MADES FP7 EU Project: Effective High Level SysML/MARTE Methodology for Real-Time and Embedded Avionics Systems

Imran R. Quadri*, Etienne Brosse*, Ian Gray†, Nicholas Matragkas†, Leandro Soares Indrusiak†, Matteo Rossi‡, Alessandra Bagnato§ and Andrey Sadovykh*
*Softeam, France
{FirstName.LastName}@softeam.fr
†University of York, United Kingdom
{iang, nikos, lsi}@cs.york.ac.uk
†Politecnico di Milano, Italy
rossi@elet.polimi.it
†TXT e-solutions, Italy
alessandra.bagnato@txtgroup.com

***Abstract*—The paper presents the EU funded MADES FP7 project, that aims to develop an effective model driven methodology to evolve current practices for the development of real time embedded systems for avionics and surveillance industries. In MADES, we propose an effective SysML/MARTE language subset and have developed new tools and technologies that support high level design specifications, validation, simulation and automatic code generation, while integrating aspects such as component re-use. The paper first illustrates the MADES methodology by means of a car collision avoidance system case study, followed by the underlying MADES language design phases and tool set which enable verification and automatic code generation aspects, hence enabling implementation in execution platforms such as state of the art FPGAs.**

***Index Terms*—Model Driven Engineering, UML, MARTE, SysML,Real-Time and Embedded Systems, FPGAs, Synthesis.**

I. Introduction

In recent years, continuous technological advances in hardware/software along with rapid increase in targeted application domains have led to new challenges in the design specification and implementation of real-time embedded systems (RTES). These systems are now omnipresent, and it is difficult to find a domain where RTES have not made their mark. Thus, 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 and efficient design tools are needed to decrease overall development costs and time-to-market, while resolving issues such as those related to system complexity, verification and validation, etc. High level 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, thus elevating the design abstraction levels.

MDE enables to elevate as well as partition the system design: by enabling parallel independent specifications of both system hardware and software; their eventual allocation, and the possibility of integrating heterogeneous components into the system. Usage of the UML graphical language increases system comprehensibility as it enables designers to provide high level descriptions of the system, easily illustrating the internal concepts (hierarchy, connections, dependencies etc.). The graphical nature of these specifications equally enables reuse or refinements, depending upon underlying tools and user requirements. Additionally, MDE encapsulates different technologies and tools such as UML and related *profiles* for high level system specifications. Finally, *Model transformations* [2] can then automatically generate executable models or code from these abstract high level design models.

The contributions of this paper relate to presenting an overview of the MADES project [3], [4] followed by our contributions. MADES 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. It proposes an effective subset of existing standardized UML profiles for embedded systems modeling: SysML [5] and MARTE [6], while avoiding incompatibilities resulting from simultaneous usage of both profiles. The MADES methodology integrates new tools and technologies that support high level SysML/MARTE system design specification, their verification and validation (V&V), component re-use, followed by automatic code generation to enable execution platform implementation.

One of the major contributions is related to presenting a complete methodology based on mixed SysML/MARTE

usage, for the design of RTES. While a large number of works deal with embedded systems specifications using only either SysML or MARTE, we 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 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 incremental composition of system specifications. The resulting models then can be taken by the underlying MADES tool set for goals such as component re-use, verification or automatic code generation, which are also briefly detailed in the paper.

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 design methodology and implementation approach.

The rest of this paper is organized as follows: section 2 gives a brief overview of the MADES project along with its underlying methodology, and integrated tool set. Afterwards, section 3 presents our case study and related design phases: modeling, verification and code generation, finally followed by a conclusion in section 4.

II. An overview of the MADES project

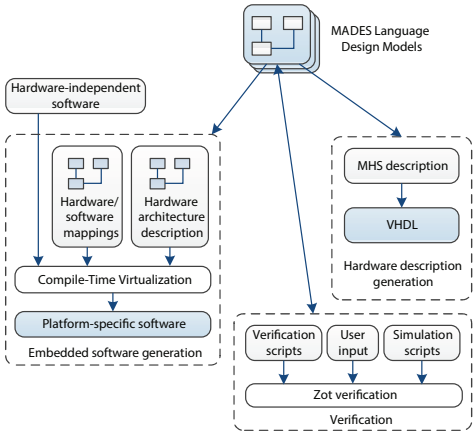

Figure.1: The global MADES methodology

In this section, we provide a brief overview of the MADES design methodology, as illustrated in Figure.1. Initially, the high level system design models are carried out using the MADES language and associated diagrams, which are represented later on in section II-A. After specification of the design models that include user requirements, related hardware/software aspects and their eventual allocation; 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 [7], that enables model transformations, code generation, model comparison, merging, refactoring and validation [8].

