Integrated Timing Analysis in the Model-Driven Design of Automotive Systems[*]

Dulcineia Oliveira da Penha, and Gereon Weiss

Fraunhofer ESK, Automotive, Hansastrasse 32 80686, Munich, Germany
{dulcineia.penha, gereon.weiss}@esk.fraunhofer.de

Abstract. Automotive electronic systems integrate steadily increasing number of functions. Model-driven development of such systems enables to handle their complexity. With the integration of software-components and their intricate interactions ensuring the non-functional behavior, like timing, becomes a crucial matter. Timing analysis allows the validation of these properties but is mostly only loosely integrated with the development process. Therefore, we introduce an integrated approach enabling the iterative timing validation of model-driven designs. It consists of a framework comprising an UML modeling tool and a simulation-based timing analysis tool. By integrating the design models with respective analysis models, the development of timing-accurate designs is enabled. With the example of an automotive infotainment case study we show the applicability of our approach.

Keywords: Timing-Behavior, Model-Driven, Automotive

1 Introduction

Early validation has become an important task for developing modern networked automotive software. It allows detection and correction of flaws still during design, avoiding problems at later stages of development and during integration. Since automotive applications are often safety-relevant, it is not enough to solely validate the system's functional behavior. In addition, the validation of non-functional properties, such as timing is required in order to ensure the full system safety and correctness, to improve system's efficiency and to save resources. Several automotive applications are subject to hard real-time constraints intended to ensure full system safety and correctness [1] [2]. Furthermore, the presence of firm and soft real-time constraints in those systems is constantly increasing and ensures correctness and quality, improves the user experience and in some cases guarantees safety. Considering the increasing

[*] *The project was funded by the Bavarian Ministry of Economic Affairs, Infrastructure, Transport and Technology.*

number of software functions and the complexity imposed by real-time constraints, the integration of software components within automotive systems becomes a challenging task. Therefore, the system's timing behavior should already be considered and evaluated in early phases of the development process in order to guarantee the fulfillment of timing requirements in later stages and after deployment to the target platform. [2]. Model-driven approaches play a particular role in the design and early validation of embedded software systems and are widely used in the automotive industry [3]. Model-driven design is supported by various standards and tools.

Nevertheless, in most cases, the solutions provided for the validation of timing behavior by such model-driven approaches and tools are precarious and must be manually realized, demanding extra effort from engineers. For example, developers need to define scheduling tables manually and iterate until certain timing constraints are met, which could cost precious development time. On the other hand, several approaches and tools specialize on a detailed analysis of the system's timing behavior based on a particular model of the system and its target platform. However, such models are usually specific for each timing analysis approach demanding the design of a second model of the system for that purpose only. The fact that the design and development methods and the approaches for timing analysis are currently realized independently demands the development of different and separated models, requiring extra effort and time. In order to improve the development of automotive systems by reducing the effort necessary for timing validation, we present an approach for the integration of model-driven design and timing analysis of timing-dependent automotive applications. It allows the usage of a single model for the design and validation of functional and timing behavior in an automated and optimized process.

The paper is structured as follows: Section 2 discusses the Context and Challenges for Real-Time Automotive Applications. Section 3 presents our approach for integrating timing analysis in the model-driven design of automotive systems. In Section 4 we introduce the infotainment case study, evaluate our approach and present the results. The related works are discussed in Section 5 and we conclude in Section 6.

2 Real-time constraints in the automotive domain

Many functions in the automotive domain involve real-time constraints and can benefit from early timing analysis. These functions are characterized by hard, firm and soft real-time constraints. In a hard real-time function no deadline can be missed. In firm and soft real-time occasional deadlines can be missed, however it compromises the system quality and thus degrades the user experience. Engine control and X-by-wire are examples of hard real-time applications, while the so-called infotainment systems contain several firm and soft real-time applications. As previously discussed, the increasing number of real-time software functions in nowadays automotive systems of networked applications represents a challenge for their integration. This increases the importance of validating not only hard real-time constraints, but also the firm and soft ones, at early stages of the development process. In this paper, we focus primarily on automotive software with firm and soft real-time constraints.

