UML-BASED SINGLE-SOURCE APPROACH FOR EVALUATION AND OPTIMIZATION OF MIXED-CRITICAL EMBEDDED SYSTEMS

P. Peñil, H. Posadas, J. Medina, E. Villar

University of Cantabria
ETSIIT, Santander, Spain
{pablop, posadash, villar@teisa.unican.es}, medinajl@unican.es

ABSTRACT

Mixed-critical systems combine highly-critical tasks with non-critical activities. Among them, there is a trend of sharing the available HW resources for all these activities, as a way of optimizing costs and power consumption. However, the design of such complex systems is a challenge, due to the different requirements and goals of critical and non-critical tasks, and the side effects resulting from their integration. As a consequence, the design of mixed critical systems requires the integration of different design flows and tools, and consideration of the whole system during the development of each specific component. To handle this challenge, this paper presents a single-source proposal where UML models are used as a common input that drives a flow of different design tools in a mixed-critical context. The flow covers performance estimation, static schedulability analysis and code synthesis to guarantee the critical tasks' constraints and to evaluate the remaining times available for the execution of non-real time tasks.

Keywords: Mixed-critical, UML, single source, analysis models, code synthesis

1. INTRODUCTION

Current embedded systems typically integrate multiple functionalities in a shared computing infrastructure. This integration results from the increase of computational power in the last HW platforms, and presents clear benefits in parameters, such as the cost of the product or the power consumption, compared with the use of separate computing elements. However, the integration of different functionalities working together increases the design complexity, due to the interactions resulting from the concurrent utilization of the different HW resources, such as processors, memories and peripherals.

This problem is especially important when the integrated functionalities have different criticalities. Mixed-critical systems require design flows capable of combining the strict design techniques needed to guarantee real-time requirements, with best-effort techniques for non-critical tasks. Additionally, these different requirements imply the use of several design steps and tools, complicating the design flow.

This work has been funded by the FP7 611146 CONTREX and the Spanish MINECO TEC2014-58036-C4-3-R projects.

For example, design of non-critical tasks can require the implementation of low-power optimizations to reduce consumption, and the integration of external SW packages that provide pre-designed functionality and complex communication features to reduce design time and effort. Additionally, the use of heterogeneous HW resources, such as co-processors, DSPs or GP-GPUs can also be required in order to minimize mean execution times in order to maximize system capabilities

On the other hand, design of real-time tasks involves ensuring several requirements, such as guaranteeing maximum response times and providing reliable, fault-tolerant implementations [1]. As a result, real-time design focuses on ensuring correct operation in worst-case scenarios instead of on mean values.

In order to guarantee correct response times, different static analyses and design-time decisions are required. Worst-case response time estimation, schedulability analysis and the assignment of optimized scheduling parameters are features required to guarantee time constraints. At the same time, solutions capable of guaranteeing system reliability are required, such as specific fault tolerant architectures, (e.g. Airbus A380 [2] and Hercules processors [3]), and application-level reliability-oriented techniques (e.g. duplicate with compare (DWC)[4], triple modular redundancy (TMR) and transparent re-execution [5]).

Furthermore, solutions for isolating critical and non-critical tasks are especially important to avoid non-critical tasks from interfering with high-critical task operation, since it can increase execution times, affecting real-time scheduling.

However, the exploration of all these design alternatives is not an easy task, especially in real, complex applications. The use of different tools to design all the mixed-critical functionality is typically a complex process. Furthermore, some of the tools typically used to analyze specific code sections cannot be oriented to support large systems.

Real-time design flows typically start by estimating worst-case execution times (WCET) with static timing analyses that depend on a precise behavioral model of the target hardware. However, developing and using static WCET analysis tools for evaluating complex modern systems is a difficult and expensive task. [6].

Considering all these details, it can be found that performing design flows that combine all these tools while

978-1-4673-7228-2/15/$31.00 ©2015 IEEE

including the resulting side effects in a manual way can be unsuitable. As an alternative, high-level solutions based on faster and more flexible approaches, including code synthesis and automatic tool integration are required.