Verification activities in MADES include verification of key properties of designed concepts (such as meeting deadlines, etc.) and that of model transformations integrated in the design flow [9], [10]. For verification and simulation purposes, MADES uses the Zot tool [11], which permits the verification of, among others, aspects such as meeting of critical deadlines. While closed-loop simulation on design models enables functional testing and early validation.

Additionally, MADES employs the technique of *Compile-Time Virtualization* (CTV) [12], 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, via model transformations, code generation (for example; such as either in VHDL for hardware, and Real-Time Java for software) can be carried that can be eventually implemented on modern state-of-the-art FPGAs. Currently MADES model transformations target Xilinx FPGAs, however it is also possible for them to adapt to FPGAs provided by other vendors such as Altera or Atmel. A detailed description regarding the global MADES methodology can be found in [13].

A. MADES language: design phases and related diagrams

Figure.2 gives an overview of the underlying MADES language present in the overall methodology for the initial model based design specifications. 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 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 Modelio UML Editor and MDE Workbench [14] enables full support of MADES diagrams and associated language. We now provide a brief description of the MADES language and its related diagrams.

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.
- ***Initial Behavioral Specification***: Afterwards, initial behavioral specification is carried out by means of UML use

cases, interactions, state machines or activities during the preliminary analysis phase.

- ***Functional Specification***: Once the behavioral specifications are completed, they are then linked to SysML blocks (or internal blocks) by means of MADES *Functional Block* (or *Internal Functional Block*) *Specification Diagram*, that contains SysML block (or internal block) concepts. 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 Functional Specification***: This level refines SysML aspects into MARTE concepts: The *Refined Functional 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.

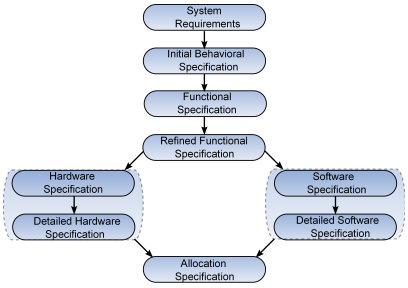

Figure.2: Overview of MADES language design phases

The refined functional specification phase links SysML and MARTE concepts but avoids conflicts arising due to parallel usage of both profiles [15]. The conflicts are avoided as we do not mix SysML and MARTE concepts in the same design phase, except the allocation aspects, present in both profiles. While the allocation concept is present both in SysML and MARTE, MARTE enriches the basic SysML allocation aspects and is thus the one adopted for our methodology. 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 along with their allocations, creating a clear separation between the two profiles. Afterwards, the designer can move onto the hardware/software partitioning of the refined functional specifications. These following steps are elaborated 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 [16] 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 an 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 modelling of 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 [17], effectively stereotyped as MARTE `HwProcessor`. Afterwards, 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.
- ***Clock Specification***: The MADES *Clock Specification Diagram* (not shown in Figure.2) is used to express timing and clock constraints/aspects. It can be used to specify the physical clocks present in the hardware platform and the related constraints, or logical clocks related to the software functionalities. This diagram makes use of MARTE's *Time Modeling* concepts such as clock types, clocks and related constraints.

Iteratively, several allocations can be carried out in our design methodology: an initial software to hardware allocation may allow associating schedulers and schedulable resources to related computing resources in the execution platform, once the initial abstract hardware/software models are completed, in order to reduce *Design Space Exploration* (DSE).

Subsequently this initial allocation can be concretized by further mapping of the detailed software and hardware models

(an allocation of OS to a hardware memory, for example), to fulfill designer requirements and underlying tools analysis results. 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. Interestingly, each MADES diagram only contains commands related to that particular design phase, thus avoiding ambiguities of utilization of the various concepts present in both SysML and MARTE, while helping designers to focus on their relative expertise. Additionally, UML behavioral diagrams in combination with MADES concepts (such as those related to verification) can be used for describing detailed behavior of system components or the system itself.

