International Journal of Computer Science and Applications,

Vol. 12, No. 1, pp. 117 – 126, 2015

TOWARDS AN UNIFIED APPROACH FOR MODELING AND ANALYSIS OF REAL-TIME EMBEDDED SYSTEMS USING MARTE/UML

Abdel mouttalib Maqil, Abdeslam En-Nouaary and Hicham Bensaid

Institut National des Postes et Télécommunications,
Rabat, MORROCO
{amaqil, abdeslam, bensaid}@inpt.ac.ma
www.inpt.ac.ma

Abstract—Real-time and embedded systems (RTES) have become a necessity in almost every aspect of our daily live. These systems are characterized by their complexity due to the integration of several hardware and software components, and their extreme need of reliability because they are usually used in safety and critical missions. These issues have been for a long time a major concern of researchers, practitioners and engineers, and have pushed them to develop rigorous methodologies and well-defined processes for the design and validation of RTES. This paper presents a unified approach to model and analyze real-time embedded systems using MARTE/UML profile to take care of both software and hardware aspects of the system being developed. The proposed approach follows the general guidelines and principles of platform-based design (PBD). An example is used to illustrate our contribution.

Keywords: RTES, UML, MARTE, PBD, V&V.

1. Introduction

Real-time and embedded systems (RTES) have become a necessity in almost every aspect of our daily live. They are present, for instance, in self-contained applications, in various devices and services such as mobile phones, medication dispensers and home appliances, as well as in heavy industry products like automotive, avionics and nuclear and chemical plants. A large proportion of these systems are mission/life critical and time sensitive and as such they must be developed rigorously to ensure their safety, their reliability and their fault tolerance. To this end, researchers, practitioners and engineers spend enormous efforts to come up with rigorous methodologies and well-defined processes for the design and validation of RTES [Lee and Seshia, 2010; Green and Edwards, 2002; Laplante and Ovaska, 2011]. Such design processes basically define three major steps: The modeling, the analysis and the hardware and software implementation. [Lee and Seshia, 2010] defines the modeling phase as “the process of gaining a deeper understanding of a system through imitation: Models imitates the system and reflects properties of the system” and the analysis phase as “the process of gaining a deeper understanding of a system through dissection: It specifies why a system does what it does (or fails to do what a model says it should do)”. It is therefore clear that the designer should start by modeling the system to produce the building artifacts, and then analyzes these artifacts to make sure that they are correct. In the case of RTES, a

well-defined process should absolutely identify the starting point of each of its activities and clearly define its interaction and dependency on others because the absence of those interactions in the process lead to more complexity than the design without following any process at all. This can't be achieved without using a unified modeling language such as UML. Indeed, several methodologies based on UML have been devised over the last decade in order to organize all the development required activities in well-defined processes.

HASoC [Green and Edwards, 2002], for instance, is a design methodology based on UML-RT notation. The design flow begins with a description of the system functionalities initially given in use case diagrams and then in a UML-RT version properly extended to include annotations with mapping information. The authors of this methodology argue that UML-RT is so restrictive to model all the aspects of Real-time systems [Chen et al., 2003].

Another well-known methodology is the Platform-based design (PBD) proposed by [Sangiovanni-Vincentelli and Martin, 2001; Sangiovanni-Vincentelli et al., 2004; Sangiovanni-Vincentelli and Di Natale, 2007]. This methodology tries to tackle the increased pressure on time-to-market, design and manufacturing costs by dividing the design process into two phases. In the first phase, a platform is designed where the hardware and software architectures for the system are clearly defined. In the second phase, the platform is specialized into a product through successive refinements. Because much of the initial design work was done in the first phase, the product can be developed relatively quickly based on the platform [Benini and De Micheli, 2005]. Generally speaking, the basic tenets of PBD are:

- The identification of the design as a meet-in-the-middle process, where successive refinements of specifications meet with abstractions of potential implementations, as shown in figure 1.
- The identification of precisely defined layers where the refinement and abstraction process takes place. Each layer supports a design stage that provides an opaque abstraction of lower layers that allows accurate performance estimations.