For this purpose, this work proposes an alternative where, from a complete UML model of the system, an implemented infrastructure is able to generate all the tool-specific files required to estimate WCET considering different application concurrency, different resource allocation and multiple platform configurations, such as several processor frequencies. After that, the tool uses these estimations to call a static schedulability analysis tool, in order to evaluate whether real-time constraints can be fulfilled.

Finally, the infrastructure developed performs code synthesis to generate the deployment code required to run the system tasks in the real platform, obtaining real evaluations of the run-time system performance. As a result, it is possible to evaluate the feasibility of the final system and the slack times unused by highly critical tasks that can be used to run non-critical tasks.

2. STATE OF THE ART

Design of highly critical and multi-critical systems has been covered in multiple works, including different areas such as schedulability, reliability and run-time management.

Schedulability analysis typically covers the estimation of WCETs and their later analysis and implementation for ensuring the fulfillment of real-time constraints. Considering WCET estimation, specific analysis tools (e.g., [6]) can preliminarily characterize application tasks based on the final implementation (in hardware or in software). These tools are based on static analysis, and aim at providing the Worst Case Execution Time (WCET) of each task to ensure predictability and thus prevent timing violations, as required in real-time systems [8].

As an alternative, it is also possible to estimate the execution times of each piece of code from its execution under different execution scenarios, to pick the maximal observed execution times (MOET). This alternative is not strict, but it is done in industry in the case of low criticality systems [9], and can also be used for early, high-level exploration. Furthermore, it can also be used in more critical systems when no other alternatives are available.

Once the execution times are obtained, scheduling solutions are required to ensure that processor sharing will not affect the accomplishment of timing constraints. Considering these estimated times, analysis tools such as MAST [10], can analyze the resulting system to guarantee that the required constraints are fulfilled.

Existing task-scheduling approaches can be roughly classified into two branches, namely static (offline) and dynamic (online) approaches [11]. On the one hand, static scheduling approaches have been studied in past decades quite intensively. The usability of SAT solvers for scheduling synthesis of distributed systems has been presented in works such as [12]. In [13], tasks are scheduled by a preemptive, fixed priority algorithm, following deadline monotonic scheduling with preemption. In [22], a Tabu search-based design optimization is applied.

On the other hand, dynamic (online) scheduling strategies have also been investigated. For example, a run-time spatial mapping and demonstration of streaming applications on heterogeneous MPSoCs at runtime is presented in [14].

Additionally, several works propose analysis and synthesis processes oriented to combining reliability with real-time constraints. COFTA [4] is a co-synthesis tool that introduces fault detection capabilities for reliable embedded systems. It adds software assertions and duplicate-with-compare (DWC) tasks to ensure critical operation.

Finally, different works also propose solutions to isolate critical and non-critical tasks in order to minimize the side effects resulting from their interactions. This alternative helps in certification processes, and allows non-critical tasks to run in parallel with the critical task [15].

However, all these approaches rely on restricted input models [13], which limits the semantics of the task interconnections and complicates the reuse of system models among tools.

To solve this problem, the approach presented in this paper proposes starting the design process with a generic model based on UML. Then, it will be easier to integrate tools developed for design of generic embedded systems together with the improvements performed for mixed-critical systems. For this purpose, model transformations and automatic code synthesis are used.

Automatic code synthesis from high-level models has received significant interest in the last decade. In this context, several research works on synthesis from UML models have been presented. For example, synthesis of code from UML activity diagrams is presented in [16]. In [17] a semi-automatic method for generation of HW/SW infrastructures from UML models using Remote Method Invocation (RMI) semantics is presented.

The works focused on UML profiles for embedded, real-time systems are also especially important. Among them, the MARTE profile was created and standardized by the OMG [18]. Taking MARTE-based models as input, several synthesis approaches have been proposed. Gaspard2 [19] is a design environment for data-intensive applications which enables MARTE description of both the application and the hardware platform. Finally, in [20] and [21], automatic synthesis is applied to integrate user-provided functionality and communication semantics. In these, automatic code generation is applied to easily explore different concurrent architectures, simplifying the evaluation of different parallelization and mapping alternatives.

