ProMARTES: Accurate network and computation delay prediction for component-based distributed systems

Konstantinos Triantafyllidis*, Waqar Aslam, Egor Bondarev, Johan J. Lukkien, Peter H.N. de With

Technische Universiteit Eindhoven, Postbus 513 5600 MB EINDHOVEN, Netherlands

1. Introduction

The development of component-based real-time distributed systems (CB-RTDS) has become an adopted practice, since it enables rapid system prototyping at the early design phase, based on existing HW and SW components. Moreover, the composed system should meet the predefined performance requirements such as throughput, latency and robustness. For this rapid prototyping, reliable assessment methods are required, evaluating and predicting the performance of the composed system at the early design phase. Incorrect performance predictions may lead to deployment of an inefficient system architecture resulting in iterative system re-design or re-implementation. Furthermore, due to the large population of architecture alternatives that need to be assessed, the performance prediction method should provide a fast system performance evaluation.

In order to avoid the consequences of system re-design or re-implementation, as well as to achieve a rapid performance evaluation, in this paper we concentrate on the development of a performance prediction method, that is sufficiently accurate, and yet fast in terms of processing time. For such a method supporting CB-RTDS, various aspects need to be addressed, which are expressed in the following three research questions.

RQ1. When establishing a reliable assessment method for CB-RTDS, both the computation and the network delay need to be taken into account. The majority of the performance analysis tools for CB-RTDS are based on simplistic computation and communication models, thereby focusing more on the scheduling and synchronization schemes and less on the accuracy of the computation and communication delay. Both computation and communication delays are difficult to analyze and predict simultaneously, due to their following intrinsic properties. In order to accurately predict computation delays, low-level performance metrics (execution cycles, cache misses, memory accesses) mapped
on specific HW nodes need to be taken into account. Moreover, possible conflict of processes/operations, competing to access the same processing resources, influence the computation delay, which makes the prediction even more arduous. The communication delays are influenced by the network media (wired, wireless), the incorporated network protocols and the load on the network.

The computation and communication delays are strongly dependent on the CB-RTDS. The exact triggering moment of an operation influences the network congestion, indicating that the computation delays affect the network delays and vice versa, establishing a mutual dependency. Both computation and communication feature joint intrinsic properties while having a mutual dependency, which makes their integration in one performance analysis method a challenging task. Based on these observations, we formulate our first research question:

“How to integrate both computation and network load and delays in one performance analysis method with the aim to increase the performance prediction accuracy?”

RQ2. The performance analysis methods can be classified in two main categories: analytical (formal) methods and simulation techniques. Analytical methods provide reliable WCET, but do not output detailed execution timeline data. On the other hand, simulation techniques are not able to provide a guaranteed WCET, but they feature timeline data and Average Case Execution Time (ACET) performance metrics. Moreover, analytical performance analysis methods can be executed within a few seconds due to their formal substance, while the simulation techniques may require substantial time span (from minutes to days) to converge. Ideally, a performance analysis method should provide a guaranteed WCET, alongside with detailed execution timeline data and ACET. A majority of the performance analysis methods focus either on formal or on simulation-based analysis techniques, while a minority supports both. However, applying both types of analysis simultaneously in each architecture alternative requires substantial processing time, leading to the research question:

“How to obtain guaranteed WCET, ACET and execution timeline data, while limiting the performance analysis processing time?”

RQ3. For accurate and reliable performance analysis, explicit and detailed SW and HW component models are indispensable. The performance models should support the intrinsic data of the HW usage, such as cycle-accurate processor allocation, cache load, bus usage, network and memory load.

Such cycle-accurate SW component models act as a reliable source for the performance analysis of the system. However, the exploitation of these highly accurate metrics throughout the performance analysis phases is not an easy task. The analysis of the cache usage is difficult to be incorporated for the system-wide analysis, since the cache content changes at high rate and depends on other SW components executing on the core. Moreover, the internal HW structure of the CPU, such as the cache hierarchy (number of levels, size) and bus frequency (communication between CPU and RAM) influence the computation delays, leading to the research question:

“How to propagate the obtained cycle-accurate component performance properties through all composition/analysis phases?”

In this paper, we present a cycle-accurate performance analysis method for CB-RTDS (ProMARTES). Our analysis method consists of six individual phases: (a) profiling SW components at high-accuracy level, (b) modeling the obtained performance measurements in MARTE-compatible resource models, (c) composition of the system architecture, (d) performance analysis (scheduling, simulation and network analysis) of a system model, (e) evaluation of the obtained performance predictions compared to the system requirements and (f) a subsequent architecture improvement.

The profiling, modeling, composition and part of the performance analysis phases have been discussed in detail in our previous work (Triantafyllidis et al., 2012, 2013). In this paper, we mainly focus on the network analysis prediction model and its integration in the overall Design Space Exploration (DSE) framework. In our previous work we have focused on the tool implementation and the advantages that it features rather than the modeling of the system. Therefore, in this paper we present part of the previous work enriched with the related models. These related models are used through the individual phases resulting to the final system model, that is applied to performance analysis.

In this paper, we propose a conservative method for combining the computation and communication delays within the same system model. This method is supported by both scheduling and simulation analysis. The former provides a guaranteed WCET, while the latter enables the identification of possible bottlenecks and delivers detailed time-line data. Both types of analysis use cycle-accurate performance metrics, which are obtained during the profiling phase. These cycle-accurate performance metrics are translated into precise execution time models. Our models not only involve computing performance analysis, but also the networking aspect of distributed system architectures. It will be shown that the analysis methods can be applied in a specific way so that they can be exploited in parallel to achieve a faster selection of an architecture operating within time boundaries with a known WCET.

The most important contributions of this paper are: (a) the ProMARTES toolkit, which has been updated compared to the existing version (additional network analysis tools, metamodel transformations, profiling and resource modeling including data transmission over the network), in order to support both processing and communication resource-load and delays, (b) the modeling scheme adopted for the integration of the cycle-accurate resource-processing attributes alongside with the data transmitted over the network, and (c) the combination of both scheduling and simulation analysis techniques (processing and communication) for obtaining multiple performance analysis metrics for the assessment of a component-based real-time distributed system at the early composition phase. The developed ProMARTES performance analysis toolkit is open-source and available at Triantafyllidis (2013).

The sequel of this paper is structured as follows. Section 2 presents related work. Section 3 describes the proposed performance prediction framework. Section 3.1 presents the profiling and modeling approach adopted by ProMARTES. Section 3.2 focuses on the architecture composition phase and Section 3.3 gives a detailed description of the system model analysis from ProMARTES. Section 4 illustrates a case study used for the validation of the ProMARTES framework, while Section 5 presents the findings from that case study. Section 6 presents the applicability of the ProMARTES framework to different system types. Finally, Section 7 concludes on the benefits and the limitations of the approach.