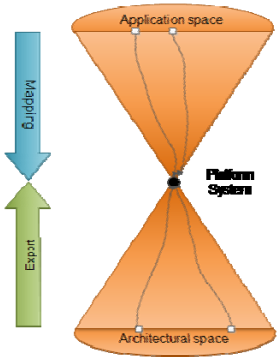

Fig. 1. Platform-based Design

The platform-based design is a powerful design methodology to construct embedded systems using the notion of platform, but it lacks the definition of steps to design each layer, and there is no adopted and defined language to use across the design steps.

Moreover, the behavioral and architectural analysis activities to verify and validate each layer are not addressed. Given these facts, one can easily deduce that there is an urgent need of a new approach to design real-time and embedded systems using the unified modeling language UML and associated profiles.

In this paper, we present a preliminary version of a unified approach to model and analyze real-time embedded systems using MARTE/UML profile to take care of both software and hardware aspects of the system being developed. The proposed approach follows the general guidelines and principles of platform-based design andmakes it possible to apply model driven architecture (MDA) for code generation.

The rest of the paper is organized as follows. Section 2 introduces the MARTE/UML profiles and explains its main components. Section 3 presents our methodology and describes how MARTE elements are used across its steps with simple illustration examples. Section 4 concludes the paper and presents future work.

2. MARTE/UML profile

MARTE (Modeling and Analysis of Real-Time Embedded systems) is a UML profile intended for model-based development of real-time embedded systems [Benini and De Micheli, 2005]. It provides support for specification, design, and verification/validation stages. This new profile is intended to replace the existing UML/SPT Profile (Schedulability, Performance and Time) [OMG, 2011]. MARTE is structured into packages, the three main of which are: MARTE Foundation which defines basic sub-profiles (Alloc, Time, NFP, GRM, etc.) which the two other packages rely on, the MARTE Design Model is used to model all software and hardware aspects, and finally MARTE Analysis Model which is intended for the analysis of schedulability and performance of the system being modeled. Figure 2 shows the overall structure of MARTE/UML profile.

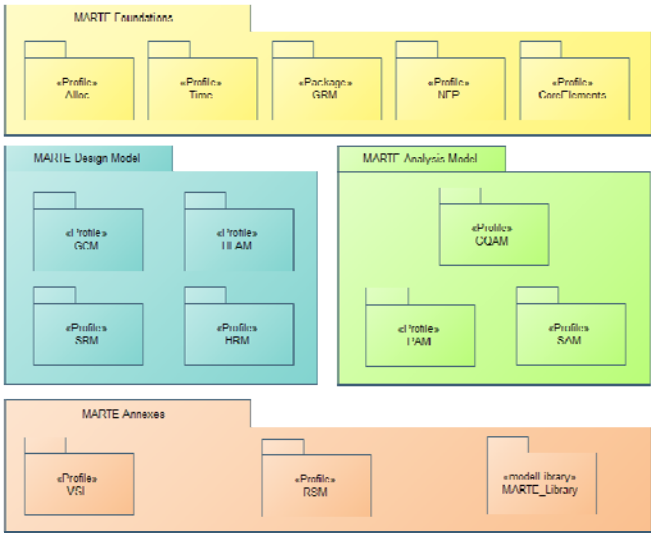

Fig. 2. MARTE/UML Structure

It should be noted that MARTE provides only the concepts and mechanisms required for modeling and analyzing RTES specific aspects but it does not define any methodology of how to do it. So, it is left open to the designer to select the appropriate stereotypes and the order in which they should be used for modeling the system under construction. This way of using MARTE might work for small-size systems that involve one single team for its development. However, this can cause inconsistencies and misinterpretations of building models when multi-parties are involved in the development process. To address this issue, we set up as objective of our research the definition of a unified approach for the design and analysis of RTES based on MARTE and PBD.