Thus, the proposal in this work is to apply UML/MARTE, to generate single source models that can integrate all the information required to automatically call other design tools needed for the development of mixed-critical embedded systems.

3. INTEGRATED TOOL FLOW

The work described in this paper presents a design flow and an implemented infrastructure that, from a single, reusable UML model, are capable of automatically calling a set of tools oriented to obtaining early WCET estimations, checking real-time schedulability and performing automatic code generation for real implementation. The goal is to ensure real-time constraints for critical tasks and evaluate the unused processor times that can be used for other non-critical activities in a simple way.

As stated before, developing and using static WCET analysis tools for evaluating complex modern systems is a difficult and expensive task, due to the complexity of the HW platforms and to the difficulty of performing static analysis to evaluate real, large application codes. Thus MOET estimations are obtained instead. Nevertheless, integration with static WCET analysis is also possible, and is currently under development. To do so, the UML model includes information to call PAREON simulation obtaining MOET times for different system configurations available for the application mapping and processor execution modes. Then, the infrastructure uses this information to call the MAST tool to perform the schedulability analysis of the system.

Later on, the eSSYN tool can be called for the automatic generation of the deployment codes required to execute the system in a real target platform. That tool enables on-board performance measurements to be obtained. As a result, the different design alternatives for the system can be analyzed and explored, in a fast, low-effort way, providing all the information required to adequately start the following refinement steps.

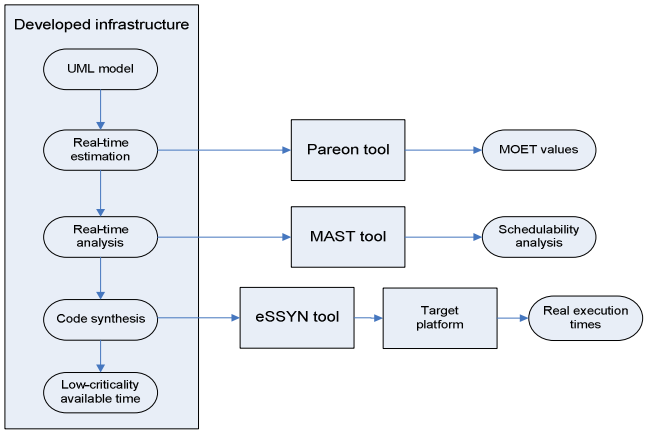

Figure 1 Developed integrated flow

4. UML/MARTE METHODOLOGY

The UML/MARTE methodology applied is characterized by following a component-oriented approach and applying Model-Driven Architecture (MDA) principles in the development of the HW/SW embedded systems. System specification combines the definition of platform-independent details, forming the Platform-Independent model (PIM), with platform description and implementation mapping details.

The system functionality is defined as a set of application components interconnected by channels, and communicated by using service calls encapsulated in interfaces. The channels have semantics associated with the service calls, with several blocking alternatives, different priorities, time outs and buffering capacity. This model is completed with the C code files providing the functionality of the application components. Then, the application components are allocated into memory spaces.

Afterwards, the model is completed describing the HW/SW platform through the Platform Description Model (PDM). This model includes the definition of the SW and HW resources that composed the HW/SW platform and their interconnections. Considering SW resources, it is possible to describe the available operating systems, the scheduling policy and the APIs that can be used to implement communication and concurrency during the synthesis step.

At the same time, the list of HW resources includes the modeling of different types of processing elements such as GPP, DSPs and GPUs. In addition, the methodology enables the capture of buses, memories, caches, peripherals and all the relevant attributes that characterize them.

Finally, in the Platform Specific Model (PSM), the memory spaces are allocated on the processing resources included in the HW/SW platform.

All the previous modeling aspects constitute the essence of the UML/MARTE methodology that was previously presented in [21]. Moreover, in order to enable the schedulability analysis and the SW synthesis, additional modeling aspects have to be included.