Finally, the MADES language also contains additional concepts used for the underlying model transformations for code generation and verification purposes, which are not present in either SysML or MARTE. A detailed description of these aspects can be found in [18].

Once the modeling aspects are completed, verification and code generation can be carried out, as explained subsequently.

B. Verification and Validation using Zot

Verification is carried out by transforming MADES diagrams into temporal logic formulae, using the semantics defined in [10]. These are, in turn, fed to the Zot verification tool, which signals whether the stated property holds for the modeled system or not, and in the latter case, returns a counterexample, i.e., a system trace violating the property.

In fact, once the temporal logic model is created from the diagrams describing the system, the Zot tool can be used in two ways: to check whether user-defined properties hold for the system; and to produce traces compatible with a formal model, in what amounts to a simulation of the system. The simulation capabilities of the Zot tool can be used, as described in [19], in combination with a simulation tool such as OpenModelica [20] to perform closed-loop simulations of the designed embedded system with its physical environment.

C. Model transformations and Code generation features

The underlying MADES model transformations focus on the following areas:

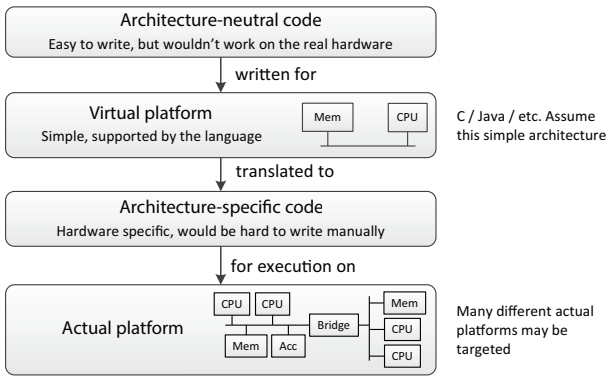

Figure.3: Overview of Compile Time Virtualization

- ***Generation of platform-specific software from architecturally-neutral software specifications***: Using Anvil J [21] that utilizes *Compile-Time Virtualization*, as seen in Figure.3, architecturally neutral code can be developed which is then ported to the real hardware. This enables hardware independence without redevelopment of the software functionality. Thus the software can be automatically refactored in case of change in details related to the underlying hardware. Additionally, software can be restructured throughout the target architecture: tasks can be moved from an overloaded processor without recoding the application. CTV will generate the split output code appropriately, and handle all communications and shared memory use to ensure that the code still operates correctly.
- ***Generation of hardware descriptions of the modeled target architecture***: The MADES transformations allow for the generation of implementable hardware descriptions of the target architecture from the input system model. The input model is created by means of Modelio and the MADES language detailed previously. The generated hardware could be a complex, heterogeneous system with a non-uniform memory architecture, but is supported and programmed by the software generated via the code generation transformations described earlier.
- ***Verification of functional/non-functional properties***: Verification capabilities are provided in the MADES framework by Zot, which requires a verification script and a set of properties to carry out verification. The verification script is automatically generated from a combination of the behavioral diagrams of the MADES language, particularly State, Sequence and Interaction Overview diagrams. A specification of the structure of the system is also needed which is derived from the MADES structural diagrams. Moreover, timing/clock information is required, which can be extracted from the *Clock Specification Diagram* present in the MADES language. The properties to verify must be solicited from the user. Results from Zot are fed back into the modeling tool in order to give the user feedback on them and locate errors, if any are found. The code generation facilities are used to integrate the back-end of the verification tool, which is Zot, with the front-end, which are the models expressed using the MADES language.
- ***Simulation of embedded systems***: In the case of simulation, the simulation tool requires an appropriate simulation script and a Modelica [22] model. The simulation script will be automatically generated from a combination of the behavioral diagrams of the MADES Language, particularly State, Sequence and Interaction Overview diagrams. A specification of the structure of the system, and of its links to the environment is also needed which is derived from the MADES diagrams. Finally, information about the environment will be required, and this information can be modeled using the MADES language.

- ***Traceability aspects***: The capture/maintenance of traceability information is of paramount importance in order to ensure consistency between the various artifacts or tools present in the MADES tool chain. Traceability support is provided for tracing the results of the verification activity back to the models, for tracing the generated code back to its source models and finally for tracing requirements to model elements such as use cases or operations, as well as to implementation files and test cases.

