Data flow analysis from UML/MARTE models based on binary traces

Héctor Posadas, Javier Merino, Eugenio Villar
University of Cantabria
Santander, Spain
{posadash, javierm, villar}@teisa.unican.es

Abstract - The design of increasingly complex embedded systems requires powerful solutions from the very beginning of the design process. Model Based Design (MBD) and early simulation have proven to be capable technologies to perform initial design space analysis to optimize system design. Traditional MBD methods and tools typically rely on fixed elements, which makes difficult the evaluation of different platform configurations, communication alternatives or models of computation. Addressing these challenges require flexible design technologies able to support, from a high-level abstract model, full design space exploration, including system specification, binary generation and performance evaluation. In this context, this paper proposes a UML/MARTE based approach able to address the challenges mentioned above by improving design flexibility and evaluation capabilities, including automatic code generation, trace execution collection and trace analysis from the initial UML models. The approach focuses on the definition and analysis of the paths data follow through the different application components, as a way to understand the behavior or the different design solutions.

I Introduction

Embedded Cyber-Physical Systems (ECPSs) are the cornerstones of the new services that are being deployed on the Internet of Things. Their importance will even grow with the full deployment of the Internet of Everything (IoE), when almost all objects in our living environment will be smart and interconnected among them and with the cloud [1].

One of the most critical aspects to solve during this design process is to ensure that system reaction times satisfy the strict timing requirements usually imposed with respect to the data got from and produced to the physical environment where the ECPSs typically operate. This strict timing behavior goes beyond traditional real-time system design [2]. Thus, being able to specify, analyze, simulate and verify timing constraints to lead the design of such complex systems is still an open problem not yet satisfactorily solved.

Timing constraints must be verified all along the design process to ensure that they are satisfied after each design stage. Especially important is the verification process at the initial design stages, when the impact of the architectural decisions taken is potentially higher. In this context, early design-space exploration is essential to take the adequate design decisions that will enable satisfying these non-functional constraints in the final product.

Model Based Design has proven to be a powerful technology to address the development of increasingly complex embedded systems such as ECPSs [3]. For that purpose, standard solutions, such as the Unified Modelling Language (UML), are very important, since it can provide a common, graphical-based formalism for capturing, analyzing and sharing system models, which can also include the refinements and modifications done during the design process. Thus, UML is being proposed not only for initial static design and analysis, but also as a solution to collect information and be used as input for later design steps, such as simulation-based design space exploration [4].

Selecting an optimized architectural mapping is critical in the final system timings, but deciding the most adequate communication and synchronization solutions and the concurrent architecture in general is also critical. However, to perform this exploration, powerful high-level design solutions are required. Once the system is modeled, requirements traceability acquires an important role. Verification of timing constraints on complex systems requires analyzing the system operation, on real prototypes or with simulations when a prototype is not available.

One of the most common mechanisms to perform this analysis is by collecting traces of the events that occurs during execution. Event tracing allows to verify and validate the input constraints at different abstraction levels. Moreover, monitoring methods have an important role in verification and validation steps, while providing information about system state, that can be processed (e.g. filtered, interpolated, etc.) to obtain indications about parameters (e.g., workload characterization, debug action), or characterize specific components (e.g., cache memories, buses, etc.). For example, application components produce and/or consume events associated to the services provided and required at the component interfaces that can be collected and analyzing to evaluate the timing behavior of the system.

In such a context, this work presents an approach capable

This work has been funded by the EU and the Spanish MICINN/AEI through the ECSEL Comp4Drones and the TEC2017-86722-C4-3-R PLATINO projects.

of generating and analyzing traces during the initial steps of the design process. For that purpose, the system must be specified in UML, following a methodology oriented to enable easy design exploration. Then, this UML model is automatically translated into a simulation model, that generates event traces during simulation. Once simulation has finished, the obtained traces are analyzed to extract timing information. To analyze the required internal timing behavior, the user must define, in the UML model, internal data paths to be monitored. These data paths automatically drive the trace collection and the later timing analysis.