3. Our approach for designing and Analyzing RTES

Our approach is based on MARTE and PBD. It aims at bridging the gap between the programming languages and the modeling languages in order to shorten the time to market and reduce the development cost of RTES. This is possible through the intensive use of abstraction provided by PBD and MDA [Laplante and Ovaska, 2011]. As shown in figure 3, our approach defines 5 levels of abstraction for software design and 4 levels of abstraction for hardware design. The analysis, verification and validation of each level are executed after at least one level. All the artifacts are created using UML and MARTE profile. From the software side, we start the modeling with **operation modes** so that we can define all the possible sets of behaviors of the system under construction.

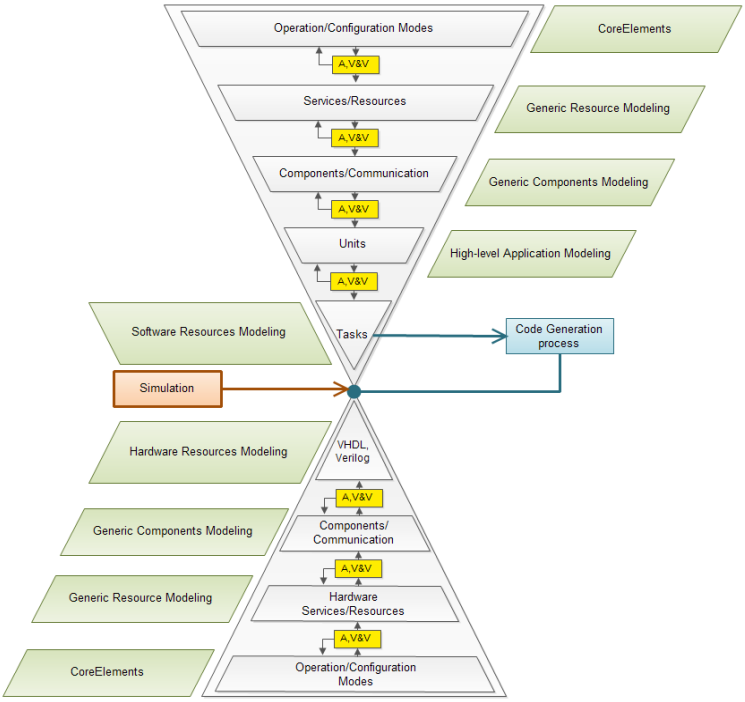

Fig. 3. Our design approach

Then, we perform the analysis of this layer using test cases which can be partially automated. After the validation is done we move to the next level of abstraction by defining all the contained services and their interaction. Afterwards, we perform the rest of refinements in order to define the components, the units and the tasks of the system, respectively. To clarify our methodology we propose simple examples which define all the abstraction levels for a battery management system in automobile. We limit the demo to the software side of the modeled system.

3.1. *Level I: Operation/Configuration Modes*

Most complex embedded systems have multiple modes of operation. Changes inmodes may be triggered externally by events such as user inputs or sensor data, or internally by interrupts, or hardware failures. The battery management system in new cars may have different behaviors when the car is in stop than when it is in start mode. The definition of all possible modes will give the designer the ability to handle all possible behaviors and thereafter, all the end-to-end scenarios. In our design process, each operation mode is then modeled as a set of service interactions. Figure 4 shows the configuration mode diagram of the battery management system.

The configuration diagram describes the first level of abstraction in our approach; it defines the application's operation modes and the possible transitions between them. The modes are defines using activity entities and the <<**mode**>> stereotype from MARTE profile. As for transitions, they are marked with the <<**modetransition**>> stereotype. The application starts all the time with the default mode named here **stopMode**. However, if there is a **startEngine** condition the application moves to the **startMode**. Finally, if there is a failure in the system, the application transits to low voltage mode and then triggers the services included in this mode.