2. Related work

In the last decade, research work resulted in several innovative methods for addressing the problems of SW/HW component modeling, predictable assembly and evaluation of CB-RTDS. Currently, a wide variety of modeling profiles are available for composition of real-time systems: PCM, SysML, UML-RT, MARTE and AADL. Each of the modeling approaches show advantages and limitations depending on the type of the composed system, the type of the analysis and the supported tools as presented in Triantafyllidis et al. (2012) and Triantafyllidis et al. (2013).
Numerous composition and evaluation approaches have been proposed based on the different modeling profiles. Regarding the modeling of systems, a few approaches focus on specific real-time systems and other cover broader systems types. The composed systems are analyzed by either simulation-based, or analytic, or both types of analysis techniques. However, the majority of the approaches consider simplistic network analysis models, while they focus more on the computation delays and scheduling policies. In the sequel, several interesting modeling and performance analysis approaches.

The UML modeling profile is the cornerstone of many well-established performance analysis methods. Bertolino and Mirandola (2004) proposed the performance analysis method CB-SPE. The components are modeled based on the UML-SPT profile, while the performance models are based on queuing networks. Another analysis method based on UML and queuing networks was proposed by Liu et al. (2005), called NICTA. Unlike the CB-SPE approach, NICTA does not distinguish the roles between the component developer and the architect of the system. Moreover, this approach is applicable only for EJB applications.

In the real-time embedded systems domain Aleti A. et al. introduced the ArcheOpterix (Aleti et al., 2009), which is an extendable tool for architecture optimization of component-based systems. The ArcheOpterix is based on the AADL modeling language and analyzes the performance of the composed system by taking into account the execution delays between the components. Additionally, it offers an optimization plugin for automated generation of architecture alternatives and identification of the optimal system architecture for different design objectives. However, the lack of cycle-accurate modeling primitives (AADL) may decrease the performance prediction accuracy of newly composed system.

A more general modeling and performance analysis approach is proposed by Cortellessa et al. (2007), which is based on UML-RT (Graf et al., 2006). The analysis is performed by a proprietary simulator RRT (IBM, 2003), limiting a broad applicability.

Since that the previous approach is based on simulation analysis, guaranteed worst-case boundaries cannot be predicted. Moreover, simulation may consume substantial time until the performance prediction metrics converge.

Hissam S. et al. proposed PECT (Hissam et al., 2003), which supports both scheduling and simulation analysis techniques. PECT delivers guaranteed WCET due to the formal substance of the analysis and ACET due to the support of simulation-based techniques. It is based on the component composition language (CCL) and supports synchronous and asynchronous communication, behavioral models as well as composite components. Focusing on the distributed system domain, Wu and Woodside (2004) have proposed the CBML method. The CBML comprises an extension of the layered queuing networks and, similar to PECT, it enables synchronous and asynchronous calls among the components. Both PECT and CBML approaches provide performance analysis, either by formal or by simulation techniques.

As we have already mentioned, abundance of different system modeling approaches (AADL, UML, UML-SPT, UML-RT, MARTE, PCM, etc.) have been adopted by the system architects, the performance engineers and the research community. Each different modeling approach applies a transformation from system model substance to a performance model which is then used for the system performance analysis. A number of different performance models are used and developed by the research community, such as Queuing Networks, Stochastic Petri Nets, Simulation models, etc. For the performance analysis of a system, the appropriateness of a performance model depends on the application area and the specification of the system. Therefore, as soon as the system architect concludes to the best-fitted performance model for the evaluation of the system, a meta-model transformation method needs to be applied for the translation from system model representation to performance model(s). On the way to support multiple types of performance models from one system model, Woodside et al. proposed the PUMA method (Woodside et al., 2014), for translation of system models based on UML-MARTE profile to various performance models. Therefore, PUMA is able to enable the applicability of different analysis approaches on predefined system models when different performance prediction metrics or prediction accuracy are required. However, this approach considers simplistic resource models (delay annotations) for both the computation and communication resources. Similarly to this approach, Tribastone et al. (2010) convert system models based on UML4SOA and MARTE to Layered Queuing Network (LQN) performance models.

Based on the CBML and PUMA approach and dealing with the performance prediction of cloud storage systems, Woodside et al. (Faisal et al., 2013) examined the influence of the network delay over the overall system performance. The system modeling is based on LQN and each task is associated with a static execution time/delay. The network model does not take into account the lower OSI layers, possible conflicts and retransmissions (network overload), but assumes a static network delay for specific network connections. Moreover, the network model does not consider wireless communication links, therefore limiting the applicability of this approach to wired connections. On the other hand, it has to be mentioned that the author focuses on cloud storage systems, where the links among the different hops is mainly achieved through wired connections.

A few system modeling and performance analysis approaches, instead of developing their own solvers for the performance analysis of the composed system, focus on system model transformation from proprietary system models such as UML, MARTE, PCM etc. to mathematic representation models, such as Petri Nets, Markov chains and Layered Queuing Networks. These approaches benefit from the existence of highly accurate performance and reliability analysis tools for obtaining the performance predictions. Methods that belong to this group are: the Palladio and KLAPER frameworks and the metamodel conversions proposed by Tawhid and Pietri (2008) and Tribastone et al. (2010).

Becker et al. proposed the Palladio framework (Becker et al., 2007) which incorporates its proprietary metamodel, the PCM (Becker et al., 2009). The PCM instantiations are transformed to specific performance and reliability models (EQN, LQN, QPN, SPA) supporting multiple analysis types (analytic, simulation and hybrid) as presented in Brosig et al. (2015), enabling different levels of prediction accuracy and analysis processing delay. Due to the transformation from PCM to Markov chains, Palladio delivers in addition to performance, reliability analysis as presented in Brosch et al. (2012). The supported scheduling processing policies by the Layered Queuing Network Solver (LQNS) (Carleton) are the Processor Sharing (PS), First-In-First-Out (FIFO), Priority Preemptive Resume (PPR) and random scheduling. However, it does not support Fixed Priority (FP) or Earliest Deadline First (EDF) scheduling policies, which are widely used in the scheduling of real-time systems. Giancone et al. (2011) developed a simulation-based performance analysis method, the KLAPER. This method, similarly to the Palladio framework, enables transformation from proprietary component structure (UML, OWL-S) to desired performance models expressed either as Markov chains or as queuing network enabling both reliability and performance analysis of component based system model. The performance analysis is performed by Layered Queuing Network Solver, thus enabling both analytic (LQNS) and simulation-based (LQSim) analysis. The resource models are based on execution time delays and do not take into account intrinsic properties of the processing units, reducing the performance analysis prediction accuracy. For the communication among the processing resources (network), both Palladio and KLAPER consider
static fixed delays, while not modeling the lower OSI layers, which are more specific to the communication technology under consideration. According to our knowledge, both tools do not consider any automated method for the profiling and the composition of the resource allocation models and follow the generation of the performance models. This notorious task is being performed manually by the component engineer, first by profiling and then by composing the performance models of the SW and the HW components.