To present this approach, the paper is organized as follows: Section II presents the state of the art regarding modeling and trace-based analysis of ECPSs. Section III describes the problem solved. Section IV presents the UML modeling methodology, and section V describes the analysis process. Finally, Section VI presents application results and section VII closes the paper with some conclusions.

II State of the Art

An Embedded Cyber-Physical System (ECPS) is an integration of computation within a physical environment. In such a domain, design challenges arising from the close interaction among the physical processes, the embedded functionality and the communication infrastructure, have to be solved [5]. One of the most important problems to address is ensuring the correct temporal behavior of the system as a whole [6]. Nevertheless, there is a lack in the definition of approaches that try to consider timing requirements into a unified design flow, while considering traceability and link between models and run-time execution.

Several Model Based Design (MBD) commercial tools are available for high-level modelling and simulation. One of the most popular is Matlab/Simulink [7]. However, being based on a single Model of Computation and Communication (MoCC) is the main limitation affecting simulation speed and code generation efficiency. CoFluent is other commercial tool extended to model IoT systems [8]. Although supporting more interaction models that Matlab/Simulink, it is also limited in the way components may interact among them. In both cases, the model is functional and no performance analysis is made depending on the underlying heterogeneous platform.

Another alternative is to use UML language as a base for MBD [9]. The problem is that UML is very flexible but lacks the semantical content required in most application domains. Therefore, the tendency has been the proliferation of Domain-Specific Languages (DSLs) [10]. Metamorph is a good example of a framework following this approach [11].

Among the available DSLs, UML/MARTE is the standard language for real-time and embedded systems design [12]. Several modeling environments such as Modelio [13] and Papyrus [14] support UML/MARTE. Nevertheless, its flexibility and semantical richness requires the definition of efficient modelling methodologies [9].

One of them is Time4Sys, a framework developed as a Polarsys pluging with the objective to bridge design and analysis tools without changing the development framework. The main goal is timing analysis. It supports schedulability analysis based on ‘Worst-Case Execution Times’ (WCET). The tool also supports workload simulation assigning constant execution times to the tasks (subtasks) [15].

UML also allows the specification of timing constraints using activity diagrams [16]. Due to the importance of timing behavior in embedded systems design, MARTE extends the UML capability in specifying timing relations among events with the Clock Constraint Specification Language (CCSL) [17]. Nevertheless, few methods allow to specify the timing constraints and verify them during system simulation

On the other hand, there are examples of software based profiling systems, that depend of the application (e.g., Gprof [18], LTTng [19]), but are not linked with high-level methodologies. Furthermore, software profiling necessarily introduces some overheads on execution time and, considering the sampling approach, it has some grade of statistical inaccuracy. Regarding unifying design and monitoring approaches, the work in [20] use time triggered run-time verification that seeks to minimize the software overhead of run-time verification for multi-core systems. [21] presents a runtime verification system that utilizes live sequence charts, that are similar to UML sequence diagrams and enable support modeling multiple system behaviors, conditional execution sequences, and activation timing, using Linear Temporal Logic (LTL). Copilot [22] is a compiler-assisted approach that automatically instruments a software binary with custom verification code compiled from a requirements specification language. It generates verification code, capable of statically analyze timings.

Moreover, there is a lack of tools able to generate performance models of the system depending on the architectural mapping over heterogeneous platforms [9]. This is the background technology we propose to extend to support trace-based timing constraint verification of the PSM, considering the HW resources selected.

III Problem description

As stated above, the development of complex embedded systems is a difficult and time-consuming task. To minimize these problems, it is required to have a tool providing flexibility, scalability and reusability capabilities, specially at the beginning of the design process. To accomplish this goal, this paper uses a component-based UML/MARTE system modeling methodology. Components are usually isolated pieces of functionality that can be developed independently of its use and reused in different projects. This is also the case of legacy or third-party components.

However, achieving the independency required to separate development and reuse is not so simple. On the one hand, components must be as platform independent as possible. The target platform where they will be executed can be unknown during development, or it can change when being reused. On the other hand, when reused, the architecture of the system application can be quite different. Thus, a reused component can be integrated in a completely different structure compared with the design where it was originally developed: services can be requested with different communication semantics, or the functionality has to be run in a completely different model of computation.

