Performance Analysis Method for RT Systems: ProMARTES for Autonomous Robot

Konstantinos Triantafyllidis, Egor Bondarev, Peter H.N. de With
Eindhoven University of Technology
5600 MB, Eindhoven, The Netherlands
{k.triantafyllidis, e.bondarev, p.h.n.de.with}@tue.nl

***Abstract*— In this paper we present a cycle-accurate performance analysis method for real-time systems that incorporates the following phases: (a) profiling SW components at high accuracy, (b) modeling the obtained performance measurements in MARTE-compatible models, (c) generation, scheduling analysis and simulation of a system model, (d) analysis of the obtained performance metrics and (e) a subsequent architecture improvement. The method has been applied to a new autonomous navigation system for robots with advanced sensing capabilities, enabling validation of multiple performance analysis aspects, such as SW/HW mapping, real-time requirements and synchronization on multiprocessor schemes. The case-study has proved that the method is able to use the profiled low-level performance metrics throughout all the phases, resulting in high prediction accuracy. We have found a range of inefficient design directions leading to RT requirements failure, and recommended to robot owners a design decision set to reach an optimal solution.**

Keywords*— profiling, modeling, component-based, simulation, scheduling analysis, performance analysis, evaluation, prediction, assesment, optimizaion, real-time embedded system, MARTE

I. INTRODUCTION

The composition of real-time systems based on the mapping of SW and HW components, has become an adopted practice, since it enables rapid prototyping and development of a system from existing blocks. The resulting real-time systems still should meet the performance requirements, such as throughput, latency, etc. At the early composition phase, the architect needs reliable assessment methods in order to evaluate and predict the performance of the designed system. An incorrect performance prediction may lead to adopting an inefficient system architecture with the consequences of system re-design or re-implementation.

The challenge of performance predictions comes from the fact that at design-time the HW and SW components implementations are frequently not available. Instead, models representing abstractions of the components are used for computation and reasoning. Accuracy of such models is vital for obtaining reliable performance predictions on behaviour of the future system. Component vendors, supplying the models, still face the challenges of automated generation and detailed profiling/specification of models.

The analysis of the component composition should take into account the intrinsic properties of the hardware, such as cache hierarchy and dynamics, bus/network congestion, tasks floating across the processor cores and parameter-dependent execution/workload. These system aspects severely complicate the performance analysis of a composed system, even if an architect deploys detailed and accurate component models.

Another challenge comes from the limitations of analysis mechanisms, which are normally classified in two categories: analytical (formal) methods and simulation techniques. The former are not able to provide a detailed execution timeline, while the latter cannot guarantee reachability of worst cases.

In this paper, we present a cycle-accurate performance analysis method for real-time systems (ProMARTES). Our analysis method consists of four individual phases: (a) *profiling* SW components at high accuracy, (b) *modeling* the obtained performance measurements in MARTE-compatible models, (c) *composition*, *scheduling analysis* and *simulation* of a system model, (d) *analysis* of obtained performance metrics and (e) a subsequent architecture improvement. The presented method is the cornerstone of our Design Space Exploration (DSE) approach [1], targeting automated identification of optimal architecture alternatives.

In our previous work [1], we have presented the first phase of cycle-accurate profiling and parameter-dependent MARTE-based modeling of individual components. In this paper, we extend this phase with network utilization metrics and a detailed memory usage model, thereby addressing the above-mentioned challenges of model generation.

The focus of this paper is on the component/system composition, performance analysis and evaluation phases. The following paragraphs outline the contributions to these phases.

At the component *composition* phase, candidate SW/HW architectures and a set of workload scenarios are defined. The SW architecture is represented by composition of individual components with associated performance models. The mapping of software components on hardware nodes defines the SW/HW architecture. A set of scenarios defines the worst-case workload on a system. The instruction-level metrics of models are converted to processor-specific execution-time metrics, thereby incorporating intrinsic hardware properties.

At the system *evaluation* phase, we perform scheduling analysis and simulation of the scenarios, obtaining the usage and sharing of all involved hardware IP blocks. The scheduling and simulation results provide predicted performance properties, i.e. latencies, throughput and bottlenecks of the designed system. Comparing the predicted properties to the system requirements, we identify weak points of the candidate architecture which direct us to a more efficient alternative.

To validate this method, we have performed a case study on the real-world problem of a new autonomously navigating robot with advanced sensing capabilities. The most critical performance attributes of the system are latency in the navigation control loop and throughput. We have profiled and modeled the available components, proposed a number of architectural alternatives and analyzed them with respect to the critical attributes. Based on the analysis results, we have proposed to the robot owners the optimal architecture with low HW costs that still satisfies the real-time requirements.

The sequel of this paper is as follows. Section II records the related literature to our work. Section III explains the overall DSE methodology. Section IV, V and VI describe the method in detail. Section VII presents the tooling developed for the method. Section VIII illustrates the case study used for the validation of our method. Section IX describes our findings from the case study. Section X concludes the paper.