Distributed systems inherently rely on underlying network connections. All the aforementioned performance analysis methods either do not support distributed systems, or the network scheme is based on a simplistic model. Similar to all above-mentioned approaches, Gonzalez et al. (2001) assumes a simplistic model for the network analysis delay of Ethernet. Unfortunately, the associated tool does not guide how network settings affect the delays and performance. Additionally, this tool does not provide packet delay for WLANs.

Existing models for delay performance in WLANs mostly focus on saturated WLANs (Bianchi, 2000; Cai et al., 2006; Tinnirello and Bianchi, 2010), in which the underlying assumption is the guarantee of always-available packets. A seminal work presents an intuitive expression using the Little’s law (Bianchi and Tinnirello, 2005), but again for the same condition of saturated MAC queues. An analysis for throughput and delay is derived by forming a gaming problem in Nguyen et al. (2012). In Jamali et al. (2014), a delay expression is derived in context of a throughput optimization problem. A case of a central entity i.e. an access point based WLAN is analysed for delay derivation in the uplink and downlink directions (Kuo and Tsai, 2013) at the cost of modification of the standard backoff algorithm (IEEE, 2007). The delay performance in access point based saturated networks does not suffice in our case, which has periodic traffic that is generated synchronously amongst the nodes, which are connected by direct wireless links.

Our case study requires synchronization of transmission epochs amongst the nodes, occasionally giving increased delays due to collisions in hyperperiods. Due to these limitations, we will develop later a delay predictor for the best-, average- and worst-cases.

In CBSE the SW component models are shipped alongside with their relative performance models, developed by the component engineers. The relative diversity of execution platforms requires the composition of different resource allocation models (performance models) for each different platform. Additionally, the performance of the majority of the components is strongly dependent with the data used as input. Therefore, Tuma et al. (Bulej et al., 2012) have proposed SPL (Stochastic Performance Logic), a mathematical formalism for expressing and evaluating performance requirements avoiding platform dependency. The performance models are described as a unimodal distribution of execution times for specified workload. SPL allows the comparison of the performance models of relative methods. Unfortunately, the profiling method (at least for the presented case study) is based on high-level performance metrics (delay in time spectrum). Low-level performance metrics such as cycles and instructions retired, cache misses and bus load are hidden under the execution time, reducing the accuracy of the resource models. However, the SPL method is a general approach and could be possibly applied for low-level performance metrics as soon as the required SPL formulas are updated to the new models.

In the domain of pure model-based techniques, Bondarev et al. (2007, 2006) have proposed a solution for design and performance analysis of conventional CBSE embedded real-time systems (ROBO-COP components ITEA, 2000). The approach is supported by the CARAT toolkit, which synthesizes SW/HW component models and then constructs a system model with corresponding scenario models. Afterwards, it 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 contrast with the above-described simulation- and scheduling-based techniques, Wandeler et al. (2006) have presented a compositional method, incorporating simulation-based analysis 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 worst-case resource load and latencies. However, intrinsic cycle-accurate execution properties cannot be incorporated during the analysis and moreover, task execution timeline and interleaving aspects cannot be obtained with this technique.

Table 1 presents the specifications, in terms of modeling and performance analysis, of the state-of-the-art system composition and performance analysis frameworks and illustrates a comparison against the ProMARTES framework. All the current composition and performance analysis frameworks consider simplistic resource models based on execution and communication delays. Moreover,
the generation of these resource component models is being performed manually by the SW component engineer, requiring substantial effort and time for covering multiple existing SW and HW component instances. Regarding the performance analysis, a number of frameworks support simulation or analytic-based or both types of performance analysis techniques. However, all of them lack in performance analysis cycle-accuracy due to the absence of cycle-accurate resource and performance models. Moreover, most of the frameworks deal with network delays and they do not consider additional communication delays due to packet retransmissions (collisions). Additionally, they do not support wireless network links and various communication protocols limiting their application to wired communication links while they support limited communication protocols.

This paper proposes the ProMARTES framework that targets on the overall design space exploration of component-based real-time distributed systems. It features a semi-automated generation of cycle-accurate resource models, aiming at increase of the overall performance prediction accuracy. The system architecture composition is performed graphically, enabling rapid system prototyping. The performance analysis of the virtual prototype is achieved through both analytic and simulation-based analysis techniques. Both analysis techniques integrate a highly detailed network model that supports wired and wireless communication links among the nodes and several communication protocols. The approach is evaluated for the composition and the performance analysis of a distributed multi-robot system. Finally, the optimal selection of four architecture alternatives is presented with respect to the resource utilization, the end-to-end execution delays for several individual scenarios and the robustness of the system under overload conditions.

3. Performance prediction framework

The CB-RTDS performance analysis is an integral part of the overall Design Space Exploration (DSE) in our work through the last decade (Triantafyllidis et al., 2012; Bondarev et al., 2007, 2006). In order to support a rapid and accurate performance prediction of real-time architectures, we have developed a framework, called ProMARTES, which integrates three individual phases: (1) Component Profiling and Modeling, (2) Architecture Composition, (3) Architecture Performance Evaluation (see Fig. 1). Let us provide a brief description of these phases.

Component profiling and modeling. The component developer profiles the SW components at cycle-accurate instruction level, generating a performance model for each individual component. ProMARTES provides all the profiling and modeling primitives for this phase. Each performance model addresses various HW resources aspects (CPU, BUS, RAM, Network, etc.) and can be specified for multiple platforms. Moreover, the generated performance models are joined into the repository for using them during the subsequent Performance Analysis phase.

Architecture composition. This phase aims at selection of required components (based on functional requirements), guided architecture composition and 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 specifies component instantiations and connections, HW and network topology, as well as the SW component mapping on the HW platform. The critical execution scenarios are specified by the architect and combined with an alternative, thereby defining a system model. Other challenges addressed in this phase are the support for multiple component architectural styles and provisioning of resulting composition models in common formats (MARTE, UML, etc.).

Architecture performance evaluation and optimization. The generated system models are applied for a two-fold performance analysis, based on both scheduling and simulation. Both types of performance analysis support various HW platforms, multiple scheduling policies and different network protocols. They yield accurate prediction of performance properties such as latency, hardware usage and throughput. The obtained performance results are validated with respect to the system requirements.

For the architecture definition of the system, the modelling approach proposed by Barros et al. (2013) has been adopted. In this approach, the system model is based on the resource-reservation paradigm, enabling both virtual and physical platform.

The performance validation may lead to a sequence of 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). Then it finally converges to an optimal architecture alternative that meets the system performance requirements, which fulfill other criteria defined by the stakeholders (robustness, cost etc.). The following subsections detail each phase.