To design components completely independent from its implementation and use, first, component internal functionality must be designed in a platform independent way. Additionally, components should support interconnections among them without restrictions, specially in terms of communication semantics, synchronization mechanisms or model of computation.

To achieve so, application components must be developed separating functionality and communication. Additionally, all the information about synchronism, communication buffers or concurrency as component parameters in the UML model instead of in the source code. Then, an automatic code generation tool is used to create the glue code required to implement this information.

This approach also improves exploration possibilities, since the proposed automation makes it easy to evaluate system performance when each component operates under different concurrency and communication parameters. As a result, the information added in the UML model can be used to define the Model of Computation (MoC) under each component will operate, including MoCs such as Kahn Process Network (KPN), Synchronous Data Flow (SDF), Timed Data Flow (TDF), Synchronous Reactive (SR), etc.

However, these MoCs are typically defined to ensure that the resulting system has certain characteristics, but considering that all the components of the system follows the same MoC. For example, in KPN, no input data is lost during the computation process. However, if we use a set of periodic processes that communicate using shared variables, it is very likely that some data will be overwritten on that intermediate variables before used.

To analyze so, the proposal we present in this article is to define the most important paths followed by the data internally through the system. These data paths are defined as the list of services each data must cross from the input to the output, or between two internal services of the system.

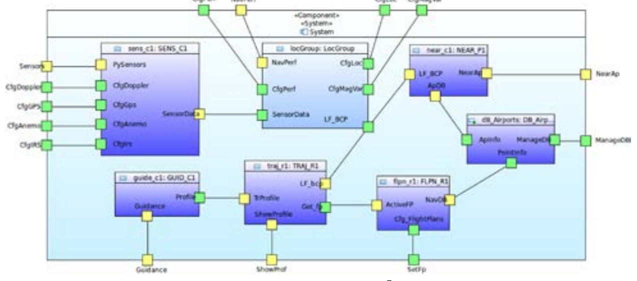

Figure 1.- UML system description

Analyzing data flow on these paths, and putting constraints, such as the maximum latency for an specific path from input to output, can let the designer to have a more clear idea of the resulting operation, and to drive the exploration process, selecting or rejecting different alternatives. That way, the traces collected at the specific marks added in the automatically generated glue code and their later analysis can help designers to optimize the system.

IV UML Modeling Methodology

A.- Methodology Overview

The first step in the proposed design process is the modeling of the full system in UML. To do so, the modeling methodology must fulfill three requirements. First, it has to minimize the modeling effort and to reduce the number of mistakes, the modeling methodology should be simple. But, at the same time, the model must include all the information required to automatically perform the evaluation of each design alternative the designer selects, as described in next section. Additionally, the capability to reuse components or to integrate legacy or third-party components requires the use of standards. Thus, in this paper, the OMG standards UML and MARTE, are used to specify the system.

To capture all the relevant information in a simple way, the system model is divided in ‘views’ which conforms three sub-models: Platform-Independent Model (PIM), Platform Description Model (PDM) and Platform-Specific Model (PSM). The PIM describes the functional application, independently on the target platform, the PDM describe the target platform, and the PSM describes how the application is mapped into the resource of the platform.

As stated before, the fundamental modeling element is the component. Thus, the definition of the application (PIM) is divided in two parts: the description of the application components themselves, and the description of how these components interconnect to create the full application.

Components communicate with other components through ports in a client/server approach. Ports contain interfaces which define the communication methods, called services. The components either require communication services through required interfaces and/or offer communication services through provided interfaces (Figure 1). Ports and interfaces specify the semantics of each communication. Using them, it is possible to define if a service call will be synchronous or asynchronous; if multiple calls can be executed in parallel or they must be protected; if its execution must be immediate or a request can be stored in a queue; the definition of timeouts, retries, etc.

Application components also specify its internal functionality. As a preferred solution, each component is linked to the C++ files containing its functional code. In our approach, application components have two ways for executing its internal functionality. First, each component can have an internal execution flow, which automatically starts when the component is created. This functionality can have any pattern, but it is usually based on a loop that executes periodically, or depending on a certain event. Secondly, components provide services that are executed when a client requests them.