II. Related work

In the last decade, the real-time research community developed several innovative methods addressing the problems of SW/HW component modeling, predictable assembly and evaluation of real-time systems. Currently, a wide variety of modeling profiles are available for composition of (real-time) embedded systems: SysML, UML-RT, MARTE and AADL. The SysML [11] is a UML profile for specifying, analyzing and verifying complex systems that may include hardware, software, information, personnel, procedures and facilities. However, SysML does not provide sufficient primitives for the real-time systems domain. Thereby, the OMG group released the MARTE profile [20], which targets specification of real-time and embedded systems. MARTE enables the HW and SW modeling and defines specific primitives for timing and power-consumption analysis. Nevertheless, MARTE lacks low-level resource modeling metrics, such as instructions, effective execution cycles and cache misses. Another alternative, AADL [23], enables modeling of SW and the HW components of real-time embedded systems. The AADL models are of high abstraction level and therefore do not provide cycle-accurate modeling primitives. Numerous composition and evaluation approaches have been proposed based on the aforementioned modeling profiles.

Cortellessa *et al.* [7] have proposed a comprehensive approach for modeling, composition and mapping of SW components onto HW platforms and consequent behavior simulation. Their modeling methodology is based on UML-RT [10], which is the predecessor of the UML-MARTE. The simulation is performed by RRT [14] which is a proprietary simulator and this limits broad applicability of the approach. However, this approach can be adopted by any modeling and simulation technique without focusing on a specific toolkit.

For automotive real-time systems, Klobedanz *et al.* [2] have discussed a performance analysis technique based on the AUTOSAR [22] model. The technique targets such performance attributes as CPU load, end-to-end latency and throughput. The AUTOSAR model lacks high-detailed performance attributes, since it is aiming at simple ECU-type nodes and therefore can be applied only in the automotive domain.

For a general-purpose distributed real-time system, Maatta S. *et al.* [17] have proposed an analysis method based on combining the UML-MARTE profile and Ptolemy II simulator [15]. The authors target the methodology to multi-core network-on-a-chip platforms. The Ptolemy II broadly supports network communication schemes, which is crucial for performance analysis of RT distributed systems. Moreover, it provides the majority of the real-time scheduling policies. A limitation of this method is that the conversion of the MARTE models for the Ptolemy II tool is not fully automated, which requires a high effort of the architect during the analysis.

Following a completely different analysis approach, both Pimentel *et al.* [4] and Silvano *et al.* [3] annotate the source code of the SW components with relative performance costs of the corresponding HW platform and simulate the execution providing the performance results. Both approaches require source code of the components and cannot be considered as a conventional model-based solution.

In the domain of pure model-based techniques, Bondarev *et al.* [6] [8] have proposed a solution for design and performance analysis of conventional CBSE embedded real-time systems (ROBOCOP components [13]). The CARAT toolkit supporting the approach, synthesizes SW/HW component models, constructs a system model with corresponding scenario models and simulates the resulting models for worst-, best- and average-cases of CPU load, latency and throughput. CARAT supports EDF, RMA and DMA scheduling algorithms, but it does not support network modeling and simulation. Also, CARAT does not provide a cycle-accurate profiling tool for components, which leads to less precise performance analysis.

In contrary to the above-described simulation- and scheduling-based techniques, Thiele *et al.* [5] have presented a compositional method, incorporating both types of techniques for distributed embedded systems. The Modular Performance Analysis (MPA) approach models resources and their usage in a high abstraction layer, while the performance components represent the transformation of the input timing properties to the output timing properties. The modularity of the MPA approach enables the analysis and the exploration of different mapping and resource sharing strategies. As a result, the technique guarantees rapid identification of the worst-case resource load and latencies. However, intrinsic cycle-accurate execution properties cannot be incorporated during the analysis, while the task execution timeline and interleaving aspects cannot be obtained with this technique.

III. Overview: Architecture Analysis & optimization

The presented performance evaluation approach is part of a larger Design Space Exploration framework for real-time embedded systems, which has been developed over the last decade [1] [7] [9]. Let us first outline the framework phases, that are subdivided into three blocks: (1) Profiling and Modeling, (2) Architecture Composition and (3) Architecture Evaluation and Optimization (Fig. 1).

During the *Profiling and Modeling* phase, the component developer profiles the developed SW components at cycle-accurate instruction level and generates a performance model for each individual component [1]. Each performance model may target various hardware usage aspects (CPU, BUS, RAM, Network, etc.) and can be specified for multiple platforms. Our tooling supports automated profiling and model specification, as well as the repository placement for the subsequent phases.