Together, these assist with mapping the programmer's code to complex hardware architectures, describing that architecture for implementation (possibly as an ASIC or on an FPGA) and verifying the correctness of the final system. Detailed descriptions about these model transformations, along with their installation and usage guidelines have been provided in [18], [23].

D. MADES Component Repository

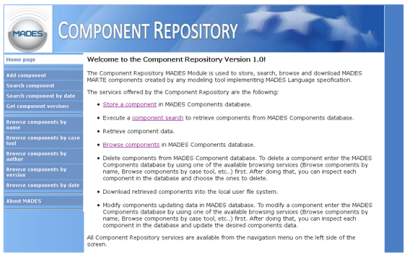

Figure.4: The MADES Component repository welcome page

The MADES Component Repository (CRP), as shown in Figure.4, is used to store, search and download MADES components created by the MADES developer with the Modelio UML Editor and MDE Workbench. It accesses a central MADES component database while offering various web services via a flexible graphical user interface to manage the components stored within the database, and the queries that have been performed on its contents. Thus the CRP enables *Intellectual Property* (IP) re-use, enabling designers to create, store or re-use IP blocks to build different applications, platforms or complete systems, while reducing design time. Complete details about the MADES CRP can be found in [18], [23].

III. Car Collision Avoidance Example (CCAS)

We now provide the car collision avoidance system (CCAS) case study that is modeled in Modelio using the MADES language and then carry out verification and code generation aspects using the MADES methodology. While MADES has two real-life case studies provided by Cassidian and TXT focusing on a ground based radar processing unit as well as an onboard radar control unit; the CCAS has been developed as a reference example to provide guidelines to the MADES partners as well as to the general embedded systems community for usage of the MADES language and the methodology.

The CCAS as shown in Figure.5 has been exhaustively modeled using the MADES language in the open source Modelio UML Editor/MDE Workbench, as shown in Figure.6. The right side of Figure.5 lists the different design phases of the CCAS. We refer the reader to [24] for a detailed description related to the modeling of the CCAS. Due to space limitations of the paper, we only focus on an extract of the CCAS and illustrate the associated modeling, verification, code generation and implementation aspects.

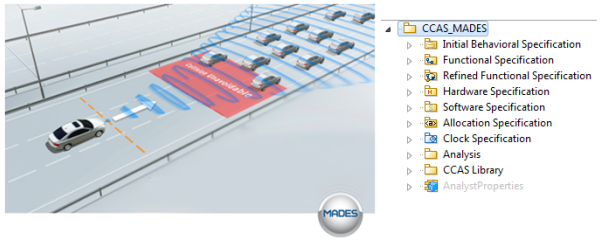

Figure.5: The CCAS installed on a car to avoid collisions with incoming objects

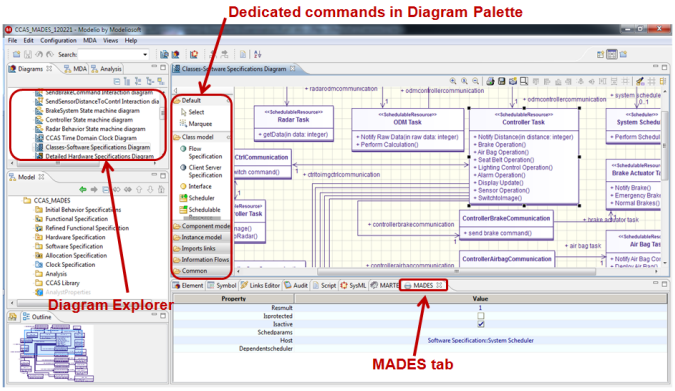

Figure.6: Open Source Modelio UML Editor/MDE Workbench

A. Modeling of CCAS using the MADES language

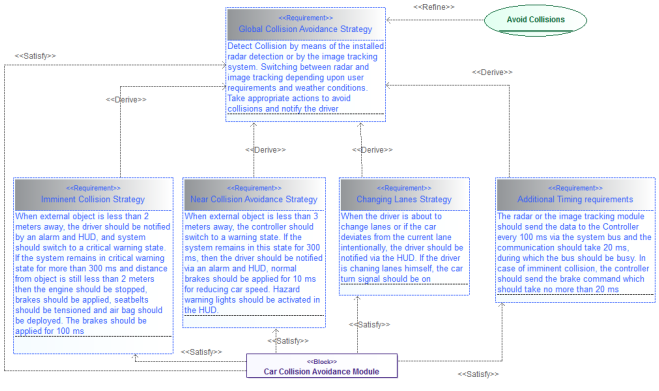

Figure.7: The Global system requirements related to the CCAS

The CCAS design specifications start with SysML based modeling, which involves the initial design decisions such