Additionally, functional and extra-functional constraints may be imposed to the application components and to the services, that must be verified during system development.

Then, the system application is conceived as a hierarchical network of application components. Once the components are specified, the system functionality is obtained as a composition of such components, connected each other through concrete, compatible ports and interfaces.

Once the application is defined, it is required to describe the HW platform, also in a hierarchical component-based approach. A complex system can have several nodes, being each node composed by one or several processors, with busses, memories and peripherals. Finally, depending on the platform details, application components are grouped in memory spaces that lead to the creation of one or several executables (one per memory space), which are mapped into the hardware resources. Mode details can be found in [4].

B.- Data path definition

This paper proposes analyzing the internal communications to know the behavior of the resulting system when multiple MoCs are used in it. Our proposal is to define the most important paths that data follow within the system, from input to output, or between internal points. For example, to define a data path we can consider an input data A that is used by service 1 to generate data B, which is used by service 2 to generate data C, which again is used by service 3 to generate data D, and so until the output. These services or internal loops can be part of the same or different components. Once one data is received in a component, it can be computed internally by its internal loop (or not), and it can be sent to another component through a call to a service of this component.

To be able to get and analyze this information, the most important data paths must be described in the UML model. The modeling of these data paths in UML is done as shown in Fig 2. The modeling proposed is based on a sequence diagram, where each lifeline represents an application component. Service calls are identified in the diagram by arrows with the name of the corresponding service. Internal component loops are defined by a rectangle located in the lifeline. When an arrow starts from a rectangle, it is considered that it is executed by the component internal loop. When it directly connects with the lifeline (no rectangle) it is considered that it is executed by the service executed as a result of the previous incoming service request. For example, in figure 2, service trHightFreqBCP is requested by loc_c1 and provided by loc_c2. As a result of the call, the D_BestComputedPosition datum is sent from loc_c1 to loc_c2. Then, this datum is computed internally in the component loop and the result is gathered by traj_r1 calling to getCurentBCP service, which gets this data from loc_c2. Additionally, it is important to note that service calls can deliver or gather data, depending on whether the data is read or written in the service call. This fact specially impacts the later analysis: each service is monitored by three control points: the service request, the execution start and the end of the service (In loops, it corresponds to the beginning and the end of each iteration). Depending on the type of service and the communication semantics these tree points are considered in a different way: the analysis is not the same in an asynchronous call, than in a rendezvous.

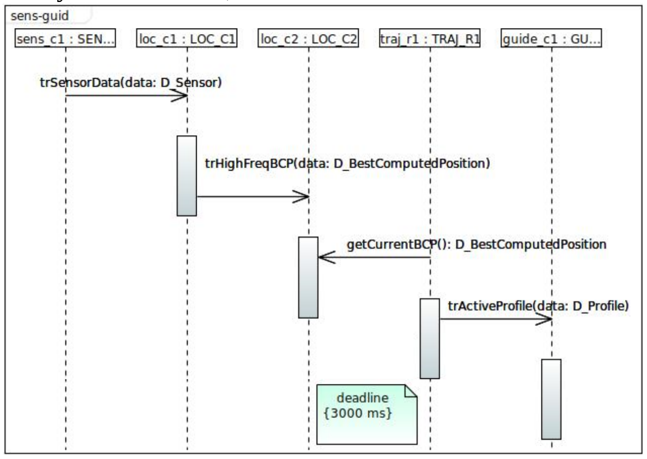
sens-guid
sens_c1 : SEN...
loc_c1 : LOC_C1
loc_c2 : LOC_C2
traj_r1 : TRAJ_R1
guide_c1 : GU...
trSensorData(data: D_Sensor)
trHighFreqBCP(data: D_BestComputedPosition)
getCurrentBCP(): D_BestComputedPosition
trActiveProfile(data: D_Profile)
deadline
{3000 ms}

Figure 2.- Data path model

Two more elements must be considered when defining the graph. The first element appears when a service call acts both as a deployer and a gatherer (inout argument). As a result, the data path must continue from the client instead of from the server. To detect so, the analyzer identifies the type of the service depending on the location of previous and next arrows. The second problem is service call chaining. A service can call another service inside its code, resulting in deployer or gatherer chains.