2.1 Real-time automotive applications

In today’s cars, many applications which share resources and have priorities according to real-time constraints. Applications such as TA (Traffic Announcement), Navigation System, Telephone Integration, Parking Assistance System among others share resources on the Infotainment System, for example, the audio system. Considering for instance that at a given moment the driver is listening to music and the navigation system is activated. The audio system is being shared by both applications at the same time. The music is interrupted or its volume lowered when the navigation system generates a stimulus to give instructions. The response to this stimulus must happen in a couple of hundred milliseconds. Otherwise the navigation information will be useless or distract the driver, degrading the user experience of that application and possibly imposing safety risks. If real-time constraints are not fulfilled, the system’s requirements are compromised. There is an increasing number of others applications with soft and hard real-time constraints which share resources such as the audible warning mechanism and require the definition of priorities and real/time constrains. For instance, analysis of driver’s condition (e.g. driver drowsiness), speed limit and icy road warning, cruise control (warning for its deactivation by the vehicle), lane departure system (warning if the vehicle begins to move out of its lane).

This vast number of applications makes the system’s integration and planning of timing and resources more complex. Thus, today’s development approaches are in urgent need for the support of timing validation. In this scenario, the early validation of real-time constraints in automotive applications contributes to a flawless behavior in later stages of the development process and after the deployment to the target platform. However, since today’s priorities and interaction of integrated functions is done manually, this is cost-intensive with respect to development and validation time and it does not scale with the increasing number of applications in newer vehicles.

3 Validating timing behavior in the design process

For improving the development process of automotive systems we present an approach which integrates the early and iterative validation of timing behavior in the model-driven development of automotive systems. Based on this approach, we developed a framework which integrates a modeling tool and its execution framework widely-used in the automotive domain with timing analysis. The developed approach and framework allow engineers to model automotive-conformant systems, to perform a detailed timing analysis using proper tools and by validating the system’s timing behavior based on a single design model. The developed approach starts with modeling the automotive system in UML as shown in Fig. 1. This design model represents the system architecture and behavior. The system’s architecture is conformant to the automotive standard AUTOSAR [4]. The behavior is modeled using State Machine Diagrams. The target platform and allocation of software functions to the hardware are also modeled. The developed design model can be the basis for testing and validation of the system’s functional behavior using regular approaches for tests. In parallel, our approach automatically generates an analysis model out of the design model. It is extended with the necessary timing information by the engineer.