as system requirements, behavioral specifications and functionality description. Using the SysML inspired MADES *Requirements Diagram*, system requirements are described at the initial system conception phase, as illustrated in Figure.7. It should be mentioned that only the functional requirements of a system are described at this level. Afterwards, an initial behavioral specification phase is carried out, and in the particular case of CCAS, some use cases are specified to describe the different system scenarios of the car containing the CCAS module. Subsequently, the designer describes the functional description of the CCAS system, using SysML inspired *Functional/Internal Functional Block diagrams*. These behavioral and functional specifications are in turn used to complete the initial requirements, as shown in Figure.7. Here, the `Car Collision Avoidance Module` block and the `Avoid Collisions` use case help to complete the requirements. Figure.8 illustrates several design phases, mainly the behavioral, functional and refined functional specifications and the mapping between them. The functional specification illustrates the whole architecture of the car, of which we are mainly interested in the `Car Collision Avoidance Module` or CCAS. A use case is mapped to this block to describe its behavior, and the block is then refined into an equivalent MARTE concept using the refined functional specification phase.

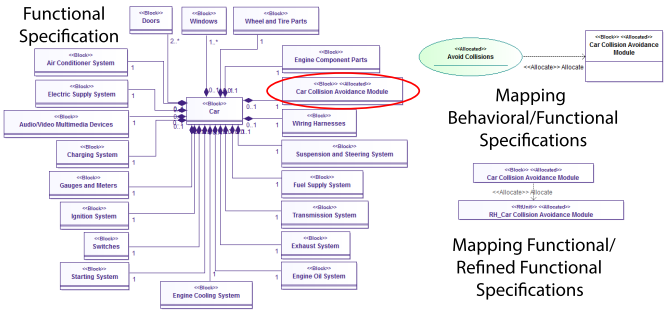

Figure.8: Extract of CCAS Functional/Refined Functional specifications

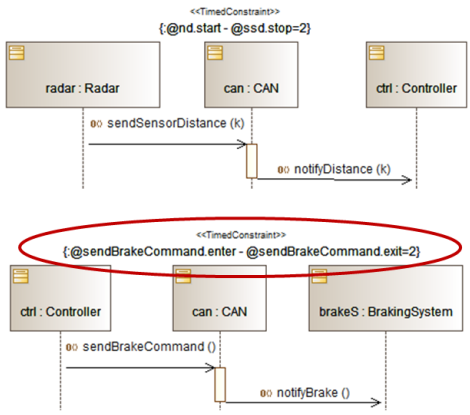

Figure.9: Timing constraints in a Sequence diagram

Afterwards, This refined MARTE component or the `RH_Car Collision Avoidance Module` is partitioned into hardware and software specifications using a Co-Design approach, as shown in Figure.10 and Figure.11, each consisting of a large number of concepts. As viewed in Figure.7, the CCAS also has some strict timing constraints, an example of which is shown in Figure.9

From hereon, we only focus on the image tracking aspect of the CCAS, which determines the distance of the car from an object by means of image computation. The `Image Processing Module` in Figure.10 and the `Img Processing Task` in Figure.11 represent the generic hardware and software aspects of the image tracking part of the CCAS, and are subsequently refined to detailed levels, corresponding to MADES detailed hardware and software specifications, shown in Figure.12 and Figure.13 respectively.