3.1. Phase 1: component profiling and modeling

At the component development stage, the component developer profiles each individual SW component using the ProMo tool from Triantafyllidis et al. (2012). The ProMo tool offers a uniform method for profiling the individual SW components at cycle-accurate level and automatically generates resource models based on the obtained profiling measurements. The ProMo tool profiles
every function of a provided interface of a component and provides the following metrics: (a) Instructions and Cycles used, (b) Cache-miss-rate for L2 and L3 cache misses, (c) amount of memory claimed and released and (d) bus load due to read/write instructions. The metrics are further integrated into a component performance model.

The structural view of the model is depicted in Fig. 2. Each Component Performance Model (CPM) contains a number of provided interfaces, implemented functions and each implemented function embeds certain performance attributes.

Hence, a CPM is modeled into $M_{CP}$ in the form of a set $I$:  
$$ I = M_{CP}, \quad \mathcal{J} = \{I_1, I_2, \ldots, I_n\}. $$  
(1)

Each individual interface $I_m$ provides a number $i$ of implemented functions $F_{mk}, k \in [1,i]$:  
$$ F_m = \{F_{m1}, F_{m2}, \ldots, F_{mi}\}, $$  
(2)

where $F_m$ specifies an interface $I_m \in \mathcal{J}$.

Each implemented function $F_{mk}, k \in [1,n]$ defines a set of resource-usage attributes $RU$. The resource utilization attributes $RU$ are grouped as $\{cpuUsg, memUsg, busUsg, netUsg\}$.

$$ RU = \{cpuUsg, memUsg, busUsg, netUsg\}. $$  
(3)

The SW components can be profiled on different HW platforms, defining $j$ sets of resource utilization attributes $RU$, which are associated with the type of the hardware IP (Intelligent Property) block used in each HW platform and for each attribute we specify

$$ \text{cpuUsg} = \{\text{cpuUsg}_1, \text{cpuUsg}_2, \ldots, \text{cpuUsg}_j\}, $$  
(4)

$$ \forall \text{cpuUsg}_k, \text{cpuUsg}_k = \{\text{Instr}, \text{Cycles}, L1, L2, L3\}, $$  

$$ \text{memUsg} = \{\text{memUsg}_1, \text{memUsg}_2, \ldots, \text{memUsg}_j\}, $$  
(5)

$$ \forall \text{memUsg}_k, \text{memUsg}_k = \{\text{VM}, \text{PM}, \ldots, \text{PgS}\}. $$

The ProMo performance model from Fig. 2 is further transformed to the MARTE compatible resource model by the ProMo2Marte meta-modeling tool. The ProMo2Marte converts the ProMo resource models into MARTE compatible SW/HW instantiations (see Fig. 3). The generated MARTE compatible resource models are then placed into an open repository for seamless reuse during the second phase Architecture Composition. The benefits of the ProMo profiling method are manifold and are in details presented in Triantafyllidis et al. (2012).

**Behavior model** As we have already mentioned, each SW component features a number of interfaces that feature a number of implemented operations. The interfaces of a SW component are distinguished in two main categories: the provided and required SW component interfaces. The relationship (provided-required interfaces) among the SW components defines the behavior models of the SW components, which are used during the composition phase, enabling semi-automated architecture composition phase, as we detail in the following Section 3.2.

### 3.2 Phase 2: architecture composition

The obtained MARTE-compatible SW and HW components resource models are utilized by an architect during the Architecture Composition phase. At this phase, an architect selects SW components which may satisfy functional requirements. Subsequently, the architect specifies the component composition by first instantiating the selected components and then binding these instantiations. The binding of the instantiations reflects the SW and the HW architecture of the composed system. Finally, the architect specifies the critical execution scenarios $S_n$, by defining the internal or external triggers for the system and the functions that are invoked by those triggers:

$$ S_n = \{T, Ttype, D, Dtype, f_1, f_2, \ldots, f_k\}. $$  
(9)

where $P$ is the triggering period, $Ttype$ is the triggering type (sporadic, periodic, ...), $D$ is the scenario deadline, $Dtype$ is the type of the deadline (hard, firm, soft) and the functions $f_k, k \in [1..K]$ are the scenario-involved functions, derived from the instantiated SW components. A set of the individual scenarios $S_n$ defines the scenario set $S$:

$$ S = \{S_1, S_2, \ldots, S_n\}. $$  
(10)

The aforementioned design activities are supported graphically by the toolkit.
In our previous work (Bondarev et al., 2007, 2006), we have presented an architecture composition process that allows scenario definition with minimal user intervention. For each scenario the architect of the system selects the SW components that satisfy the functional and extra-functional requirements. Then, he binds the SW component interfaces and he defines the trigger types and periods and possible scenario deadlines. Based on this approach and on the behavioral models presented in Section 3.1, we use the provided and required interfaces of the SW components for automating the architecture composition phase.

In the current ProMARTES implementation, we have developed an eclipse plugin that enables automated tracing of the provided and required SW component interfaces that are involved in the execution scenario $S_n$. This plugin filters out interfaces that cannot be combined among the involved scenario SW component instances, and proposes to the architect only interfaces and the related operations that are composable through the required-provided interface relationship. As long as the architect chooses the execution interfaces, he defines the fired operations which deliver the required functionality. In our case study Section 4, each interface defines a specific functionality, thus firing specific operations. This means that by defining only the interfaces of the instantiated SW components, the ProMARTES composition module is able to perform the binding among the different SW components by even specifying automatically the operations to be executed, reducing further the required workload for the system architect. Fig. 4 depicts a pseudocode diagram that integrates instantiated components and their provided and required interfaces which integrate a number of different operations. Depending on the involved instantiated components the developed plugin allows or not the connection among interfaces and the selection of the fired operations. Moreover, for each execution scenario (activity diagram), the architect defines the internal or external triggers for the system and the operations that are invoked by those triggers.

The execution scenarios are represented by using Activity Diagrams, as shown in Fig. 4. The scenario definition and the hardware architecture specification are separate tasks, so that they can be performed in parallel (right side of the Fig. 4). The architect selects hardware components from a repository and chooses a specific topology, number of processing nodes, communication protocols and scheduling policies defining the hardware platform $H_p$:

$$H_p = \{\text{CPU, Net, Task}\}.$$ (11)
where CPU is the set of the incorporated processing nodes, Net defines the network protocol and topology and Task defines the scheduling scheme for the components and their functions, specified by
\[
\text{CPU} = \{\text{CPU}_1, \text{CPU}_2, \ldots, \text{CPU}_N\},
\]
\[
\text{CPU}_n = \{\text{type, frequency, cores, cache, bus}\},
\]
\[
\text{Net} = \{\text{Net}_1, \text{Net}_2, \ldots, \text{Net}_N\}, \text{Net}_n = \{\text{protocol, topology}\},
\]
\[
\text{Task} = \{\text{Task}_1, \text{Task}_2, \ldots, \text{Task}_N\},
\]
\[
\text{Task}_n = \{\text{cpuHost, sPolicy, priority}\}.
\]