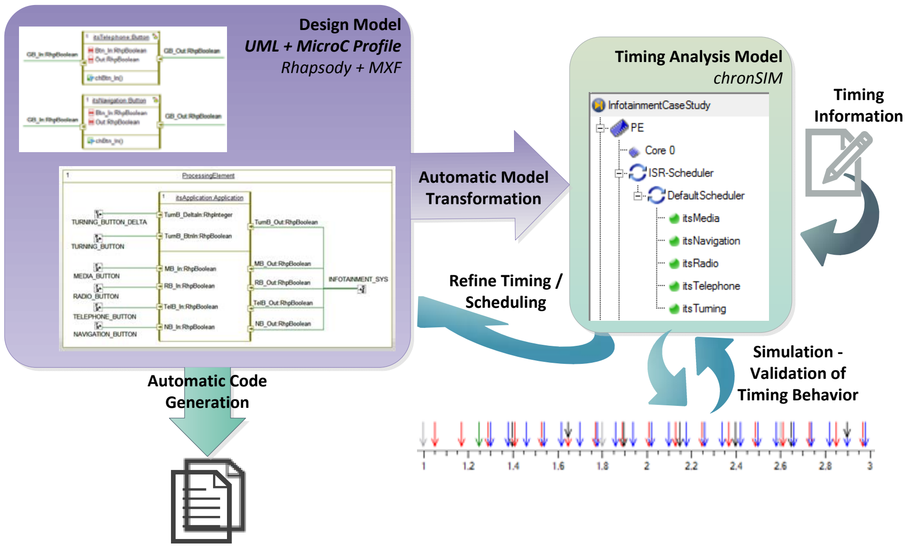


Fig. 1. Validation of timing behavior in the design process

The analysis model is the basis for the timing analysis, which is realized by proper tools. It is based on the generated model and timing constraints while taking into account the characteristics of the target platform. Based on the results of the analysis, the timing behavior of the system can be analyzed and iteratively improved. This eliminates possible conflicts, e.g. scheduling problems. The process for designing and validating timing behavior is automated and based on a single model. Through this the effort traditionally necessary for creating a separated model for timing analysis can be saved.

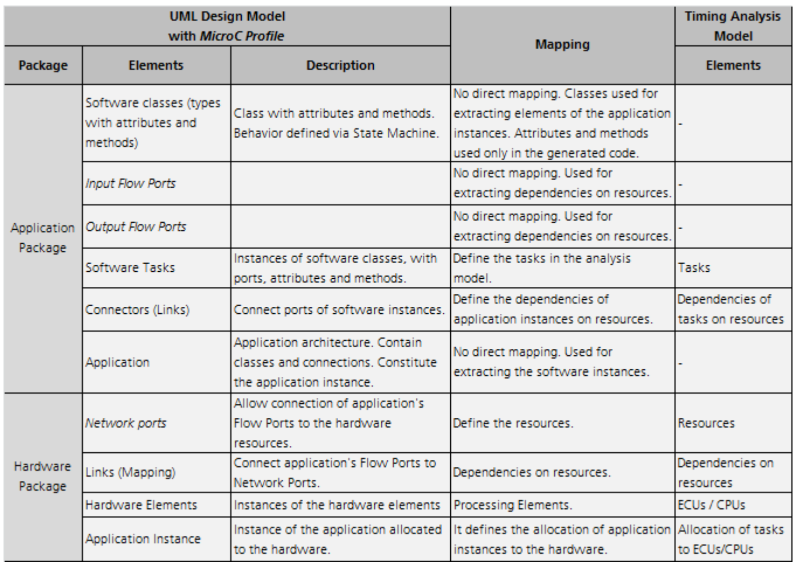
3.1 Timing Analysis

For a definition of timing analysis we refer to Wilhelm et. al. in [5], where they state that “timing analysis attempts to determine bounds on the execution times of a task when executed on a particular hardware”. There are two classes of methods for timing analysis, static- and measurement-based methods. Timing analysis via measurement-based methods is obtained by the execution of tasks (or task parts) on a given target platform or through simulation of some set of inputs. On the other hand, static methods analyze the set of possible control-flow paths through the task. The control flow is analyzed based on a model of the hardware architecture in order to obtain upper bounds for this combination. [5] Static methods and simulations are both proven suitable for early validation of timing behavior. In this paper we focus on simulation, but our methodology may also be integrated with static methods.

3.2 Semantic Mapping of Model Elements

In order to automatically generate a timing analysis model out of the design model, a model transformation is realized. The elements and concepts of the design model which are relevant for the timing analysis are mapped to respective elements in the timing analysis model, as presented in Table 1.