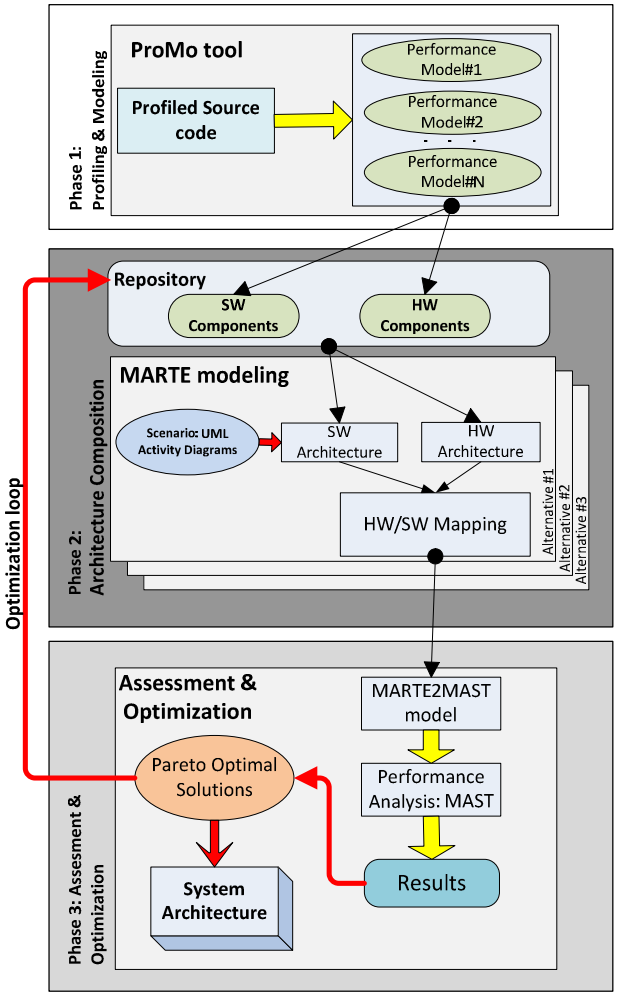

Fig. 1. Overview of the DSE methodology. We focus on the Architecture composition, analysis and assessment phases

The *Architecture Composition* phase aims at selection of the required components (based on functional requirements) and the automated generation of a model of the composed system using defined workload scenarios. The composition can be performed for a number of architectural alternatives. Each alternative design includes the component instantiations and connections, as well as the mapping on a selected HW platform. Being applied onto the critical execution scenarios, the design specification is converted into a system model. Other challenges addressed are the support for multiple component architecture styles and provisioning of resulting composition models in common formats (MARTE, UML, etc).

The *Analysis and Optimization* phase enables evaluation of system performance properties by schedulability analysis and simulation of the obtained system models. Both techniques support various hardware platforms, multiple scheduling policies and different network protocols. Performing schedulability analysis and simulation of the system model brings predicted performance properties such as latency, hardware use and throughput. The system is validated with a comparison to the requirements, leading to consequent design iteration(s). Each iteration searches for an optimal architecture by tuning the allowed *factors of freedom* (SW component, hardware structure, SW/HW mapping and scheduling policies).

In the sequel, we focus on the architecture composition and analysis blocks of the framework.

IV. Detailed Profiling and Modeling

With the ProMo tool [1], a component developer profiles the execution behavior and hardware resource usage of each individual SW component. The ProMo tool provides the following benefits. Firstly, the profiling phase is *generic*, supporting the majority of the CPUs available (AMD, Intel, ARM). Secondly, the obtained performance metrics are cycle-accurate, since the measurements are directly collected from the performance monitor unit of the attached CPU. Finally, the automatically composed performance models are compatible with MARTE or AADL resource models.

The ProMo tool has been extended in order to support network utilization metrics and a more detailed memory usage model. The structural view on the model is depicted in Fig. 2.

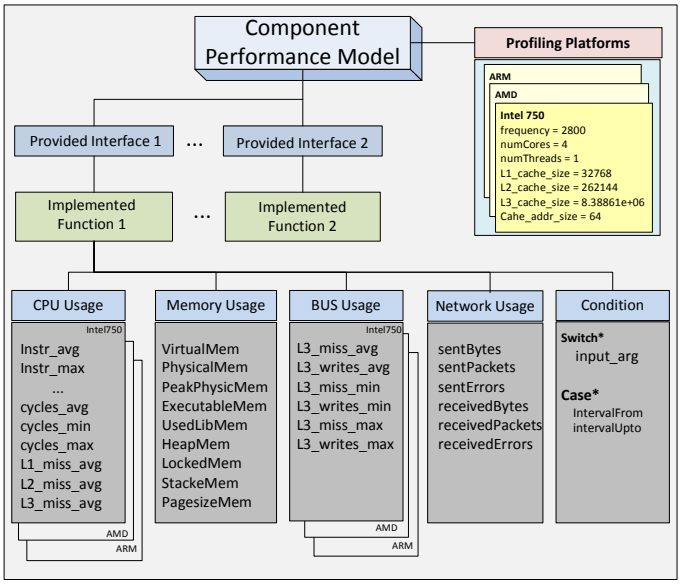

Fig. 2. Structure of a component performance model

We model performance attributes for each operation of the provided interfaces of the component. For each operation, the attributes are grouped into four types: `cpuUsage`, `memoryUsage`, `busUsage` and `networkUsage`.

The Profiling and Modeling phase supports identification and specification of best-, average- and worst-cases execution of the profiling, as well as parameter-dependent usage of hardware resources.

V. ARCHITECTURE COMPOSITION

At this phase, an architect selects software components that may satisfy the defined functional and extra-functional requirements and graphically specifies the component composition by instantiating and connecting the involved components. The hardware architecture can be specified in parallel, but in most of the cases, a hardware platform is already pre-specified. If not, an architect can select available hardware components from a repository and choose a specific topology, number of processing nodes, types of memory, communication means and scheduling policy.

Once the software and hardware architecture are specified, the mapping of the software components on the hardware nodes is made. The mapping shows on which processing node each software component should be executed. Efficient mapping is required to distribute the load of hardware resources in an optimal way. However, at the first mapping iteration, it is not clear how to best deploy the software components to achieve the optimal load distribution. Various mapping alternatives are possible at this stage. Each alternative represents a system architecture.