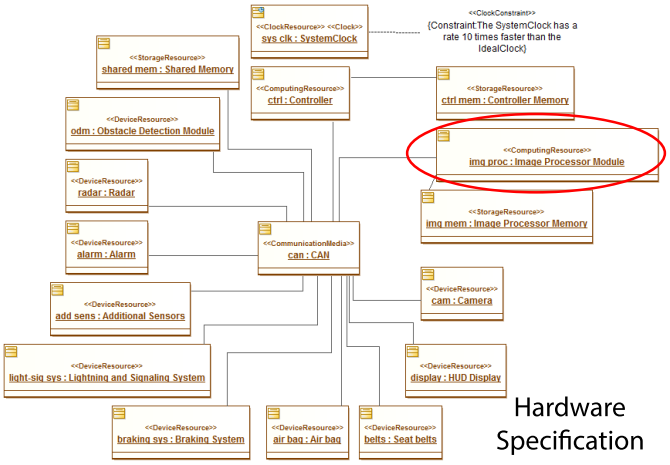

Figure.10: Extract of CCAS: Hardware Specification

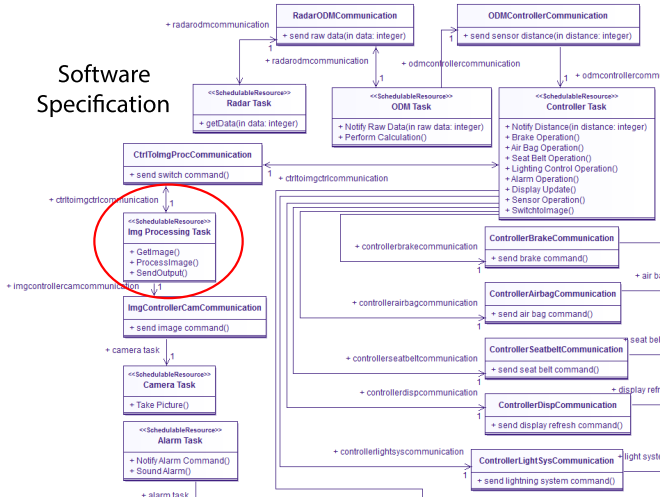

Figure.11: Extract of CCAS: Software specification

Figure.12 depicts the refined structure of the `Image Processing Module`, consisting of three processors with a distributed memory architecture, connected to a UART and a camera via PLB bus. The module also contains a system clock called `clock` of the type `mainClk` specified in the

MADES *Clock Specification Diagram*, but also shown here to represent a complete picture. The different classes/instaces are annotated with MARTE concepts, but also with certain MADES concepts defined in [18] relative to verification and code generation. Similarly, Figure.13 represents the refined internal structure of the `Image Processing Task`. It consists of several threads containing a number of operations, along with a shared object or mutex. Finally, Figure.14 describes the allocation of the detailed software/hardware aspects related to the CCAS image tracking part. Here, the `readThread` and `dctThread` instances are allocated to instances of `cpu1` and `cpu2` respectively, while the instances of `quantizeThread` and `OutputStage` are allocated to an instance of `cpu3`.