5. TOOL INTEGRATION

5.1 WCET estimation

The first step in the integrated design flow is the estimation of the WCETs of the different services in order to verify that the real-time requirements of the critical tasks can be fulfilled. Since strict WCETs are hard to evaluate, the execution of multiple tests obtaining the maximum observable execution times (MOET) has been considered as a more feasible way of getting representative numbers for any kind of application code.

To handle this, the PAREON simulation tool has been used. To do so, the developed infrastructure starts generating the executable codes necessary to launch PAREON, combining the functional codes provided by the designers with the deployment and connection codes automatically synthesized from the UML model information. Then, the infrastructure calls the PAREON simulator for each processor frequency, obtaining reports that enable the identification of the operation conditions that will guarantee the real-time requirements.

In order to capture the different conditions under which processors can work, their operation modes are described with UML state machines associated with the processors. In these state machines the operation modes are represented as UML states specified by the MARTE stereotype <<Mode>>. Moreover, transitions are represented as "UML transitions", specified by the MARTE stereotype <<ModeTransition>>. Each operational mode is characterized by a frequency.

Finally, from the executions of the PAREON tool, a segment costs' table can be created by analyzing the execution times of all the segments in their execution and extracting the MOET information.

5.2 Schedulability Analysis

For enabling automatic performance of schedulability analysis, the UML/MARTE methodology has been enriched with new modelling features. First of all, the high-critical functionality of the application components is described covering both the services provided by each component and other internal functions that have potential impact for the schedulability analysis. Then, functions are characterized by different timing properties in order to create analysis scenarios to be checked by the analysis tool. Non-critical tasks are ignored for this analysis.

These functions are modeled as UML operations owned by the application component. Then, these functions are specified by the MARTE stereotype <<SaStep>>. The attribute *execTime* defines the worst and the best execution time of the function (Figure 2).

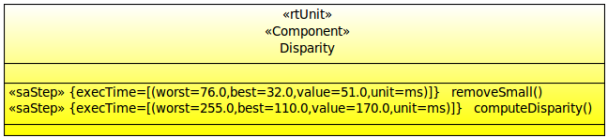

Figure 2 Application component and function definition

For the schedulability analysis, the designer also has to explicitly model the number of threads that execute the application component. These concurrent elements are specified by the MARTE stereotype <<SchedulableResource>>. Then, each of these tasks is associated with a processing resource included in the HW/SW platform in the attribute *host*.

Next, the details of the HW platform must be considered to perform the analysis. To do so, each *HwProcessor* is specified by the MARTE stereotype <<SaExecHost>>. This MARTE stereotype enables properties such as the range of priorities and context switching times to be captured.

Then, the platform description is composed of interconnected instances of these HW resources (Figure 3).

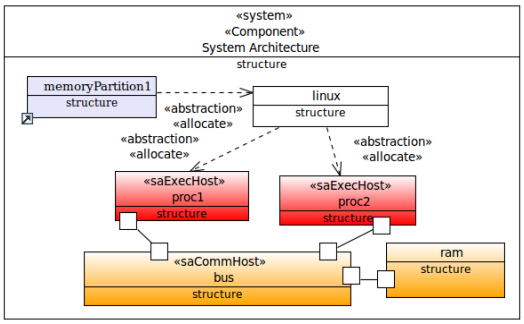

Figure 3: HW/SW platform

Furthermore, it is also necessary to capture working scenarios. The scenarios are real-time situations defined by a sequence of functions which are executed by tasks. These scenarios are modeled with the MARTE stereotype <<SaEndtoEndFlow>> and described in Activity Diagrams (Figure 4). Each *SaEndtoEndFlow* has an initial node. This element is specified by the MARTE stereotype <<GaWorkLoadEvent>> to denote the characteristics of events that trigger the scenario (periodic, sporadic…). Then, the atomic actions that make up the complete activity are modeled as UML Opaque actions specified by the MARTE stereotype <<SaStep>>. Each *SaStep* captures an ordered sequence of function calls, which are executed by a *SchedulableResource*.