Additionally, an architect needs to define the workload on a system by means of execution scenarios, which represent either internal or external triggers for the system and the operations that are invoked by those triggers.

The creation of a system model is based on the performance models of the involved components, the scenario models and the SW/HW mapping architecture. The resulting system model represents an executable structure that can be simulated and/or analysed for performance.

VI. SYSTEM MODEL ANALYSIS

The system model obtained from the previous phase is applicable to both types of evaluation techniques: schedulability analysis and simulation.

Schedulability analysis enables prediction of the best- and worst-case response latencies for each task instance, associated with a real-time deadline. This type of analysis provides guaranteed worst-case boundary conditions and can be executed within few seconds. However, it does not provide detailed behavior timeline data and average-case resource usage and latencies. To find the latter metrics, we also apply simulation techniques.

A simulation-based analysis deploys *JsimMAST* virtual schedulers that simulate the execution of the tasks specified in the system model. The selection of scheduling algorithms is dictated by the types and the number of the used CPUs/HW platforms, the protocols and the topology of the deployed communication lines/networks and the operating system used for the composed system. While simulation techniques cannot guarantee identification of worst-case executions, they provide detailed system behavior (execution timeline of the system tasks), thereby enabling identification of possible bottlenecks already at the early design phases. However, simulation requires a substantial time span (from minutes to days) to obtain stable prediction results and therefore can be selectively used for a detailed exploration of execution problems in the architecture, such as buffering and task interleaving problems.

The worst-case performance properties obtained from the schedulability can be further used as a guideline for next iterations of the design space exploration process, since the analytical techniques are time-efficient. At the saturation point of identification of local or global optima, the simulation techniques facilitate the analysis of other efficient alternatives.

VII. TOOLING

A. Marte2mast

In the *Architecture Composition* phase we specify the generated system model by using the MARTE profile. In order to simulate the composed system, we use a MAST performance analysis tool [9] [20] with a proprietary model input format. Therefore, a meta-modeling tool for converting the MARTE system model to the MAST format is required. We have performed this conversion by using the *marte2mast* tool [19] and have extended the MARTE conversion [26] to support: (a) the latest Eclipse IDE Juno and its Papyrus modeling plugin, (b) the extended MARTE profile and (c) the MAST 1.4.0 scheduling analysis tool.

B. MAST

The MAST analysis tool provides a set of schedulability analysis methods resulting in identification of worst-case latencies, throughputs, blocking times and resource utilization. Furthermore, the MAST sensitivity analysis techniques enable predictions on the system robustness. In our approach, we deploy the following algorithms: Response Time Schedulability Analysis (RTA), Offset-Based Optimized RTA and Holistic RTA for fixed priorities, as well as the local and global EDF algorithms. For hierarchical scheduling, we deploy the varying priorities RTA, EDF Mono-processor RTA and EDF-within-priorities RTA.

C. JsimMAST

The *JsimMAST* [21] simulation tool is used for analysis of the architectural alternatives that have been pre-approved by the MAST tool. The *JsimMAST* simulation algorithms receive a pre-formatted MAST-2 system model as an input, and result in detailed task-execution timelines, buffer/bus/network-load timelines and task-interleaving/blocking aspects.

D. MARTE extension

Since the MARTE profile does not support the Network Drivers of the MAST tool, we have extended the MARTE profile with the new stereotype `SaNetwork` in the *Schedulability Analysis Modeling* package. Our ProMARTeS toolkit is available as an open source distribution [16].

VIII. CASE STUDY: AUTONOMOUSLY NAVIGATING ROBOT

A. Introduction

We have been requested to verify real-time requirements for an advanced setup of an autonomous robot with complex

navigation algorithms and propose an optimal architecture with respect to the latency of control loops and cost. We use this case study to validate our performance evaluation approach.

The autonomous robot control is normally composed of multiple hardware units: a robot with an embedded PC and a set of processing remote workstations (Fig. 3). The provided robot has a set of infrared laser sensors and a differential axis with two wheels (left and right). The data from infrared sensors is used to compute the map of the obstacles surrounding the robot. Based on the computed map, remote workstations suppose to send timely feedback signals to the robot for the wheel control, thereby imposing real-time requirements for specific tasks.

B. Component Selection

For the case study, we have selected a set of suitable components from a ROS [12] repository, such that the component set enables the functionality of the autonomous control loop. ROS is an open-source operating system for robot-based applications. It provides hardware abstraction, device drivers, message-passing and package management. The ROS is based on publish-subscribe architectural style, where each component subscribes for, or publishes a service wrapped into a *topic* type.

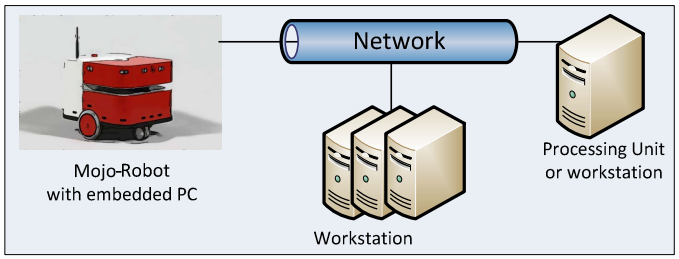

Fig. 3. Infrasrtucture for autonomous robot control

From the hardware point of view (Fig. 3), the system is fully distributed. The robot sends the infrared and odometry sensor data to a Processing Unit that facilitates wireless communication with the robot and tunnels the packets to remote Workstations that handle major computations. The computation results are transmitted back to the Processing unit to control the robot movements in real-time.