Finally, constrains can be specified for the data path. As a result, only chains fulfilling the constraint are considered correct. For example, in figure 2 a deadline value is defined, specified the maximum amount of time of a data path, from the input to the output.

V Evaluation process

The execution flow of the infrastructure developed for the analysis of the system data paths is shown in figure 3. The flow starts with the development of the UML/MARTE model. Eclipse/papyrus is used for that purpose. Then, this model has to be executed to generate the traces for the analysis. The glue code generated is standard C++ code that can be compiled and used in most simulators or in real boards, so the proposed approach is no limited to a single simulation tool. In order to obtain results for this paper, the tool described in [4] has been adapted to automatically generate simulation models using Vippe[23] simulation tool, which uses host-compiled annotation.

In order to create the simulation model, three elements are required: the application executable codes, the makefiles required to compile them, and the XML files used to configure the platform model in the simulation tool (number of cores, processor frequency…). Executable codes are automatically created from the information stored in the UML model, combining the internal component functionality with synthesized glue code used to implement the specified communication semantics. This glue code also includes the trace points at the beginning and end of the services for later analysis, avoiding manual user intervention.

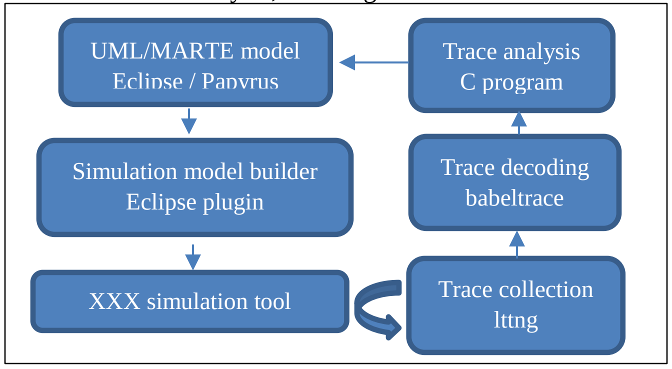

Figure 3.- Data path analysis flow

Once all the elements required for the simulation are created, the simulation is performed, collecting traces which are automatically analyzed reporting information about the system behavior, such as the execution times of the different services, number of calls to them, data paths correctly executed, internal data overwritten or read multiple times.

To collect the traces, the binary CTF (common trace format) format has been used. This format enables collecting large amount of data in a reduced memory area. For that collection, the LTTng tracing application has been used as SW monitor. Additionally, babeltrace tool has been used to decode the binary CTF files. The decoded output is dynamically loaded by a specifically generated C program that performs the analysis, following the information specified in the model, specially the data paths. As a result, the tool reports information about the correct path iterations, and their timing characteristics (maximum, mean and minimum time), duplicated and lost data paths, number of correct data paths accomplishing the constraints sets, etc.

Currently, only remote procedure call (RPC) and rendezvous (RV) communication semantics have been covered, while other characteristics such as FIFOs, retries or timeouts will be covered in the future.

VI Results

The proposed approach has been applied to a safety-critical Flight Management System (FMS) of an airplane, provided by Thales. The purpose of the FMS is to provide the crew with centralized control for the aircraft navigation sensors, computer-based flight planning, and geographical situation information. From pre-set flight plans (take-off airport to landing airport), the FMS is responsible for the plane localization, the trajectory computation allowing the plane to follow the flight plan, and the reaction to pilot directives. The FMS has been decomposed into several components, as shown in figure 1. It computes various data (i.e., exact location, trajectories, and nearest airports list among many others) and it sends guidance instructions to the autopilot and to a display.

The example has been used to analyze the system behavior under different platform architectures and communication semantics. Thus, the goal of the example is not to demonstrate the best implementation for the FSM code, but the possibilities the analysis infrastructure proposed can provide to system designers. All simulations have been created automatically from the UML model, minimizing designers time and effort.