Table 1. Model Transformation: Mapping from UML Design to Analysis Model



UML Design Model with *MicroC Profile*			Mapping	Timing Analysis Model
Package	**Elements**	**Description**		**Elements**
Application Package	Software classes (types with attributes and methods)	Class with attributes and methods. Behavior defined via State Machine.	No direct mapping. Classes used for extracting elements of the application instances. Attributes and methods used only in the generated code.	-
	Input Flow Ports		No direct mapping. Used for extracting dependencies on resources.	-
	Output Flow Ports		No direct mapping. Used for extracting dependencies on resources.	-
	Software Tasks	Instances of software classes, with ports, attributes and methods.	Define the tasks in the analysis model.	Tasks
	Connectors (Links)	Connect ports of software instances.	Define the dependencies of application instances on resources.	Dependencies of tasks on resources
	Application	Application architecture. Contain classes and connections. Constitute the application instance.	No direct mapping. Used for extracting the software instances.	-
Hardware Package	*Network ports*	Allow connection of application's Flow Ports to the hardware resources.	Define the resources.	Resources
	Links (Mapping)	Connect application's Flow Ports to Network Ports.	Dependencies on resources.	Dependencies on resources
	Hardware Elements	Instances of the hardware elements	Processing Elements.	ECUs / CPUs
	Application Instance	Instance of the application allocated to the hardware.	It defines the allocation of application instances to the hardware.	Allocation of tasks to ECUs/CPUs

3.3 Framework Realization: Implementation of Tool-Chain

The developed framework supports the model-driven design of automotive systems using the UML-Tool IBM Rational Rhapsody [6] with the *MicroC Execution Framework (MXF)* [7]. MXF is a combination of a UML Profile, code-generator and run-time framework which allows the simulation and target execution of the system's design model. It allows the design of hardware-independent software by employing the so-called *Network Ports* which connect the model components to external hardware modules such as I/O ports, network devices, sensors or buses. Thus, the software model can be abstractly developed and later connected to specific hardware. The design model is composed by the application model and the hardware model. The system behavior is modeled using UML State Machine Diagrams. For the run-time framework the definition of a scheduling table is needed. Originally this scheduling table is created by the engineer based on rough constraints and his experience only. With our approach, the creation of a scheduling is optimized by the provision of plentiful information about the timing analysis to the engineer. Based on the detailed results of the timing analysis, the scheduling table can be more easily defined and iteratively improved.

The model transformations were implemented as Rhapsody plugin in Java and using Xtend templates. The timing analysis is realized using the simulation tool chronSIM which provides timing analysis based on an input model of an embedded system [8]. This tool was chosen because of its ability to simulate embedded software based on a well-defined input model of the system and its easily exchangeable format for the input model. This input model specifies tasks together with their timing properties and stimulation scenarios, among others. Resources and resource dependencies for each task can be defined as well. Besides that, the mapping of tasks to CPUs (Central Processing Units)/ECUs (Electronic Control Units) is modeled.

4 Case-Study

The developed approach was evaluated via a case-study based on a today's in-vehicle infotainment system. The application is composed by functions which are activated by respective buttons and share resources. The system contains the following buttons: *Turning Button*, *Navigation Button*, *Radio Button*, *Telephone Button* and *Media Button*. The *Turning Function* receives two inputs, one correspondent to its activation and the other one correspondent to a turning delta. The other functions have one input each, correspondent to their activation. Each function has one output which is connected to the infotainment main computer. The system was deployed on an 8-bit microcontroller supported by the execution framework.

4.1 Design Model

The case-study was modeled in two packages: an application and a hardware package, in order to keep the software hardware-independent. Fig. 2 presents a piece of the application model and Fig. 3 presents the system model with hardware and allocation of software functions to hardware.

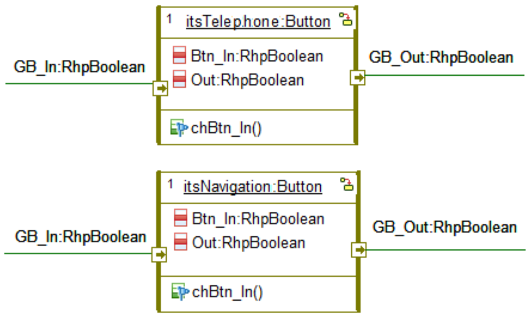


Fig. 2. Small excerpt of the application model

The hardware package consists of further sub-packages, one for each potential target platform. A sub-package called *Processing Element* (PE) was created to model the hardware-specific implementations correspondent to the target platform. The model includes an instance of the application as well as the *Network Ports* corresponding to each resource (buttons and output) described in the previous section. The Network Ports are connected to the *Application* component as shown in Fig. 3. It is worth mentioning that it is possible to deploy the application system to another target platform by defining the respective *Network Ports* and the instance of the application.