From software component point of view, the creation of surrounding map is the cornerstone of the navigation process, defining the reference model of the real world for further robot control. We have selected a `GM (Slam_Gmapping)` component for the map generation. The `GM` component instantly receives the laser and the odometry data and generates the actual 2D map.

To control the robot wheels, we have selected the `MB` (Move_Base) component. This component subscribes for the actual 2D map as well as for the laser/odometry data. It also computes the local and the global plan to provide a global strategy and to publish navigation control signals.

The `RVIZ` component visualizes the environment (2D map) to the control officer and allows setting the final destination goal for the robot. This goal is issued to the `MB` component for further wheel control. The `MD (Mojo_Driver)` component is responsible for the communication between the robot and ROS. The `MD` publishes the sensor data and subscribes to the wheel control data. The `MF (Mojo_Frame)` component publishes the 3D geometrical representation of the robot with the exact position of the infrared sensors in space.

```
SW component = MB
function = ExecuteCycles
   property = cpuUsage
   CPU = Intel(R) Core(TM) i5-2520M CPU@2.50GHz
     instructions_avg, min, max = 1.81985e+07
     cycles_avg, min, max = 1.364+07, 2.129+06, 2.183+07
     L2_miss_avg, min, max = 55068.9, 6592, 88475
     L3_miss_avg, min, max = 16698, 2200, 35827

   CPU = Intel(R) Core(TM) i5 CPU 750@2.67GHz
      instructions_avg, min, max = 1.93538e+07
      cycles_avg, min, max = 1.6003+07, 2.117+06, 2.4029+07
      L2_miss_avg, min, max = 39541.2, 6864, 69087
      L3_miss_avg, min, max = 6308.96, 391, 30626

   property = busUsage
   BUS = bus_Intel(R) Core(TM) i5-2520M CPU@2.50GHz
      L3_miss_avg, min, max = 16698, 2200, 35827
      L3_cache_wr_avg, min, max = 8960.51, 335, 15039

   BUS = bus_Intel(R) Core(TM) i5 CPU 750@2.67GHz
      L3_miss_avg, min, max = 6308.96, 391, 30626
      L3_cache_wr_avg, min, max = 7323.12, 246, 14497

   property = memoryUsage
         averageClaim, averageRelease = 966224
         minClaim, minRelease = 966224
         maxClaim, maxRelease = 966224
   END ExecuteCycles
```

Fig. 4. An example of a performance model for GM component

C. Component Profiling and Modeling

We have profiled the selected components for two different platforms: i5-750 CPU (1st generation) and i5-2520 Mobile CPU (2nd generation). The specifications of the HW platforms are summarized in Table I. During the profiling process, the ProMo tool automatically generates a performance model for each individual SW component. An example of the performance model of the `MB` component is depicted in Fig 4.

TABLE I. HW PLATFORMS USED FOR PROFILING

Computer HW						
CPU				***BUS***	***RAM***	
Model	Freq.	Cache	#cores/ #thrds	Trfr. rate	Size	Freq.
Intel Core i5 750	2.8 GHz	8MB	4/4	2.5 GT/s	4GB	1333 MHz /dual channel
Intel i5 2520	2.5 GHz	3MB	4/4	5.0 GT/s	8GB	1066 MHz /dual channel

D. System Composition phase

At the system composition phase, we have decided to use three different architecture alternatives varying in hardware topologies and SW/HW mapping (Fig. 5).

In Architecture A, we deploy 2 processing nodes. The most computationally expensive component, `GM`, is mapped on the i5-750 CPU node, while the rest of navigation components are mapped on the i5-2520 node. In Architecture B, all SW components are mapped on the single i5-750 CPU processing

node. Architecture C balances the workload of the system between two nodes: the i5-750 CPU processing node executes the GM and the MF components, while the i5-2520 CPU processing node executes the MD and the MB components.

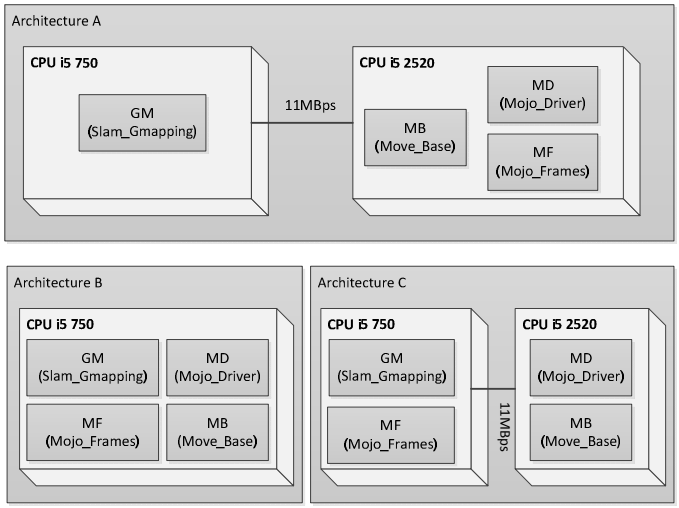

Fig. 5. Three initial architecture alternatives

E. Scenario Definition

The analysis of predicted system behaviour has shown that the workload on the system can be characterized by seven execution scenarios (Fig. 6). In this section, we describe the functionality, the trigger period and the deadline for each individual scenario. The triggering periods and real-time requirements (deadlines) for each scenario are presented in Table II. The deadlines are not scoped by periods and are defined to maintain the system's stability only, here the freshness of the data is not critical for each period.