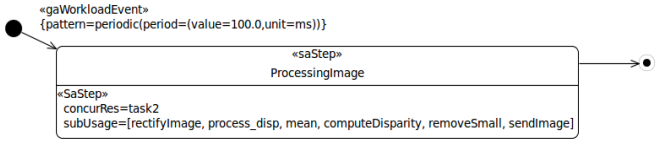

Figure 4 Specification of analysis context

As a result, all these details are automatically transformed into an analysis model that the MAST tool can evaluate, generating schedulability results. Thus, it is possible to identify which system configurations are feasible for accomplishing real-time requirements.

5.3 SW Synthesis

Once the feasible configurations have been identified, the next step is to find out the amount of time that highly critical tasks do not use under normal operation, to enable its use by non-critical tasks. For this purpose highly critical tasks can be run, measuring their slack times, which are the times available for the rest of the system.

To do so, automatic code generation is performed to generate the deployment code that enables running the application on the real platform for each possible configuration. In this process, information about components, channels and mapping is critical.

Specific features specified by the communication channels for characterizing the communication such as blockings, storing capacity or timeouts, are considered during the synthesis process. In addition, the channels have associated information about how to implement them, such

as the API that should be used for implementation (e.g. MCAPI, OpenMP, TCP/IP, …).

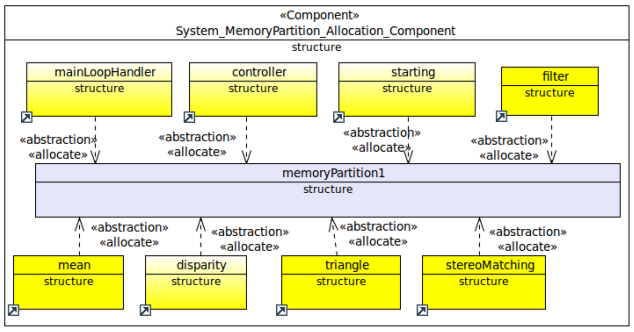

Figure 5: memory space allocation

Then, application components are associated with different memory spaces, ensuring that global variables and shared memory areas can only be used by components in the same memory spaces (Figure 5).

In order to enable a well-defined "makefile" generation, the HW Processors have the corresponding compilers and compilation flags associated. This information is annotated in an UML constraint specified by the model variables $cc_compiler, $cxx_compiler, $cflags and $lflags.

As a result, the tool can generate the deployment codes that execute and communicate the application components in the different HW resources of the platform, enabling the evaluation of their real operation.

6. USE CASE AND RESULTS

In order to demonstrate the flow and toolkit presented in this paper, a stereoscopic vision system was selected. In this system, the images received from two cameras are initially rectified to enable their later comparison in order to extract the position of the different objects in the image with respect to the observer.

The processing application is divided into two main phases. In the first phase, the application is connected to the system environment to acquire images and store the processed images. The "controller" calls the camera to acquire the new images and sends them to the next application component. The "mainLoopHandler" component collects the image acquired by the controller and sends it to the second phase, the stereo matching process. Then, after the image processing, this component sends the processed images to store them.

In the second phase, composed of the components "stereoMatching", "triangle", "mean", "filter" and "disparity", the images are processed by applying the stereo-matching algorithm. Following this division, application components are modeled as Figure 2 shows.

Then, the structure of the application is composed of interconnected instances of the components, as indicated in Figure 6. The system concurrency is defined according to the application phases described above. Four tasks are considered: one task ("Task 1") acquires the images (first phase), while another three tasks ("task 2, 3 and 4") analyze three pairs of images in parallel.

Regarding the HW platform, an Intel Duo-based board, (Figure 3) has been considered. Tasks "task1" and "task2" are executed in processor 1 and the others in "proc2".