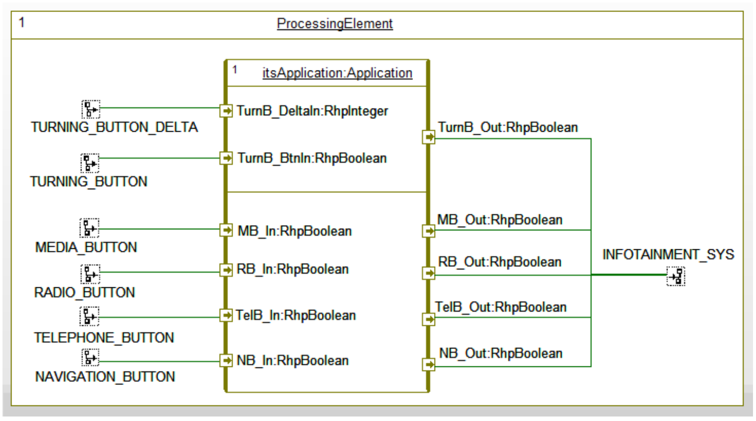


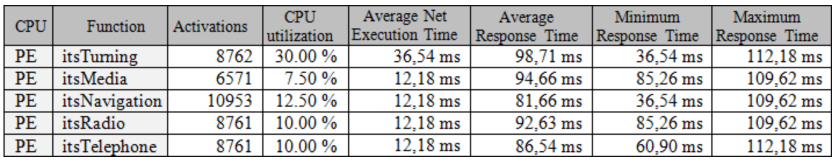
Fig. 3. System model of the infotainment case-study

4.2 Results and Timing Analysis

For this case-study we performed a timing analysis following our integrated approach. As described previously in Section 3.3, the simulation tool receives as input a specific model of the system, which it simulates. Afterwards it delivers a detailed timing analysis and report of the system's timing behavior. The design model presented in Section 4.1 was transformed to an analysis model for the *chronSIM* simulation framework. The system's timing information was added to the generated analysis model and the complete model was then simulated. Based on the detailed timing report generated out of the simulation and whose excerpt is showed in Table 2, the timing analysis was performed. In our case, such analysis results could be used for evaluating end-to-end deadlines and schedulability. Several timing constraints were evaluated for the infotainment case-study. The results of the timing analysis were used for defining and optimizing the scheduling table. Without the loss of generality, we describe the exemplary evaluation of a specific timing constraint in the following. Based on the specification we defined a timing constraint for the function *itsNavigation*. The time spent between identifying a signal from the button being pressed until sending the processed signal to the infotainment main computer shall not take longer than *100 ms*.

For analyzing if the task meets this constraint, we could examine the *Average Response Time (RAvg)* and the *Maximum Response Time (RMax)*. Considering the *RAvg* for *itsNavigation* (*81.66 ms*), the timing constraint was met. However, the monitored *RMax* was equal to *109.62 ms*, which indicates that the previously specified timing constraint was violated. Since other functions presented surplus time-slots, we could adjust the timing behavior of the overall system by modifying the execution order. Therefore the system could meet the *100 ms* required for the navigation function.