Table 1: Simulation data for RPC system implementation

Cores	GHz	Simulated time (sec)	Total events	Data In	Data Out
1	1500	4022	648323	20114	804
	750	4022	648323	20114	804
	325	4022	608783	18884	804
	150	4022	287545	5206	804
2	1500	4022	648326	20114	804
	750	4022	648326	20114	804
	325	4022	648326	20114	804
	150	4022	505571	17324	804
4	1500	4022	648330	20114	804
	750	4022	648330	20114	804
	325	4022	648330	20114	804
	150	4022	648330	20114	804

The code has been simulated under different platform configurations, changing the number of cores and processors' frequency. Additionally, the communication semantics of a set of services have been modified, generating two implementations: a synchronous RPC (remote procedure call) implementation, and another where half of the internal system communications have been changed from RPC to RV. Additionally, the change from RPC to RV has forced multiple tasks to lost their waiting periods, relying on the RV synchronization to get new data. As a result, four tables have been generated: Tables 1 and 3 with general simulation information for both full RPC and RPC+RV and tables 2 and 4 with the analysis of data the path described in figure 2.

Table 2: Data path info for RPC system implementation

Cores	GHz	Max time	Min time	Mean time	Full Path	Data Rep.	Data Lost	Deadl. Lost
1	1500	2780	1370	2075	804	0	19283	0
	750	2940	1580	2272	804	0	19283	0
	325	3110	1320	2301	804	0	18053	93
	150	6101	2371	4202	804	0	4393	768
2	1500	2785	1385	2087	804	0	19283	0
	750	2770	1370	2072	804	0	19283	0
	325	2940	1550	2247	804	0	19283	0
	150	3150	1371	2373	804	0	16495	131
4	1500	2795	1395	2097	804	0	19283	0
	750	2780	1380	2081	804	0	19283	0
	325	2750	1350	2052	804	0	19283	0
	150	2860	1260	2157	804	0	19283	0

As results show (table 1), in RPC, the frequency reduction implies a reduction on the number of events, and input data read, since the processor can reach 100% CPU utilization, especially in the case of 1 and 2 cores. As a result, the minimum time required to execute the full data path is reduced as frequency increases (table 2). The reason is that, typically, data needs to wait the next iteration of each task to be computed. However, as the CPU reaches 100% utilization deadlines are lost, tasks execution order is altered, leading to punctual smaller minimum data path times, while the mean and max times are maintained or even increased. Additionally, all output data are the result for different input data (full paths), while multiple input data have not generated output data (data lost). Thus, an exploration of the most adequate task order could improve system latency.

Table 3: Simulation data for RPC+RV system implementation

Cores	GHz	Simulated time(sec)	Total events	Data In	Data Out
1	1500	4022	816986	20114	20114
	750	4022	816986	20114	20114
	325	6035	768023	20113	13817
	150	18755s	1028516	20113	14558
2	1500	4022	816988	20114	20114
	750	4022	816988	20114	20114
	325	4022s	816988	20114	20114
	150	7393s	768988	20113	11826

In RPC+RV (table 3), synchronizations modify data operation. A lot of blockages occur in the system, especially as the frequency decreases. Thus, the execution time of the proposed test-bench increases, and so, the number of events. However, as calls are synchronized by rendezvous, the min, mean and max times required to execute the data path always increase when the CPU utilization reaches 100% (table 4). At the same time, as the period of the last task has been removed, more output data are obtained, however, the generated amount of output data is not constant. Additionally, most of the output data are duplicated (data rep) due to multiple readings of intermediate data variables.

Table 4: Data path info for RPC+RV system implementation

Cores	GHz	Max time	Min time	Mean time	Full Path	Data Rep.	Data Lost	Deadl. lost
1	1500	390	220	242	4022	16086	16088	0
	750	309	260	270	4022	16087	16088	0
	325	1790	928	1269	6091	7722	14016	0
	150	4860	2850	3747	14523	32	5586	14380
2	1500	410	225	365	4022	16086	16088	0
	750	410	240	370	4022	16087	16088	0
	325	460	260	337	4022	16087	16088	0
	150	2199	1209	1693	7394	4429	12715	0

VII Conclusions

The paper presents an integrated solution, that from a UML model of the system, is capable of automatically generate a simulation model, collect execution traces and analyze the results. To drive the trace collection and analysis, it is possible to specify data paths within the system.