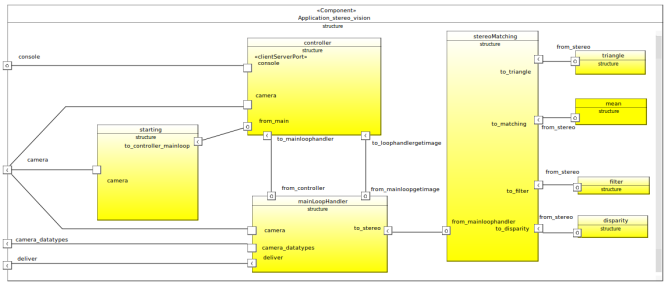

Figure 6 Stereovision system

The goal of the use case is to guarantee a certain frame rate, analyzing the available time that could be used for other non-critical activities that the system can perform. In this context, the first step is to estimate execution times and perform a schedulability analysis. Then, automatic synthesis is applied to demonstrate the validity of the static analysis and to identify the slack time that is available for other non-critical tasks.

For that static analysis, first each function has been characterized in terms of execution time, as Figure 2 shows. Then, an analysis scenario has been modeled as Figure 4 shows. This analysis scenario is characterized by a workload that defines the event which triggers the behavior. In this case, the event is an image acquisition. This event is defined as periodic. The period is the frame rate that is considered. Then, the image is processed. The image processing is a sequence of functions. The functions executed are shown in the attribute *subUsage* in Figure 4.

Function	MOET 3.8GHZ	MOET 2.4GHZ	MOET 1.6GHz
main_app	20	35	50
rectifyImage	11000	18000	28000
Process_disp	55000	88000	130000
computeDisparity	110000	170000	255000
removeSmall	28000	42000	62000
mean	31000	49000	74000
sendImage	500	800	1000

Table 1: Functions and execution times in us

The functions considered in the analysis are shown in Table 1, where the name and the individual MOET are annotated depending on the operational mode of the processors, defining the best, the worst and the average execution times.

Next, the schedulability analysis is performed with MAST, considering some additional rules. First, a security margin of processor utilization has been established: the utilization of the processor should be less than 90%. Secondly, 5% of additional processor utilization is considered as a consequence of OS operation.

The results obtained from the schedulability analysis are shown in Table 2, capturing the utilization of each processor for each fps. Due to the CPU utilization constraint, the cases "fps = 12" and "fps=16" are considered as invalid.

Processor	Utilization (8fps)	Utilization (10fps)	Utilization (12fps)	Utilization (16fps)
proc1	33.4 %	41.7 %	49.9 %	66.3%
proc2	66.7 %	81.4 %	99.3 %	133.5%

Table 2: Platform processor utilization

Next, the SW synthesis process is applied. Functional codes (specified as files in folders, specific source-code files or pre-compiled files) are compiled and linked together with the generated code described above, applying generic codes or optimized codes when available. Thus, all the feasible solutions detected by the static analysis can be easily run on the real platform to obtain real, mean execution values. The experimental results are shown in Table 3, indicating the CPU utilization and whether there are lost frames. Thus, it is possible to verify the results of the static analysis and know the time available for running other non-critical tasks that we could add to the system.

	8fps	16fps	24fps
CPU mean utilization	28%	61.6%	97.2%
Frame loss	No	No	Yes

Table 3: Execution in board

In this experiment the goal is that all images acquired should be processed. According to this goal, the case "fps=24" is clearly discarded, since frames are lost. This is consistent with results obtained from the static analysis.

Moreover, it can be shown that although the static analysis established fps=10 as a limit, the implementation results show that higher fps can be considered, enabling the use of available free time for non-critical tasks. In this case, no frames were lost at 16 fps, meaning that for the selected images computation is possible, but not secure.

7. CONCLUSIONS

This work presents an infrastructure capable of integrating several of the design tools required for the development of mixed-critical systems from a single, high-level input model. Estimation of maximum execution times, schedulability analysis and code synthesis for real platform execution can be automatically performed from the high-level model in order to evaluate the expected behaviour of the system under different design choices.

To do so, an initial UML model is used to specify all the information required to automatically call the auxiliary tools required for the design process. With this information, the developed infrastructure first generates the files required to perform different simulations with the PAREON tool in order to obtain the maximum observable execution times for the selected platform under different application mappings and processor frequencies.