Naturally, the amount \( N \) varies in size for each attribute, but we have skipped further subscript notation for simplicity. Once the software and hardware architecture are specified, the architect maps the software components on the hardware nodes. The mapping defines on which processing node each software component has to be executed. For each scenario \( S_n \in S \), each function \( f_k \in S_n \) is mapped on a specific task \( \text{Task}_n \in \text{Task} \). The SW/HW mapping can be performed manually by connecting each SW component instance to the desired HW processing unit, as depicted in the pseudo activity diagram in Fig. 4. Finally, for each scenario \( S_n \) all the SW component instances will be mapped on a HW processing unit therefore specifying the execution path. When data-dependent SW component instances involved in the same scenario are mapped on different HW nodes, the communication link between the SW instances is automatically carried out by the network infrastructure. In any other case the communication is being performed internally by the execution node itself, thus limiting the data synchronization delays. It has to be mentioned that the SW/HW mapping, as well as the HW platform composition are activities that are performed by the system architect for the composition of a system that potentially meets the system requirements. These two activities can be automatically performed during an automated optimization process, as we have presented in Triantafylidis et al. (2015), thereby limiting the architecture composition effort to the system scenario definition by a system architect.

When the SW component is mapped on a specific HW resource, the corresponding component performance model \( P_t \) is loaded complying with Eq. (1). In Fig. 4 it is depicted the mapping of the instantiated in the scenario SW components (left side) onto the HW platform defined in a class diagram (right side of the figure). The performance attributes of these performance models are then translated into execution time models (resource reservation models Barros et al., 2013), thereby preparing for further performance analysis. As execution time \( et \), we define the amount of time that a task needs for execution in seconds when mapped on a specific CPU.

On the way to address the RQ3 and based on the fundamental processor performance equation:
\[
\text{CPU Time} = \frac{\text{CPU Clock Cycles} \times \text{Clock Cycle Time}}{\text{CPU Clock Rate}},
\]
\[
\text{Clock Cycle Time} = \frac{1}{\text{Clock Rate}},
\]

presented by Patterson D. in Patterson and Hennessy (1990), we define the execution time equation:
\[
et_k = \frac{\text{cycles}_k}{\text{freq}_k}, \text{cycles}_k \in \text{cpuUsG}_k, \text{freq}_k \in \text{CPU}_k.
\]

In Eq. (16), the execution time \( et \) of each implemented function \( f_k \in F \) is derived by the division of the cycles \( \text{cycles}_k \in \text{cpuUsG}_k \) required for this function by the mapped CPU \( k \), with the frequency \( \text{freq}_k \) of the mapped CPU, where the \( \text{freq}_k \) is the Clock Rate (Eq. 15) of the attached CPU and \( \text{cycles}_k \) represents the CPU Clock Cycles of the aforementioned processor performance equation.

It has to be mentioned that the \( \text{cycles}_k \) metric embeds the latency introduced by the potential cache misses in all the cache hierarchy levels (L1, L2, L3), composing three values (best, average, minimum) presented in Eq. (8), which is further updated to
\[
et_k^x = \left\{ \begin{array}{ll}
\text{cycles}_k^\text{min}, & x = \text{min} \\
\text{cycles}_k^\text{avg}, & x = \text{avg} \\
\text{cycles}_k^\text{max}, & x = \text{max}
\end{array} \right.
\]

The scenarios \( S_n \) are then translated to \( S'_n \), where the resource utilization metrics of the incorporated functions \( f_u \) are replaced by execution time metrics:
\[
S'_n = \{T, \text{Type}, D, \text{Dtype}, f'_1, f'_2, \ldots, f'_N\}, f'_u = \{et, nd\}, u \in \{1..N\}.
\]

It is a set of the pure execution time \( et \) of the function \( f_u \) on the mapped HW platform (cpu, bus, memory) and \( nd \) is the network data to be transmitted by the \( f_u \) to the network.

The creation of a system model \( SM \) is based on the performance models of the involved components, the scenario models and the SW/HW mapping architecture:
\[
S \oplus H \models SM.
\]

where the operator \( \oplus \) denotes the mapping notation.

The resulting system model \( SM \) represents an executable structure that can be analysed for performance:
\[
SM = \{S'_1, S'_2, \ldots, S'_N\} \oplus \text{Net} \oplus \text{Task}
\]

It should be noted that an efficient SW/HW mapping is highly desired, since it distributes the load on the hardware resources in an optimal and balanced way. However, at the first mapping iteration, it is not clear for an architect how to deploy the software components such that the optimal load distribution is achieved. Various mapping alternatives are possible at this stage, where each alternative represents a possible variation of a system architecture.

We have developed an automated performance analysis mechanism, in order to help an architect to carefully evaluate the different alternatives and their consequences.

3.3. Phase 3: system model analysis

The system model \( SM \), obtained from the Architecture Composition phase, is further analyzed for its performance and robustness. In order to cover the requirements that RQ2 imposes, we combine two types of analysis: schedulability and simulation-based.

Schedulability analysis enables prediction of the best- and worst-case response latencies for each task instance associated with a real-time deadline. This formal analysis provides guaranteed worst-case boundary conditions and can be executed within a few seconds. However, due its formal substance, it does not provide the detailed behavior time-line data and realistic average-case latencies. Therefore, in order to obtain a more valid and clear insight on the system performance, we also apply simulation-based analysis techniques.

A simulation-based analysis is able to identify the average-case execution times, as well as the possible bottlenecks at the early design stages [execution time-line]. However, the simulation-based analysis cannot guarantee reachability of the worst-case executions. Moreover, it requires substantial time span (from minutes to
days) to obtain converged prediction results. Therefore, it can be selectively used for a detailed exploration of execution problems in the architecture, such as buffering and task-interleaving problems.

The combination of these two analysis types eliminates the deficiencies and enhances the advantages that each method imposes. For example, the worst-case performance predictions which can be quickly obtained from the schedulability analysis, can be used as a guideline for next iterations of the design-space exploration process, and fast termination of inefficient architecture alternatives. The filtered alternatives satisfying the system requirements, are further simulated to identify performance attributes, which cannot be extracted by formal methods, like possible bottlenecks and average response time. Moreover, at the exploration saturation point (local or global optima), the simulation techniques facilitate the analysis of other efficient alternatives.

The RQ1 defines the problem of combining both computation and network load and delays in order to increase the performance prediction accuracy. Ideally, both computation and network activities should undergo either schedulability and/or simulation analysis in an integrated way. However, since that there are no methods and tools supporting intrinsic network configuration and protocols, in accordance with computation schedulability or simulation analysis, we have proposed a hybrid analysis method. In this method, we combine the worst-case execution time of each operation with separately computed worst-case transmission time of the data through the network. The disadvantage of this approach is that in simulation analysis, the exact starting point of an operation, associated with network transmission for the computation of the network load is not taken into account.