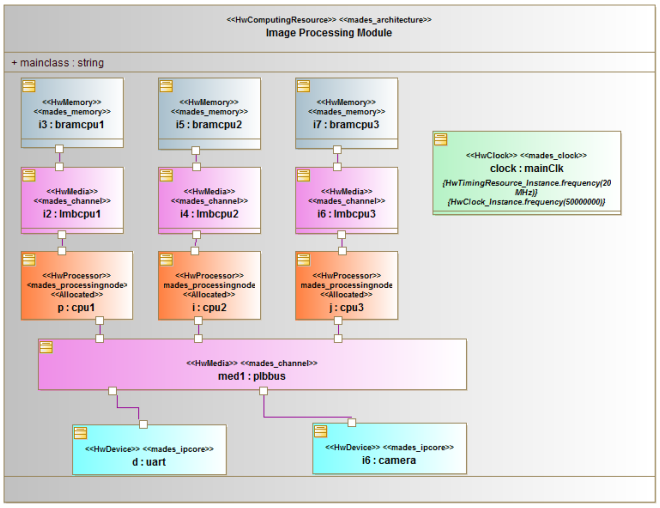

Figure.12: Detailed Hardware Specification of the Image Processing Module

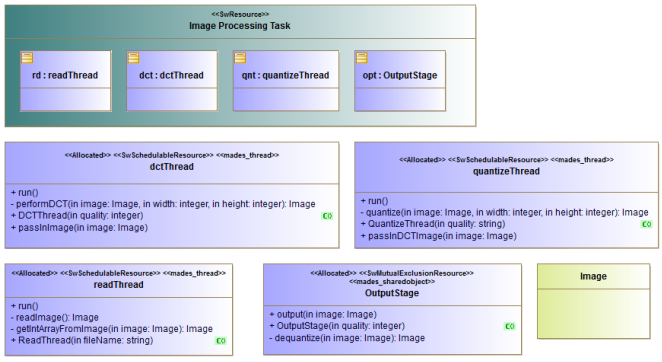

Figure.13: Detailed Software Specification of the Image Processing Task

B. Verification & Validation

After creating the MADES diagrams describing the behavior of the components of the system and of their interactions, the user can use suitable forms to define what properties of interest are to be checked on the model. For example, Figure.15 shows the dialog box of the MADES tool that allows the user to define properties to be verified and then to launch the actual verification. In this case, the property of interest states that, if the distance received by the controller remains less than 2 meters for 50 time units, within those 50 time units the system must have started to brake. Using the MADES tool chain, the user can then launch the verification on the modeled system, and in this case the tool reports a counterexample, meaning the system does not satisfy the stated property.

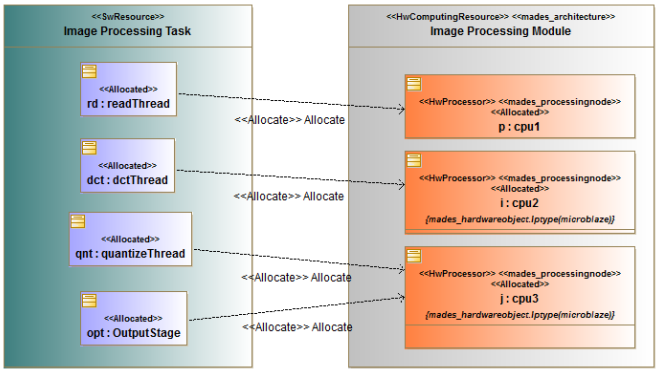

Figure.14: Allocation of detailed software/hardware aspects

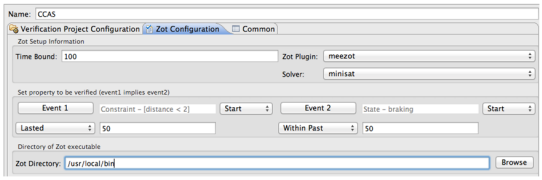

Figure.15: Temporal Logic: Example of temporal properties

C. Hardware and Software code generation

Once the modeling and verification phases have been carried out, it is possible to carry out hardware/software code generation by means of the model transformations previously described in section II-C. The hardware related model transformations take the MADES allocation model of Figure.14, and generate hardware description for input to standard commercial FPGA synthesis tools, such as Xilinx ISE and EDK tools. Presently, the model transformation are capable of generation *Microprocessor Hardware Specification* (MHS) which can be taken by Xilinx tools to generate the underlying hardware equivalent to that modeled using the MADES language, as seen in Figure.16.

Regarding the software code generation, the model transformations are capable of transforming user-provided, hardware-independent code and rewriting it to target the modeled hardware architecture. The transformation builds a minimal-overhead runtime layer to implement the modeled system, as seen in Figure.17; and translates the user-provided software to make use of this layer. If the hardware or allocations are changed in the model then the generated runtime layer is automatically reduced or expanded accordingly. This greatly

aids in Design Space Exploration (DSE) aspects. The details regarding these aspects have been detailed in [18].