This paragraph defines infrastructural scenarios with no hard real-time deadlines and is optional for reading. The scenario GM:Odom describes the robot odometry data transmission to the GM component. In this scenario, the odometry data is transmitted every 37 ms and the GM component stores these data to an internal buffer (memory). Similarly, during the scenario GM:Laser, the infrared sensor data is transmitted to the GM every 79 ms for internal storage. Commonly, the scenarios MB:Odom and MB:Laser are triggered every 37 ms and 79 ms, respectively transferring the odometry and the laser data from the robot node to the MB component for internal storage. The scenarios GM:Frame and MB:Frame describe transmission of the infrared sensor positions to the GM and MB components. The scenario is iterative with 50-ms period and a missing deadline does not lead to system failure.

The scenario GM:Map describes generation of the 2D map of the robot environment. The Map operation is the core of the GM component. The operation is triggered with a 1000-ms period. During iteration, Map loads the laser and odometry data from the internal buffer and updates/generates the actual 2D map. The completion deadline of the task instance is set to 1000 ms.

The scenario MB:Nav is the main scenario of interest, since it is computationally expensive and has a hard real-time deadline at a very low time span (150 ms). The scenario describes the feedback control loop for the robot wheels. The Nav operation of the MB component is triggered every 150 ms. It loads the odometry, laser and the actual 2D map data from the internal buffers, creates the global/local planners and sends the control signal for the engine of the robot wheels.

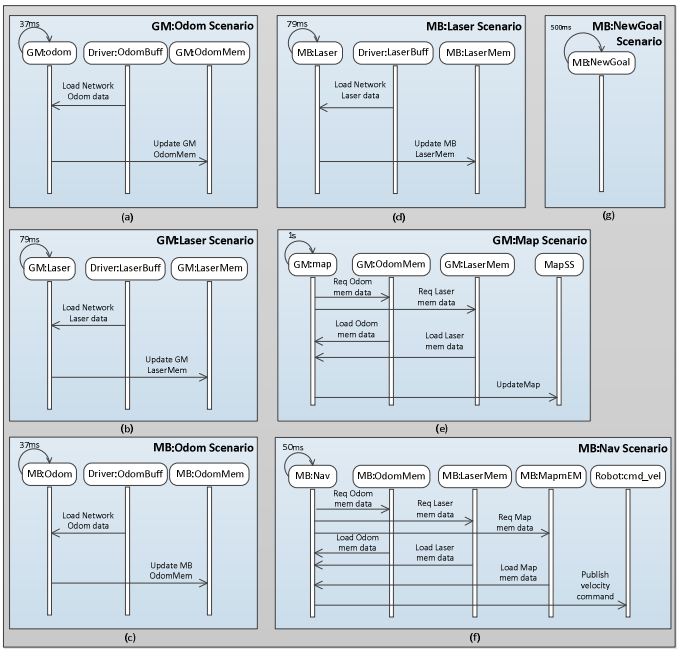

Fig. 6. Specification of scenarios in a message sequence chart diagram style

TABLE II. PERIODS AND DEADLINES FOR SCENARIOS

Scenario name	*Period (ms)*	*Deadline (ms)*
GM:Odom Scenario	37	1000
GM:Laser Scenario	79	1000
MB:Odom Scenario	37	1000
MB:Laser Scenario	79	1000
GM:Frame Scenario	50	1000
MB:NewGoal Scenario	500	500
MB:Frame Scenario	50	1000
GM:Map Scenario	1000	1000
MB:Nav Scenario	50	150

The scenario MB:NewGoal represents the workload on the system when a new goal arrives from RVIZ during the navigation process. For worst-case analysis we set it to be periodic with 500ms period.

The deadlines for the scenarios have been defined to guarantee that the autonomous navigation system: (a) avoids collisions with newly appeared obstacles and (b) does not introduce a noticeable response in delay in setting a new goal.

In this case study, we strived for identification of an architecture alternative able to satisfy the two hard real-time requirements at a low cost of materials.

F. Performance Evaluation phase

The evaluation of the three alternative system architectures has been performed using both schedulability analysis MAST and the simulation JsimMAST techniques. First, we perform schedulability analysis for all parallel tasks in the identified scenarios and check whether their real-time requirements are satisfied. The alternatives featuring all met deadlines were applied to a simulation analysis, providing detailed behavior predictions. Finally, we compare the predicted performance results of the alternatives and propose the optimal alternative with respect to the response latency in critical scenarios, robustness, resource load and cost. In the following paragraphs, we present the analysis results (see Table III).