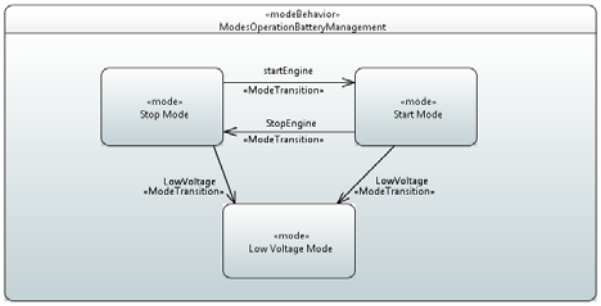

Fig.4. Example of Configuration Mode Diagram

3.2. *Level II: Services/Resources*

To interact efficiently with the environment, complex embedded systems are designed using a set of services. Such a structure makes the designed system much more reusable by providing build-time configurability [Douglass, 2002]. Figure 5 shows the service diagram of the battery management system.

The power management system contains three services, the communication service which uses the CAN protocol to send and receive data, the current/voltage measurement

service to acquire electrical data from the load, and the memory management service to write and read data for the diagnostic system. Analyzing this level will feed the designer with very important ratios like maintainability such as MTBF and MTTR. MTBF (Mean time between failures) is a ratio used to predict the elapsed time between failures of a system during operation, while MTTR (Meantime to repair) represents the average time required to repair the modeled system.

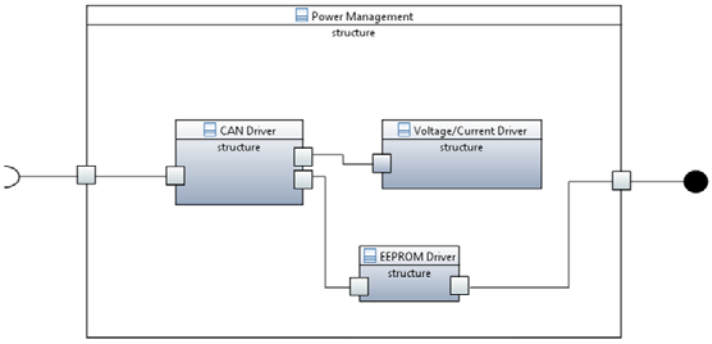

Fig.5. An Example of Service Diagram.

3.3. *Level III: Components/communication*

Services are constructed using one or more components; each component is connected with others using interfaces, and a pre-established communication protocol. In our approach, the level 3 defines the components allocated to each service. The example shown in figure 6 illustrates three components in the Voltage/Current service, the first one is used to sample data from the ADC (Analog to digital converter) then a filter component is used to filter and prepare data for storing or for other calculation.

The analysis of the behavior of each component focuses on the communication entities and their interactions in terms of synchronization, fault tolerance, response time, etc.

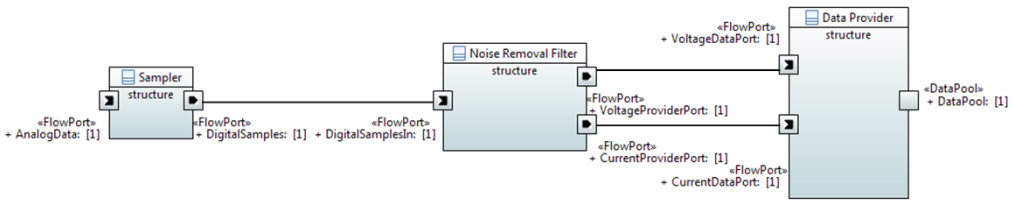

Fig.6. Example of Component Diagram

3.4. *Level IV: Units*

A unit or Real-time unit is the small entity which connects one or more tasks. It ensures the starting, the control of progression and the disposition of tasks if needed. In this level, we use the Time model of MARTE to define all Real-time characteristics, priorities and

scheduling policy. All the system real-time units and their interactions should be defined at this point. Analysis of this level is used to detect all the discrepancies with regard to schedulability and synchronization.

MARTE stereotypes used in this level are mainly <<**runit**>> and <<**punit**>>. <<**runit**>> is used to describe a real-time unit which contains at least one or more tasks, while the <<**punit**>> or passive unit is a unit with no scheduled tasks, this mechanism is used in MARTE to distinguish between the scheduled and unscheduled parts of the system's functionalities. Figure 7 shows the unit diagram for the battery management system.