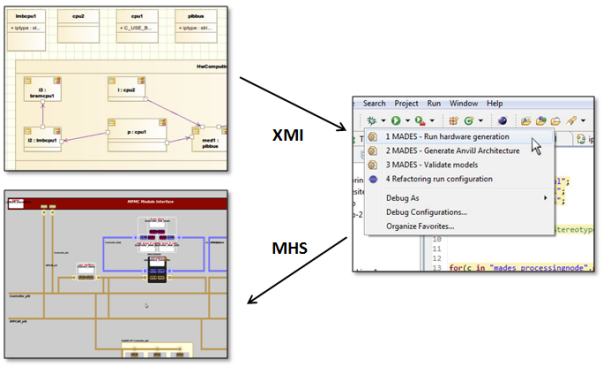

Figure.16: MADES based hardware generation

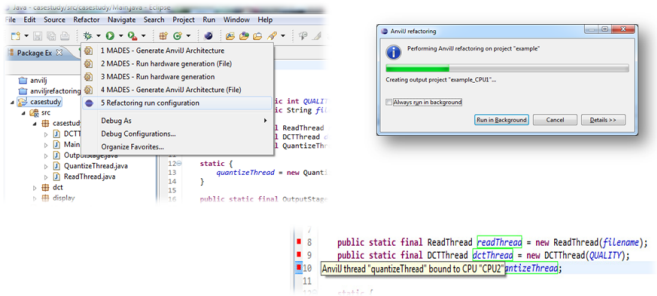

Figure.17: Generating software code using Anvil J and CTV

D. Synthesis and implementation in a Virtex V FPGA

Finally, the generated hardware/software are used to carry out synthesis, and then subsequent implementation is carried out on a Xilinx ML505 board containing a Virtex V series FPGA, as seen in Figure.18.

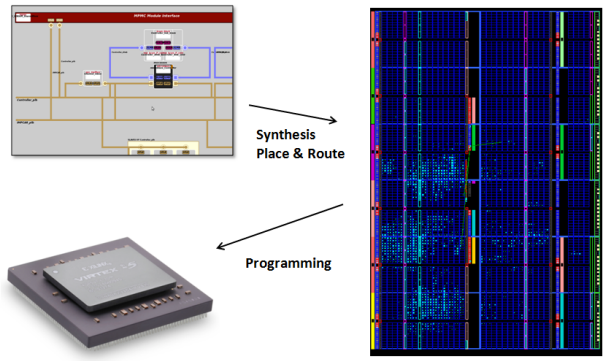

Figure.18: Synthesis and implementation on a Virtex V FPGA

IV. Conclusions

The paper describes the EU MADES FP7 project and describes its global methodology and the integrated tool set. The MADES language based on SysML/MARTE has also been detailed, followed by a Car Collision Avoidance System (CCAS) case study, where we describe the different phases of the methodology, namely: high level modeling, component based IP-reuse, verification and validation, hardware/software code generation, synthesis and final implementation on execution platforms, such as FPGAs.

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," 2010, http://www.omg.org/spec/SysML/1.2/.

[6] OMG, "Modeling and Analysis of Real-time and Embedded systems (MARTE)," 2011, http://www.omg.org/spec/MARTE/1.1/PDF.

[7] D.S. Kolovos et al, "Eclipse development tools for Epsilon," in *Eclipse Summit Europe, Eclipse Modeling Symposium*, 2006.

[8] N. Matragkas et al., "D4.1: Model Transformation and Code Generation Tools Specification," Tech. Rep., 2010, http://www.mades-project.org/.

[9] L. Baresi et al., "D3.1: Domain-specific and User-centred Verification," Tech. Rep., 2010, http://www.mades-project.org/.

[10] ——, "D3.3: Formal Dynamic Semantics of the Modelling Notation," Tech. Rep., 2010, http://www.mades-project.org/.

[11] Zot, "The Zot bounded model/satisfiability checker," 2012, http://zot.googlecode.com.

[12] 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.

[13] 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.

[14] Modelio, "Open source UML Editor and MDE Workbench," 2012, www.modelio.org.

[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.D. Gajski and R. Khun, "New VLSI Tools," *IEEE Computer*, vol. 16, pp. 11–14, 1983.

[17] Xilinx, "MicroBlaze Soft Processor Core," 2011, http://www.xilinx.com/tools/microblaze.htm.

[18] A. Bagnato et al, "D1.7: MADES Final Approach Guide," Tech. Rep., 2012, http://www.mades-project.org/.

[19] L. Baresi et al, "D3.2: Models and Methods for Systems Environment," Tech. Rep., 2012, http://www.mades-project.org/.

[20] OpenModelica, "Open-source Modelica-based modeling and simulation environment," 2012, http://www.openmodelica.org/.

[21] Ian Gray and Neil C. Audsley, "Developing Predictable Real-Time Embedded Systems Using AnvilJ," in *IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society, 2012, pp. 219–228.

[22] Modelica, "Modelica: An object-oriented equation based language ," 2012, https://modelica.org/.

[23] I. R. Quadri et al, "D1.6: MADES Tool Set - Final Version," Tech. Rep., 2012, http://www.mades-project.org/.

[24] I. R. Quadri, L. S. Indrusiak and A. Sadovykh, "MADES: A SysML/-MARTE high level methodology for real-time and embedded systems," in *International Conference on Embedded Real Time Software and Systems (ERTS2 2012)*, 2012.