspects of the global system in consideration. Our MADES methodology could inspire future revisions of the SysML and MARTE profiles and may eventually aid in their evolution. Finally, the different language concepts and associated diagrams in the methodology have been illustrated in a case study related to a car collision avoidance system.

VII. Acknowledgements

This research presented in this paper is funded by the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement no. 248864 (MADES).

References

[1] OMG, "Portal of the Model Driven Engineering Community," 2007, http://www.planetmde.org.
[2] S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," *IEEE Software*, vol. 20, no. 5, pp. 42–45, 2003.
[3] A. Bagnato et al, "MADES: Embedded systems engineering approach in the avionics domain," in *First Workshop on Hands-on Platforms and tools for model-based engineering of Embedded Systems (HoPES)*, 2010.
[4] MADES, "EU FP7 Project," 2011, http://www.mades-project.org/.
[5] Object Management Group Inc, "Final Adopted OMG SysML Specification," mai 2006, http://www.omg.org/cgi-bin/doc?ptc/06-0504.
[6] OMG, "Modeling and Analysis of Real-time and Embedded systems (MARTE)," 2010, http://www.omg.org/spec/MARTE/1.0/PDF.
[7] A. Koudri et al, "Using MARTE in the MOPCOM SoC/SoPC Co-Methodology," in *MARTE Workshop at DATE'08*, 2008.
[8] EDIANA, "ARTEMIS project," 2011, http://www.artemis-ediana.eu/.
[9] TOPCASED, "The Open Source Toolkit for Critical Systems," 2010, http://www.topcased.org/.
[10] W. Mueller et al., "The SATURN Approach to SysML-based HW/SW Codesign," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2010.
[11] M. Mura et al, "Model-based Design Space Exploration for RTES with SysML and MARTE," in *Forum on Specification, Verification and Design Languages (FDL 2008)*, 2008, pp. 203–208.
[12] Information Society Technologies, "OMEGA: Correct Development of Real-Time Embedded Systems ," 2009, http://www-omega.imag.fr/.
[13] L. Ober et al., "Projet Omega : Un profil UML et un outil pour la modelisation et la validation de systemes temps reel," 2005, pp. 73: 33–38.
[14] INTERESTED, "EU FP7 Project," 2011, http://www.interested-ip.eu/index.html.
[15] H. Espinoza et al, "Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems," in *ECMDA-FA'09*. Springer-Verlag, 2009, pp. 98–113.
[16] D.S. Kolovos et al, "Eclipse development tools for Epsilon," in *Eclipse Summit Europe, Eclipse Modeling Symposium*, 2006.
[17] N. Matragkas et al., "D4.1: Model Transformation and Code Generation Tools Specification," Tech. Rep., 2010, http://www.mades-project.org/.
[18] L. Baresi et al., "D3.1: Domain-specific and User-centred Verification," Tech. Rep., 2010, http://www.mades-project.org/.
[19] ——, "D3.3: Formal Dynamic Semantics of the Modelling Notation," Tech. Rep., 2010, http://www.mades-project.org/.
[20] M. Pradella et al, "The Symmetry of the Past and of the Future: Bi-infinite Time in the Verification of Temporal Properties," in *ESEC-FSE'07*. New York, NY, USA: ACM, 2007, pp. 312–320.
[21] I. Gray and N. Audsley, "Exposing non-standard architectures to embedded software using compile-time virtualisation," in *International conference on Compilers, architecture, and synthesis for embedded systems (CASES'09)*, 2009.
[22] Ian Gray et al., "Model-based hardware generation and programming - the MADES approach," in *14th International Symposium on Object and Component-Oriented Real-Time Distributed Computing Workshops*, 2011.
[23] Modelio, "UML Modeling tool," 2011, www.modeliosoft.com.
[24] Papyrus, "Open source tool for UML modeling," 2011, http://www.papyrusuml.org/.
[25] D.D. Gajski and R. Khun, "New vlsi tools," *IEEE Computer*, vol. 16, pp. 11–14, 1983.
[26] Xilinx, "MicroBlaze Soft Processor Core ," 2011, http://www.xilinx.com/tools/microblaze.htm.
[27] A. Bagnato et al, "D1.3: MADES Initial Approach Guide," Tech. Rep., 2010, http://www.mades-project.org/.