Table 2. Excerpt of the timing report



CPU	Function	Activations	CPU utilization	Average Net Execution Time	Average Response Time	Minimum Response Time	Maximum Response Time
PE	itsTurning	8762	30.00 %	36,54 ms	98,71 ms	36,54 ms	112,18 ms
PE	itsMedia	6571	7.50 %	12,18 ms	94,66 ms	85,26 ms	109,62 ms
PE	itsNavigation	10953	12.50 %	12,18 ms	81,66 ms	36,54 ms	109,62 ms
PE	itsRadio	8761	10.00 %	12,18 ms	92,63 ms	85,26 ms	109,62 ms
PE	itsTelephone	8761	10.00 %	12,18 ms	86,54 ms	60,90 ms	112,18 ms

With our case-study we could show that the system's timing behavior can be validated with our integrated method and a valid scheduling for a specific target-platform can be defined. With the *RAvg* and the *RMax*, the engineer can also analyze if the system is schedulable. For that, a schedulability test for *Rate Monotonic Scheduling (RMS)* can be utilized. As discussed previously, with the simulation results and the timing analysis, the engineer can iteratively adjust and validate the design model and furthermore optimize particularly problematic sections (or tasks) of the application.

5 Related Works

There are diverse approaches available for sole timing analysis. For a detailed description of methods and tools for timing analysis we refer to [5]. In [9] [10] [11] [12] [13] [14] approaches for timing analysis and their respective realization in tools with different potential for industrial applicability are based on a particular model of the system and/or on source and executable code. In this way, even when realized in early phases of the development process, the timing analysis is not integrated in the model-driven design demanding extra-effort for developing additional models. The approach presented in [13] targets automotive, electronics, avionics and telecommunication industries. *RapiTime* implements on target an automated performance measurement based on execution traces. Because the approach is based on source files or executable as input and performs the measurements on target, it necessitates the availability of the target platform and does not allow early validation of timing behavior, as mentioned previously.

The automatic or semi-automatic integration of timing analysis in the system's design and development process has been addressed by other academic and commercial solutions but is not fully integrated. In the following, we discuss these works and compare them to our approach. The work presented in [14] integrates simulation-based timing analysis in the design process. The solution allows the automatic generation of a timing analysis model out of a UML design model with an additional UML Profile for timing features. This profile allows the design of a system in UML Activity and Architecture Diagrams with software and hardware models, as well as allocation of tasks to hardware resources considering timing features. . Since the work focuses on timing analysis, its design and timing models are not designed for validation of functional behavior and code generation, unlike our proposed approach.

TrueTime [15] supports a simulation-based analysis of real-time control system which is based, among others, on Matlab/Simulink models. In this way, the approach addresses the integration of timing analysis in the model-driven development. However, Simulink targets on continuous control systems and provides poor mechanisms to describe system architectures. Therefore the approach proposed by the authors is also affected by this disadvantage. The ARTEMIS Project CHESS [16] developed a multi-domain approach with different analysis and views which include not only timing analysis, but also dependability. Differently from our approach, CHESS contains its own modeling language built based on different existent languages and methodologies, such as MARTE. Optimum [17] is a methodology for schedulability analysis of UML models at early stages of the development process. It generates a concurrency model from a workload model designed with UML MARTE (*Modeling and Analysis of Real-Time and Embedded Systems*) and enriched with timing information and computational costs. The methodology is based on the functional model of the system, i.e., on a description of system end-to-end scenarios. In the same way as [16], it differs from our approach since we integrate timing analysis on existent design models which are already the basis for code generation and validation of functional behavior. In [18] the authors propose a scheduling analysis model which categorizes schedule relevant information and show how to perform scheduling analysis on AUTOSAR models using scheduling theory techniques. The proposed model is focused on AUTOSAR system model, containing detailed information about the hardware platform. It differs from our approach since we focus on the timing analysis of the higher level design model. The timing analysis in [16] [17] and [18] is realized with [19]. *MAST is* an open model for representing event-driven real-time applications allowing the validation of timing behavior, including schedulability analysis. *MAST* receives as input a particular model which can also be generated from UML models describing the real-time view of the system. In the same way as, the timing analysis is realized using MAST.