The optimal way to integrate network and computation delay analysis for achieving RQ1 without losing the detailed resource models (RQ2) is to focus on worst-case transmission delays where all events compete simultaneously for network transmission.

Amongst various available technologies to form networks, the flexibility and ease of deployment of Wireless Local Area Networks (WLANs) [IEEE, 2007], render them as an attractive choice for distributed systems supporting node mobility. WLANs are inherently based on the principle of carrier sense multiple access with collision avoidance (CSMA/CA), which can transmit messages with acceptable delay in low to medium traffic loads. Theoretically, CSMA/CA does not guarantee avoidance of simultaneous transmission by multiple nodes, hence may give rise to collisions. This phenomenon affect the delay performance, which is also adversely affected by environment factors such as indoor/outdoor obstacles and interference from other wireless networks. An alternate to CSMA/CA is Time Division Multiple Access (TDMA) based scheduling, which can avoid collisions, though such a solution is not compliant with the IEEE standard (IEEE, 2007). Hence, it requires changes in the WLAN device driver and an explicit management layer. Considering that WLANs have limited bandwidth, a detailed analysis for delay predictions is desired for accuracy.

3.3.1. Wireless network delay prediction model

WLANs are random access networks, implying nodes that may transmit messages at random moments. Randomization is used to decrease the contention level in the network to avoid unnecessary delays, while it is controlled by a Medium Access Control (MAC) parameter, called contention window (CW). Randomization causes uncertainty and performance fluctuations that we report by predicting best-, average- and worst-case network delays. To this end, an analytical performance model for wireless network delay prediction (WNDP) is proposed. This model leverages overall end-to-end delay performance analysis of the CB-RTDS, as required in this paper.

A distributed system in operation executes a set of tasks composed out of the functions of individual SW components. The SW components can be mapped on different HW nodes. These functions communicate with each other by firing messages. Thus, the network system under consideration is composed of N periodic functions that are indexed by a set $\mathcal{K}$, and a single domain wireless network upon which functions compete for access.

$$\mathcal{K} = \{1, 2, 3, \ldots, N\}. \quad (21)$$

We assume that the radio links have no external interference. Extension of our distributed system to Wide Area Network can be made by also considering delays resulting due to intermediate links involved. To proceed with analysis, we tag an arbitrary function to be the reference function, indexed by $k$. The transport process and the MAC process corresponding to the reference function are termed accordingly. The requirements of the function $k$ are expressed in terms of its period $d_k$ and message length $m_k$. Functions establish a transport connection over a wireless unmanaged network of MAC processes, which compete to access the channel according to their MAC settings. Messages of the functions may be fragmented according to the specification of the layers from the concerned layer till the MAC layer, where MAC Protocol Data Units (MPDUs) are formed. Details of the fragmentation process are given in Appendix. As a result of this fragmentation, MPDUs are formed with payloads of different sizes. For our delay analysis, a mean payload size suffices — it is denoted by $E(l_k)$ and derived in Eq. (29). The derivation is based on Mean Field Analysis. The time taken to transmit $E(l_k)$ is denoted by $t_{\text{mac}}(E(l_k))$ and expressed in Eq. (32). WNDP performance refers to the application layer link-delay performance that relies on the MAC layer link-delay performance. We determine the best-, average- and worst-case delays.

MAC arbitration

During its execution, the MAC reference process may access the channel exclusively to transmit its MPDU, thus experiencing minimal delay. Contrary to this, if multiple MAC processes access the channel simultaneously, the situation is termed as a collision. All MAC processes involved in a given collision, have a failed transmission that incurs a wasted busy state on the channel. The situation is followed by a collision-resolution process, during which the colliding MAC processes attempt again after waiting a longer, randomly distributed time interval. The reference process may undergo multiple collisions before it either transmits or discards the MPDU. The collision-resolution process after the $i$th collision, is considered an order-$i$ resolution. In this analysis, we restrict to only order-1 resolutions (first-order resolutions). The restriction is justified by the fact that in a system with unsaturated MAC queues, the probability of collision is much lower than in a system with saturated MAC queues. The size of a collision, $c$, is the number of MAC processes involved in it: $2 \leq c \leq N$. Collisions of large sizes are less probable. Based on these principles of MAC arbitration, the following three types of delays are determined.

1. **Best delay** is defined as the time taken to transmit in the first attempt, all MPDUs related to the reference function message $m_k$. Denoted by $\lambda_k^\dagger$, it is determined as

$$\frac{\lambda_k^\dagger}{n_k} = t_{\text{mac}}(E(l_k)). \quad (22)$$

where $n_k$ is the count of MPDUs that are formed due to function message fragmentation and $t_{\text{mac}}(l)$ maps the time to transmit $l$ bytes of payload at the MAC layer (refer to Eqs. (29) and (32) for details).

2. **Average delay** is defined as an average time taken to transmit all MPDUs related to the reference function message, $m_k$. Its derivation is based on the best case and the cases that arise after the first-order collision resolutions. We assume that the first-order collision resolution is done sequentially, with each involved MAC process being equally favored after the collision. Over longer duration, the system achieves
behavior closer to this assumption. This assumption is supported by the fact that all MAC processes draw equal average backoff waiting times due to similar uniform distributions. Formally, the average delay, $\lambda''^n_k$, is computed as the weighted mean of times for the best delay and the first-order collision resolutions. For the latter events, we consider collisions of all possible sizes that are resolved equally, in favor of each MAC process involved with them, giving an average delay of

$$\frac{\lambda''^n_k}{n_k} = p([k]) \cdot \frac{1}{n_k} + \sum_{c=2}^{N} \sum_{c \in S} p(s) \left[ E\left(\tau'_k(s) + E \left( \tau''_k(s) \right) \right) \right].$$

where $p(\cdot)$ maps a set of MAC processes to probability of their transmission. The introduced terms are defined as follows.

- $p([k])$ is the probability of transmission by MAC process, $k$ only, which is the best-case delay;
- $s_k(c)$ is the collection of collision sets of size $c$ that contain the reference MAC process, $k$;
- $p(s)$ is the probability of a collision by MAC processes in $s$;
- $\tau'_k(s)$ is the duration of the collision state on the channel by MAC processes in $s$ — it excludes the collision-resolution time;
- $E(\tau''_k(s))$ is the expected time of first-order collision resolution — it excludes the duration of the collision state.

The event space considered for the average delay is composed of a successful transmission and all possible collision cases. Therefore, $p([k])$ and $p(s)$ are normalized within this space. These terms are derived as follows.