According to them, the designer can understand the system internal behavior and easily analyze different design alternatives, including different platform configuration and different communication semantics, and in the end models of computation. Currently, only remote procedure call (RPC) and rendezvous (RV) communication semantics have been covered, while other characteristics such as FIFOs, retries or timeouts will be covered as a future work.

REFERENCES

[1] L. Jóźwiak, “Advanced mobile and wearable systems”, Microprocessors and Microsystems, V.50, 2017, pp.202-221

[2] E. A. Lee and S. A. Seshia, “Introduction to Embedded Systems: A Cyber-Physical Systems Approach”, Second Edition, MIT Press, 2017.

[3] F. Mallet, E. Villar, F. Herrera, "MARTE for CPS and CPSoS", in S. Nakajima, J.P. Talpin, M. Toyoshima and H. Yu: "Cyber-Physical System Design from an Architecture Analysis Viewpoint: Communications of NII Shonan Meetings", Springer, 2017.

[4] H. Posadas, E. Villar et all., “Mega-Modeling of complex, distributed, heterogeneous CPS systems”. Microprocessors and Microsystems, Springer, Accepted.

[5] Y. Z. Lun, A. D’Innocenzo, F. Smarra, I. Malavolta, and M. D. D. Benedetto, “State of the art of cyber-physical systems security: An automatic control perspective,” Journal of Systems and Software, 2019.

[6] A. Shrivastava et al., "Time in cyber-physical systems," Proc. of CODES+ISSS, IEEE, 2016, pp. 1-10.

[7] Simulink, https://es.mathworks.com/products/simulink.html.

[8] Cofluent, https://www.intel.es/content/www/es/es/cofluent/cofluent-studio.html.

[9] F. Herrera, J. Medina, E. Villar, "Modeling Hardware/Software Embedded Systems with UML/MARTE: A Single-Source Design approach", in Soonhoi Ha and Jürgen Teich (Eds): "Handbook of Hardware/Software Codesign", Springer. 2017.

[10] M. Brambilla, J. Cabot & M. Wimmer, “Model-Driven Software Engineering in Practice”, Morgan & Claypool Publishers, 2017.

[11] Metamorph, https://www.metamorphsoftware.com/.

[12] MARTE, https://www.omg.org/spec/MARTE/1.0/PDF.

[13] Modelio, https://www.modelio.org/.

[14] Papyrus, https://www.eclipse.org/papyrus/.

[15] Time4Sys, https://www.polarsys.org/time4sys/.

[16] L. Xuandong, C. Meng, P. Yu, Z. Jianhua and Z. Guoliang, "Timing Analysis of UML Activity Diagrams", in M. Gogolla and C. Kobryn, Cris, “UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools", Springer, 2001.

[17] F. Mallet, “Logical Time @ Work for the Modeling and Analysis of Embedded Systems”, Lambert Academic Publishing, 2011.

[18] S. L. Graham, P. B. Kessler, and M. K. Mckusick, “Gprof: A call graph execution profiler,” Proc. of SIGPLAN, 1982, p.120–126.

[19] LTTng: an open source tracing framework for Linux, 2020 (accessed: 18.04.2020). https://lttng.org/.

[20] S. Navabpour, B. Bonakdarpour, and S. Fischmeister, “Time-triggered runtime verification of component-based multi-core systems,” in Runtime Verification (E. Bartocci and R. Majumdar, eds.), pp. 153–168, Springer International Publishing, 2015.

[21] M. Chai and B.-H. Schlingloff, “Monitoring systems with extended live sequence charts,” in Runtime Verification (B. Bonakdarpour and S. A. Smolka, eds.), pp. 48–63, Springer International Publishing, 2014.

[22] A. Nassar, F. J. Kurdahi, and W. Elsharkasy, “Nuva: Architectural support for runtime verification of parametric specifications over multicores,” in Proc. of CASES, 2015.

[23] L. Diaz, E. Gonzalez, E. Villar, P. Sanchez, "VIPPE, parallel simulation and performance analysis of multi-core embedded systems on multi-core platforms", DCIS, 2014