TABLE III. PERFORMANCE EVALUATION

			Arch. A	*Arch. B*	*Arch. C*
CPU Utilization	**I5 2520**	Core 1	7.62%	-	48.74%
		Core 2	27.24%	-	2.62%
	I5 750	Core 1	53.93%	4.80%	1.67%
		Core 2	-	21.06%	44.44%
		Core 3	-	38.76%	-
		Core 4	-	93.35%	-
NETWORK utilization/slack			10.43%	-	10.39%
Events	GM: Odom Scenario		595.213 ms	915.925 ms	575.751 ms
	GM: Laser Scenario		604.806 ms	929.603 ms	576.429 ms
	MB: Odom Scenario		111.186 ms	19.726 ms	280.717 ms
	MB: Laser Scenario		112.963 ms	19.726 ms	280.967 ms
	GM: Frame Scenario		602.565 ms	1085.0 ms	576.023 ms
	MB:NewGoal		5.158 ms	176.959 ms	188.964 ms
	MB: Frame Scenario		113.653 ms	176.404 ms	280.775 ms
	GM: Map Scenario		609.131 ms	947.877 ms	577.534 ms
	MB: Nav Scenario		113.903 ms	18.635 ms	270.594 ms

The Architecture A meets all the real-time requirements specified in Table II. However, the worst-case response time of the task in the `MB:nav` scenario (113.9-ms) is close to the deadline. For this task, unexpected CPU overload may lead to a missing deadline and to an undesirable collision, making the alternative sensitive to higher load conditions (low robustness).

The Architecture B, where all the SW components are mapped on a single node, satisfies all hard real-time requirements (Table II and Table III). However, due to the mapping of all SW components on the same node, the CPU load is high. This influences the end-to-end execution time of the `GM:Map` scenario (947 ms with a 1000-ms deadline), thereby substantially reducing the robustness of the map generation task under overload conditions.

Architecture C fails to satisfy the hard real-time requirement (150-ms) for the `MB:Nav` scenario task, accounting to 270-ms. The introduced network delay increases the task latency resulting in the RT requirement failure.

In the optimal architecture selection phase, the reasoning is as follows. Both architectures A and B satisfy the real-time requirements, but have low robustness under overload conditions. The task in `MB:nav` is not sufficiently robust in Architecture A, while the task in `GM:Map` is not robust in Architecture B. The former task has more severe consequences when missing a deadline (potential collision), therefore this task has higher importance for deadline fulfillment. Therefore, the Architecture B is a first choice to be considered optimal among alternatives. Moreover, the Architecture B deploys only one processing node, reducing the system cost. Finally, deployment of more than one HW nodes (platform, network) increases the possibility of HW or communication failures. Therefore, the Architecture B has been advised to robot owners as an optimal solution among the alternatives.

G. Validation of Predictions on Implemented System

In order to validate our method, we have implemented the optimal Architecture B, then measured the latencies of the most critical scenarios, `MapScenario` and `NavScenario`, and compared them to the predicted latencies. For the worst-case, the predicted vs. actual latency deviations have shown to be within a 6% range, as presented in Table IV. In order to compute the actual worst-case latency of the two scenarios, we let the system navigate for more than 1 hour, while recording the latencies to log files. However, for the actual measurements, we cannot guarantee that the system has reached its worst-case latency.

TABLE IV. PERFORMANCE VALIDATION

Method Validation			
Model	WCET Prediction (ms)	WCET in reality (ms)	Accuracy %
GM: Map Scenario	947.877	899.123	95 %
MB: NavScenario	19.726	18.833	95 %

IX. CASE STUDY FINDINGS AND LESSONS LEARNED

During the schedulability analysis we have noticed that the mapping of tasks on CPU cores a plays a definitive role for the system performance. Two computationally expensive tasks mapped on the same core may introduce high task interleaving, therefore increasing WCET of both tasks. An advised strategy is to map the identified heavy tasks on separate CPU cores. This will reduce latencies and increase robustness for each task, even at the expense of under-use of each CPU core.

We have also observed that the architecture, where the critical tasks are data- and execution-independent from performance of all other tasks, is very robust under overload conditions. All our tasks in the scenarios execute independently of the success/failure of other tasks, by taking the data from internal buffers. Failure of neighboring task to store actual data to the internal buffer does not influence the critical task delay, but only decreases the operation quality.

Another interesting conclusion is that for low-latency tasks, the mapping of involved components on different nodes led to disproportionally high increase in latency due to the added communication delay. One example is the low-latency `MB:nav` task rendering 18-ms delay in single-node mapping and 113-

ms delay in multiple-node mapping case. In opposite, heavy but high-latency tasks improve on execution speed when being mapped on several nodes, e.g. the `GM:Map` task with 947-ms delay on a single node and 609-ms delay in two-node mapping.

The case study revealed a number of limitations of our approach. We were not able to analyze the internal memory-CPU bus usage and cache behaviour. For this case study, it was not critical, but for data-intensive systems it would be of high importance. Besides this, we noticed an influence of OS tasks on the latency values during the performance validation on implemented system, while these tasks were not taken into account during the analysis. Finally, the migration of the tasks over different cores within one processing node is extremely difficult to predict, since it is dynamically defined by an OS scheduler at run-time.