- $p([k])$ — this probability term is computed by deriving a mapping, $p(z)$, of all MAC processes in $z$, to a probability of their simultaneous transmission. MAC processes not in $z$, $K \setminus z$, are considered silent. In our system, we assume that function phasings are not known in an unsaturated network with periodic functions, the probability of transmission by MAC process $k$, is given by $\frac{1}{aSlotTime}$, where $aSlotTime$ is defined as the worst-case time a device must wait before it can reliably know about the idleness of the slot. Parameters $d_k$ and $aSlotTime$ are expressed in the same time units,

$$p(z) = \prod_{i \in z} \frac{1}{d_i/aSlotTime} \cdot \prod_{j \in K \setminus z} \left( 1 - \frac{1}{d_j/aSlotTime} \right).$$

where $p(z)$ at $z = \{k\}$ gives $p([k])$.

- $s_k(c)$ — an arbitrary subset of $K$ with cardinality $c$, may or may not contain $k$. Hence, we make a partition of $K$ into two subsets, $K'_c = K \setminus \{k\}$ and $\{k\}$. A union of an arbitrary subset of $K'_c$ with cardinality $c - 1$ and $\{k\}$, yields a set which necessarily contains $k$. Subset $s_k(c)$ is defined by

$$s_k(c) \triangleq \left\{ e \cup \{k\} \mid e \subseteq K'_c, |e| = c - 1 \right\},$$

where

$$|s_k(c)| = \binom{n - 1}{c - 1}$$

and the corresponding probability term is computed by evaluating $p(z)$ at $z = s$.

- $\tau'_k(s)$ — the duration of the collision state on the channel by MAC processes in $s$ depends on the size of the largest MPDU in $s$, hence

$$\tau'_k(s) = t_{\text{MAC}} \left( \uparrow E(l_i) \right).$$

- $E(\tau''_k(s))$ — this time term is computed by considering all possible orders of collision resolution that allow the MAC process $k$ to transmit. Each order represents a transmission schedule of MAC processes, while all orders are assumed to occur equally likely. All processes ahead of the MAC process $k$ incur delay to it. Given a collision set, $s$, let $s_k(s)$ be a set of all orders, where $|s_k(s)| = |s|!$. This leads to the time term

$$E(\tau''_k(s)) = \frac{1}{|s|!} \sum_{i=1}^{\text{pos}(k)} \left( \sum_{s \in S_k(i)} \left( t_{\text{MAC}} \left( E(l_i) \right) \right) + b_k \right),$$

where

- $\text{pos}(k), k \in \{1, 2, \ldots, |s|\}$ maps to the position of $k$ within the order $(s);$  
- $b_k = aSlotTime \cdot CW_1/2$ is an average backoff time of MAC process $i$ with contention window $CW_1$, during the first-order collision resolution.

3. **Worst delay** is defined as the time taken to transmit a message positioned as last in the schedule, during the first-order collision resolution involving all functions, all MPDUs related to the reference function message $m_k$. This worst delay $\lambda'''^n_k$ is determined by

$$\frac{\lambda'''^n_k}{n_k} = \max_{i \in K} \left( \uparrow E(l_i) \right) + \sum_{j \in K} \left( t_{\text{MAC}} \left( E(l_j) \right) + b_k \right).$$

The first term in Eq. (25) is the duration of a size $N$ collision. This concludes the derivation of the best-, average- and worst-case delays of the wireless network. These delays are used in conjunction with the processing delays at the nodes to determine the end-to-end delay per scenario.

3.3.2. Computation analysis method

As already mentioned, the overall performance analysis of the system is being performed by both schedulability and simulation-based analysis. The system model $S.M$ is initially applied to schedulability analysis, filtering out the alternatives that do not meet the worst-case boundaries. Subsequently, the architectural alternatives $S.M$ satisfying the system requirements, are analyzed by the simulation-based analysis, obtaining average-case delays and detailed execution time-line data, while identifying possible bottlenecks.

The triggering periods defined in the scenarios $S_n$, the execution time of the incorporated scenarios and involved functions, as well as the communication delays computed by the WNDP, and the platform definition, act as source for the schedulability analysis of the system. The schedulability analysis provides with the worst-case end-to-end delays of the scenarios, processing utilization and network load. These obtained end-to-end delays are compared with the respective deadlines $D_n$ defined in each individual system scenario, see Eq. (9).

In order to analyze the performance of the system, we have deployed a number of proprietary and existing tools. For the schedulability analysis of the system, we have deployed the MAST analysis tool (Gonzalez et al., 2001). It supports multiple schedulability analysis techniques such as EDF (Bertuzzo, 2011), priority based, holistic (Tindell and Clark, 1994) in preemtible and non-preemptible versions and combination of them for heterogeneous real-time distributed systems (Rivas et al., 2011). Moreover, in fixed priority scheduling policies different aperiodic servers may be specified (polling, sporadic, resource reservations) and hierarchical scheduling is supported through primary and secondary schedulers. As we have already mentioned in Section (3.2), each defined scenario (Eq. (9)) in the system model (Eq. (20)) is triggered by
event arrival patterns. MAST model supports different event arrival
patterns such as periodic, singular, unbounded aperiodic, sporadic
and bursty, and in combination with the support of multiple event
handlers for creating multi-path end-to-end flow in the execution
scenarios (delay, offset, fork, branch, join, merge, rate divisor),
esases accurate system behavioral modeling. The MAST analysis tool
provides with performance results such as best- and worst-case
latencies, throughput, blocking times and resource utilization.

The system model $S,M$, modeled by MARTE profile, is tranformed
of the MAST format as in Eq. (26) by the Marte2Mast tool
(Medina and Garcia Cuesta, 2011; Medina and Cuesta, 2011).

\[ MASTE_m \xrightarrow{\text{marte2mast}} MAST_m. \]  

The architecture alternatives with satisfactory worst-case del-
ays, computed by the MAST tool are further analyzed by simu-
lating the system model with the JSimMART tool (Cuesta,
2010). JsimMART is a discrete event simulator tool providing with
average-case response times, detailed task execution time-lines,
buffer/bus/network-load time-lines and task-interleaving/blocking
aspects. Moreover, it provides the resource utilization metric (CPU,
network) and it eases the identification of possible bottlenecks.
However, it requires as input a pre-formatted MAST2 system model
(Harbour et al., 2013), which is the new generation of the MAST
model that the schedulability analysis tool utilizes (transformation
from MAST to MAST2 model, see Eq. (27)). Hence,

\[ MASTn \xrightarrow{\text{mastTomast2}} MAST2_m. \]

For the schedulability analysis we have focused on the offset-
based technique, which was initially proposed from Tindell (1994).
The offset-based analysis was extended by Palencia and Har-
bour (2003) for supporting multiprocessor and distributed sys-
tem schemes, while supporting both priority and EDF schedul-
ing policies. The offset based analysis approximates the interference
of each end-to-end flow by utilizing maximum interference func-
tion that is independent of the current phase of the end-to-end
flow. The advantage of this analysis method is that it provides
less pessimism compared to the other methods for schedulabil-
y on distributed systems, i.e. holistic (Tindell and Clark, 1994).