Then, analysis models are generated in order to perform schedulability analysis with the MAST tool. Finally, the eSSYN tool is called to generate the executables required to run the selected configuration in the real platforms, obtaining real values of the system execution.

Thus, it is possible to select the adequate configuration of the target platform, ensuring the fulfilment of the real-time constraints of the critical tasks, and maintaining enough available time for the execution of the non-critical activities.

8. REFERENCES

[1] Axer, P. ; Sebastian, M. ; Ernst, R., "Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints". CODES+ISSS, 2011

[2] Advanced Control Systems for Airbus A380, http:// www.tttech.com

[3] Texas Instruments, "Overview of Hercules ARM Safety MCUs," http://www.ti.com/mcu/docs/mcuprodoverview.tsp? sectionId=95&tabId=2835&familyId=1931, 2013

[4] B. Dave and N. Jha, "COFTA: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems for Low Overhead Fault Tolerance," IEEE Trans. Computers, vol. 48, Apr. 1999

[5] V. Izosimov, I. Polian, P. Pop, P. Eles, and Z. Peng, "Analysis and Optimization of Fault-Tolerant Embedded Systems with Hardened Processors,"Conf. Design Automation and Test in Europe, 2009.

[6] Michael Zolda, "Precise Measurement-Based Worst-Case Execution Time Estimation", http://researchprofiles.herts.ac.uk/portal/files/1344125/phd_thesis.pdf

[7] M. Lattuada and F. Ferrandi, "Performance Modeling of Embedded Applications with Zero Architectural Knowledge", Conf. Hardware/Software Codesign and System Synthesis, pp. 277-286, 2010

[8] Bolchini, C. ; Miele, A. "Reliability-Driven System-Level Synthesis for Mixed-CriticalEmbedded Systems". Transactions on Computers, 2013

[9] Reinhard Wilhelm et al. "The worst-case executiontime problem - overview of methods and survey of tools". ACM TECS, April 2008.

[10] MAST, www.mast.unican.es.

[11] Bagheri, M. ; Jervan, G.. "Fault-Tolerant Scheduling of Mixed-Critical Applications on Multi-processor Platforms". EUC, 2014.

[12] A. Metzner, M. Franzle, C. Herde, and I. Stierand, "Scheduling dis- ¨ tributed real-time systems by satisfiability checking," in RTCSA '05.

[13] Voss,S. ; Schatz, . "Deployment and Scheduling Synthesis for Mixed-Critical Shared-Memory Applications". ECBS, 2013.

[14] P. K. F. Holzenspies, J. L. Hurink, J. Kuper, and G. J. M. Smit, "Run- ¨ time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (mpsoc)," in. DATE '08.

[15] Kritikakou, A. ; Pagetti, C. ; Baldellon, O. ; Roy, M. ; Rochange, C., "Run-Time Control to Increase Task Parallelism In Mixed-Critical Systems". ECRTS, 2014

[16] S. Kang, H. Kim, J. Baik, H. Choi, C. Keum."Transformation Rules for Synthesis of UML Activity Diagram from Scenario-Based Specification". Proc of COMPSAC, 2010.

[17] J. Barba, et all, "Automatic HW/SW Interface Generation for Seamless Integration from Object-Oriented Models", Proc of ESA, 2009.

[18] OMG, "UML Profile for MARTE", www.omgmarte.org, 2013.

[19] É. Piel et al: "Gaspard2: from MARTE to SystemC Simulation", DATE workshop on Modeling and Analysis of Real-Time and Embedded Systems with MARTE, 2008.

[20] P. Peñil, H. Posadas, A. Nicolás, E. Villar, "Automatic synthesis from UML/MARTE models using channel semantics", ACES-MB, 2012

[21] H. Posadas, P. Peñil, A. Nicolás, E.Villar, "Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation". Microelectronics Journal. Volume 45, Issue 10, October 2014.

[22] D. Tamas-Selicean, P. Pop, "Design Optimization of Mixed-Criticality Real-Time Embedded Systems", ACM Transactions on Embedded Computing Systems, May 2015