X. CONCLUSIONS

In this paper, we have proposed an accurate performance analysis method for real-time systems. We have evaluated and validated our method on a real-time autonomous navigation robot system. The method incorporates (a) profiling and modelling of SW components at cycle-accurate level (b) automated generation of system performance model from the models of individual components (c) evaluation of the obtained system model by schedulability analysis and simulation techniques, resulting in predicted latencies, throughput, resource usage and robustness. The presented component and system models are MARTE-compatible.

The method features multiple advantages for profiling, modeling and evaluating real-time systems. Firstly, the profiling provides cycle-accurate performance measurements collected directly by the PMU (Performance Monitor Unit) of the attached CPU. Secondly, the performance models are compatible with the commonly used UML-MARTE profile. Thirdly, the established pipeline generating models at different analysis phases automates the analysis process and carries the profiled low-level metrics of the components through all phases till the overall system performance is predicted. This brings high accuracy in predictions, as it was shown by the case-study (6% error range). Fourthly, the method deploys both types of techniques: schedulability analysis and simulation, enabling predictions of both guaranteed worst-case executions and detailed behaviour of tasks. Finally, we integrate the Eclipse Papyrus IDE into our tooling pipeline, so that an architect can easily design the SW/HW architectures graphically and automatically convert the design into models.

Our method has a number of *limitations* that require further research. Firstly, the component performance models can be obtained only for Linux-based operating systems, and the component profiling requires availability of actual HW platforms. Secondly, we do not provide reliable solution for generation of behaviour models for component operations, leaving this notorious task to a component developer. Thirdly, we do not fully take into account the influence of the memory, internal bus and cache behaviour on the performance of the system. Similarly, we do not provide support for GPU-based and OS tasks, which may decrease the prediction accuracy.

Let us briefly outline our future research steps. Since our performance analysis method does not incorporate the memory and the CPU cache structures, we plan to integrate a cycle-accurate platform simulator executing cache behaviour. Moreover, due to the increasing popularity of applications that can be executed on a GPU, it is vital to support the modeling and the evaluation of such systems. Last but not least, the architecture alternatives are composed manually, which bounds the DSE process to a limited number of alternatives. We are developing an architecture generation method to automate the creation and assessment of optimal alternatives.

References

[1] K. Triantafyllidis and E. Bondarev, "Low-Level Profiling and MARTE-Compatible Modeling of Software Components for Real-Time Systems," 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications, vol. 0, pp. 216-223, 2012.

[2] K. Klobedanz, C. Kuznik, A. Thuy and W. Mueller, "Timind Modeling and Analysis for AUTOSAR-Based Software Development - A Case Study," in EDAA, 2010.

[3] C. Silvano, W. Fornaciari, et al, "MULTICUBE: Multi-objective Design Space Exploration of Multi-core Architectures"," 2010.

[4] A. D. Pimentel, M. Thompson, et al, "Calibration of abstract performance models for system-level design space exploration," J. Signal Process. Syst., vol. 50, pp. 99--114, feb 2008.

[5] L. Thiele, "Performance analysis of distributed embedded systems," 2007.

[6] Bondarev et al., "CARAT: a toolkit for design and performance analysis of component-based embedded systems," 2007.

[7] V. Cortellessa, et al., "Integrating Software Models and Platform Models for Performance Analysis", vol. 33, pp. 385-401, 2007.

[8] E. Bondarev, M. Chaudron and P. de With, "A process for resolving performance trade-offs in component-based architectures," 2006.

[9] M. Gonzalez Harbour, et al., "MAST: Modeling and analysis suite for real time applications," in Real-Time Systems, 13th Euromicro Conference on, 2001, 2001.

[10] "UML-RT Profile for Real-Time Systems," [Online]. Available: http://www.omg.org/news/meetings/workshops/RT_2002_Workshop_Presentations/02-2_Watson_RT-UML.Tutorial.pdf.

[11] "SysML," [Online]. Available: http://www.omgsysml.org/.

[12] "ROS" [Online]. Available: http://www.ros.org/wiki/.

[13] "Robocop Components," [Online]. Available: http://www.hitech-projects.com/euprojects/robocop/.

[14] "Rationale Rose: A modeling environment," [Online]. Available: http://www-01.ibm.com/software/rational/.

[15] "Ptolemy II: Modeling, Simulating and Designing RT Systems tool," [Online]. Available: http://ptolemy.berkeley.edu/ptolemyII/.

[16] "ProMARTES: Profiling, Modeling,Analysis of RT Embedded System," [Online]. Available: http://vca.ele.tue.nl/demos/ProMARTES.

[17] Maatta S., "Model Based Approach for Heterogeneous Application Modeling for Real Time Embedded Systems".

[18] "MAST: The Performance Analysis Suite for Real-Time Systems," [Online]. Available: http://mast.unican.es/.

[19] "Marte2MAST metamodeling converter tool," [Online]. Available: http://mast.unican.es/umlmast/marte2mast/.

[20] "MARTE," [Online]. Available: http://www.omgmarte.org/.

[21] "JsimMAST: The Performance Analysis Simulator for RT systems," [Online]. Available: http://mast.unican.es/jsimmast/index.html.

[22] "AUTOSAR: Automotive System Architecture " [Online]. Available: http://www.autosar.org/download/papersandpresentations/AUTOSAR_Brochure_EN.pdf.

[23] "AADL," [Online]. Available: http://www.aadl.info/aadl/currentsite