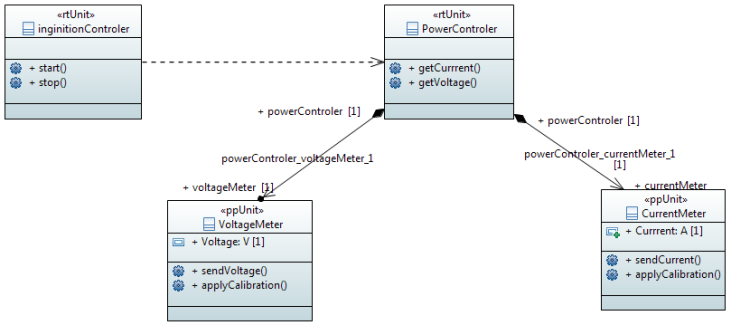

Fig.7. Example of Real-Time Unit Diagram

This example describes the contents of the sampler service in the service level. Two passive units feed the system with data when called by the real-time unit's functionalities in the power controller which is triggered by the ignition controller.

3.5. *Level V: Real-time Tasks*

Task level is the last abstraction level in our approach. At this level, the designer should describe all the tasks for each unit. To do so, the designer defines the behaviour of each task using a FSM (Finite State Machine) diagram. Tasks are characterized by different attributes such as the execution time, the response time and the variability of that response between the worst- and best-case scenarios (i.e., the jitter) and end-to-end scenarios simulation.

The real-time task defined in the example above is a safety task executed every time the ignition trigger is set. It checks the validity of the key being inserted, and then checks the power and safety rules of the system. Finally a start engine signal is sent to deliver the needed electrical power to the engine.

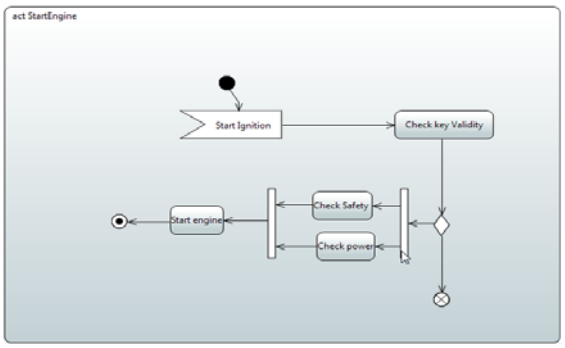

Fig.8. Example of Task Behavior

After finishing the task level, the analysis of the overall model is done by buildinga separated model annotated with elements from the SAM (Schedulability Analysis Modeling) or PAM (Performance Analysis Modeling) packages. The output parameters are checked and possible modifications could then be done on the annotated model. The system is declared valid if the resulting values are the same as what the designer expects. Figure 9 shows the overall analysis procedure.

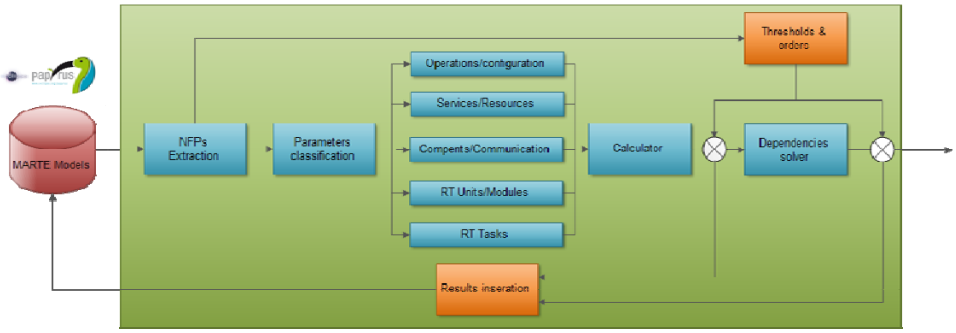

Fig.9. Analysis procedure based on PAM and SAM

The analysis procedure is based on the extraction of functional and non functional parameters to simulate and verify the reliability and the safety of the modeled system; the procedure is used repeatedly every time a level of abstraction is finished and also at the end of the modeling process to ensure that the final design of the overall system is correct.