6 Conclusion

With the presented approach for integrating validation of timing-behavior in the design process, an early and detailed timing analysis of the system can be realized. In order to achieve this, a model for timing analysis is automatically generated from the UML design model of the application and hardware platform. The timing analysis is performed based on this generated analysis model by taking into account the characteristics of the target platform. The results can be used by the engineer to iteratively adjust and validate the timing-behavior and the scheduling in the design model of automotive systems. A tool-chain was implemented to realize the approach and can be extended for different timing analysis tools which provide well-defined input model formats, both using simulation-based or static analysis. The approach and framework were validated through an automotive infotainment case-study. We could demonstrate that our integrated approach allows developing correctly timed systems with regards to the scheduling of software components on embedded hardware in early phases of the design.

In future works the modeling process could be extended for supporting timing information directly in the design model. Moreover, we intend to expand our approach for supporting multiple ECUs. This includes not only end-to-end deadlines and a scheduling analysis but also the analysis of allocations to target platforms.

References

1. Anssi, S., Gérard, S., Albinet, A., Terrier, F.: Requirements and Solutions for Timing Analysis of Automotive Systems. In: Kraemer, F. A., Herrmann, P. (eds.) 6th International Workshop, SAM 2010. LNCS, vol. 6598, pp. 209-220. Springer Berlin Heidelberg (2010)
2. Navet, N., Simonot-Lion, F. (eds.): The Automotive Embedded Systems Handbook. Industrial Information Technology series. CRC Press / Taylor and Francis (December 2008)
3. Broy, M., et. al.: Cross-layer Analysis, Testing and Verification of Automotive Control Software. In: 9th ACM International Conference on Embedded Software EMSOFT'11, pp. 263-272, ACM, New York (2011)
4. AUTOSAR, AUTomotive Open System Architecture, http://www.autosar.org/
5. Wilhelm, R., et. al.: The Worst-Case Execution-Time Problem – An Overview of Methods and Survey of Tools. In: ACM Transactions on Embedded Computing Systems (TECS), vol. 7, Issue 3, Article No. 36, ACM, New York (2008)
6. IBM Rational Rhapsody, http://www.ibm.com/developerworks/rational/products/rhapsody/
7. MicroC Execution Framework (MXF), http://pic.dhe.ibm.com/infocenter/rhaphlp/v8/index.jsp?topic=%2Fcom.ibm.rhp.microc.doc%2Ftopics%2Fr_mxf.html
8. chronSIM, http://www.inchron.com/chronsim.html
9. aiT, AbsInt, http://www.absint.com/ait/index_de.htm
10. Li, X., et. al: Chronos: A Timing Analyzer for Embedded Software. In: Science of Computer Programming, Special Issue on Experimental Software and Toolkit, 69, pp. 1-3, (2007).
11. SymTA/P, http://www.ida.ing.tu-bs.de/forschung/projekte/symtap/
12. Ermedahl, A.: A Modular Tool Architecture for Worst-Case Execution Time Analysis. PhD Thesis, Dept. of Information Technology, Uppsala University, Sweden (2003)
13. RapiTime, http://www.rapitasystems.com/system/files/RapiTime%20Explained.pdf
14. chronSIM, Integration, http://www.inchron.com/integration.html
15. Cervin, A., Henriksson, D., Lincoln, B., Eker, J., Årzén, K.: How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime. In: IEEE Control Systems Magazine, 23:3, pp. 16-30, IEEE, (2003).
16. CHESS, Composition with Guarantees for High-integrity Embedded Software Components Assembly, http://www.chess-project.org/
17. Mraidha, C., Piergiovanni, S., Gerard, S.: Optimum: A MARTE-Based Methodology for Schedulability Analysis at Early Design Stages. In: ACM Sigsoft Software Engineering Notes, Vol. 36 Issue 1, pp. 1-8, ACM, New York (2011)
18. Anssi, S., et. al.: Enabling Scheduling Analysis for AUTOSAR Systems. In: Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, 2011 (ISORC 2011), pp. 152-159, (2011).
19. Pasaje, J. L: M., Harbour, M. G., Drake, J. M.: MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems. In: Proceedings of the 22nd IEEE Real-Time Systems Symposium, 2001 (RTSS 2001), pp. 245-256, (2001).