As mentioned in figure 9, the analysis starts with the extraction of parameters from models annotated by MARTE profile, then an entity named a classifier in the Verification and Validation (V&V) engine is used to classify the parameters of each level. Finally, a calculation of results and a comparison with the defined instructions is done. By the end of the analysis procedure, the system is said to be correct if all the calculations are as estimated by the designer.

4. Conclusion

This paper presented a preliminary version of a unified approach for the design and validation of embedded and real-time systems. The approach uses MARTE/UML profile for modeling and validating the different artifacts produced during the development process. It also follows the general guidelines and principles of the platform-based design that helps the designer use abstractions and refinements in a structured and efficient way. Moreover, the proposed approach makes it possible to apply model driven architecture (MDA) for automatic code generation, which definitely reduces the overall cost of the final product and shorten its time-to-market.

We are currently working on the refinement of our approach in order to implement it in a framework built over Eclipse framework and MDT papyrus modeling tool, as shown in figure 10. The target framework will help the designer get global estimations of different performance ratios and software engineering metrics for the overall system. It will also allow the designer to perform the simulation of HW/SW interactions in addition to the automatic generation of stable and efficient C/C++ and Ada code.

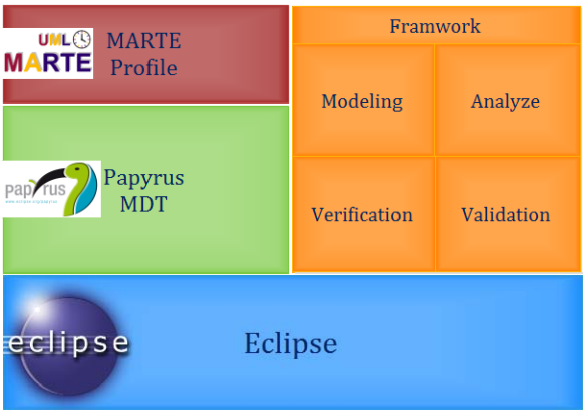

Fig.10. Overview of the Target Framework

References

Benini, L. and De Micheli, G. (2005). Chapter 3 - networks on chips: A new paradigm for component-based {MPSoC} design. In Jerraya, A. A. and Wayn, W., editors, Multiprocessor Systems-on-Chips, Systems on Silicon, pages 49 – 80. Morgan Kaufmann, San Francisco.

Chen, R., Sgroi, M., Lavagno, L., Martin, G., Sangiovanni-Vincentelli, A., and Rabaey, J. (2003). Chapter 4 - Uml and platform-based design. in Lavagno, L., Martin, G., Selic, B. V., editors, UML for Real Design of Embedded Real-Time Systems, pages 107-126. Springer.

Douglass, B. P. (2002). Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Green, P. N. and Edwards, M. D. (2002). The modelling of embedded systems using hasoc. In Proceedings of DATE 02, pages 752–759. IEEE Computer Society.

Laplante, P. A. and Ovaska, S. J. (2011). Real-Time Systems Design and Analysis: Tools for the Practitioner. Wiley-IEEE Press, 4th edition.

Lee, E. A. and Seshia, S. A. (2010). Introduction to Embedded Systems - A Cyber-Physical Systems Approach. Lee and Seshia, 1 edition.

OMG (2011). UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE), Version 1.1.

Sangiovanni-Vincentelli, A., Carloni, L., De Bernardinis, F., and Sgroi, M. (2004). Benefits and challenges for platform-based design. In Proceedings of the 41st Annual Design Automation Conference, DAC ’04, pages 409–414, New York, NY, USA. ACM.

Sangiovanni-Vincentelli, A. and Di Natale, M. (2007). Embedded system design for automotive applications. IEEE Computer, 40(10):42–51.

Sangiovanni-Vincentelli, A. and Martin, G. (2001). Platform-based design and software design methodology for embedded systems. IEEE Des. Test, 18(6):23–33.