This is happening because offset-based analysis performs globally
schedulability analysis on the distributed system, in contrast to
the holistic analysis where independent analysis on each process-
ing or communication resource is taking place. Detailed description
of the adopted offset-based analysis exists in Palencia and Har-
bour (2003), showing significantly better results than using holistic
analysis for EDF.

3.3.3. Overall performance analysis

The performance analysis of the system is based on the individ-
al computation of the network delay and the utilization of this
metric in accordance with the computation delay, in order to com-
pute the overall CB-RTDS performance analysis.

The performance analysis method utilizes the methods and
models described in the previous Sections 3.1–3.3 in order to sup-
port the research questions RQ1, RQ2 and RQ3 that have been de-
defined in the introduction section.

During the SW/HW mapping stage, the cycle-accurate perfor-
ance metrics of components are converted to execution time
slots for each operation (Eqs. (16), (18)). During the analysis, each
time an operation is executed on the processor, the amount of time
equal to the time slice is subtracted from the execution time of the
operation (Fig. 5), where the time slice or quantum is the period
of time for which a process is allowed to run in a preemptive mul-
titasking system. When the remaining execution time of an opera-
tion becomes equal to zero, the execution of this operation is over
and the delay from the first call of this operation until the end of
its execution is the actual execution delay of this operation. When
an operation is followed by a network transmission, then the exe-
cution of the scenario is blocked until the moment that the trans-
mision of the message through the network is realized. The time
elapsed during the network transmission is added to the end-to-
delay of the execution scenario.

Fig. 5 depicts an example of two scenarios executed on a
two-node system, incorporating the network communication be-
tween the two nodes. The Scenario 1 involves operations of
components C1 and C2, which communicate through the network
and the Scenario 2 involves operations of component C1. C1
and C2 are hosted on Node 1 and Node 2, respectively. In this example, triggering of the two scenarios occurs simultaneously. Functions f1 and f3 compete with each other to be executed on Node 1. Assuming that the f1 function has higher priority than f3, it is executed first. The end-to-end delay \( \Lambda^s_n \) for scenario \( S_n \) is computed as the sum of the execution times of the operations, delays initiated by the context switches and the network delays (if applicable). Let \( E_n \), \( C_n \) and \( W_n \) be the sets of the operations, context switches and network links involved in scenario \( S_n \), so that \( \Lambda^s_n \) becomes

\[
\Lambda^s_n = \sum_{i \in E_n} \epsilon^i + \sum_{j \in C_n} \tau^j + \sum_{k \in W_n} \left\{ \begin{array}{ll}
\lambda^s_n, & x = \text{bct} \\
\lambda^s_n, & x = \text{acet} \\
\lambda^s_n, & x = \text{wct} 
\end{array} \right. \tag{28}
\]

where \( \epsilon^i \) is the execution time of the function \( i \) as presented in Eq. (17), \( \tau^j \) is the delay caused by a context switch, and \( \lambda^s_n \), \( \lambda^s_n \) and \( \lambda^s_n \) are the best-, average- and worst-case network delays, derived in Eqs. (22), (23) and (25), respectively.

Since the network simulators integrated into both MAST and JSimMAST analysis tools do not consider the details of the lower OSI layers, such as e.g. the wireless network protocols that have specific retransmission schemes, we have integrated two mechanisms for the network performance analysis. The first method is a formal method based on the schedulability analysis of the network (WNDP discussed in Section 3.3.1), while the second is based on the well-known network simulator NS3 (Riley and Henderson, 2010). Both network analysis approaches receive as input the MAST models (PreMAST), generated by the MART2MAST converter. From the MAST models, the approaches compose the network architecture of the system and they analyze its performance by computing best-, average- and worst-case transmission delays. The MAST and MAST2 models are updated with the new transmission delays. In other words, for each scenario that incorporates network transmissions between the executed operations, the third term in Eq. (28) is computed with the corresponding network delay.

The presented performance analysis method features many advantages. First, it incorporates the cycle-accurate component performance properties extracted from the profiling and modeling phase (RQ2). Second, due to the combination of formal and simulation-based analysis techniques, the analysis mechanism predicts both guaranteed WCET and ACET (RQ2). The hybrid analysis method (formal and simulation) enables early identification of WCET (schedulability), while it identifies possible bottlenecks providing the execution time-line data (simulation) of the different scenarios. Finally, the integration of highly accurate network analysis tools during the performance analysis phase, leads to accurate CB-RTDS performance predictions, which support various network schemes and protocols (RQ1).

4. Autonomously navigating robots

4.1. Introduction to the case study

The profiling, modeling and analysis method presented in Section 3 has been utilized to verify the real-time requirements of a more advanced case study with three robots instead of one (Triantafyllidis et al., 2013), where multiple autonomously navigating robots scan an environment simultaneously, composing three individual indoor maps. The three individually generated maps are then combined to create a global reference map. Moreover, we should propose the optimal architecture with respect to the following multi-dimensional criteria: latency of control loops, robustness and system cost.

Let us briefly outline the system hardware and functionality. The autonomously navigating robot system is composed of multiple hardware units: (a) three individual robots, each of them integrating an embedded PC used for the control of the signals coming from- and directed to- the robot, and (b) a set of processing remote workstations, executing heavy tasks of navigation and the map composition of the environment (Fig. 6).

The provided robots have a set of infrared laser sensors and a differential axis with two wheels (left and right). Each embedded PC works as an interface for reading the sensors data (odometry and infrared), as well as for controlling the robot by adjusting the velocity of the two wheels attached on the differential axis. The data from the infrared sensors are used to compute the maps (Occupancy Grid) of the obstacles surrounding the robots. Based on the generated maps, remote workstations transmit timely feedback signals to the robots for the wheel control, thereby imposing real-time requirements for specific tasks.

4.2. SW component selection

For the case study, we have selected a set of suitable components from the ROS (Garage, 2007) repository, enabling the functionality of autonomous control loops, map reconstruction for each robot and the multiple-map integration to one global map.

From the hardware point of view (Fig. 6), the system is fully distributed. The robots send the infrared and odometry data to a Processing Unit, using pre-defined sockets. The end-to-end delay, defined from the moment that the sensor data are read by the embedded PC to the time that they are transferred to the Processing Unit, is being considered as a constant delay. Hereby we consider that the embedded PC acts as a black box providing these sensor readings. The Processing Unit serves as an intermediate node converting the odometry and the infrared data from the robot binary format to ROS-compatible published topics. The ROS-compatible sensor readings are then transmitted through the network infrastructure to the Workstations for handling 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 the surrounding map is the cornerstone of a global indoor map reconstruction, defining the reference model of the real world. In addition to this, the actuality of the sensor data determines whether newly appeared objects in the environment can be detected and depicted into the map, so that they can be avoided while the robot is navigating through the space. Moreover, the stitching of the three individually generated maps to one map is also a crucial task, since it involves possible transfer of each map (16 MB size) through the network infrastructure from node to node. The conversion of the three individual maps to one global map imposes high network utilization and also requires substantial processing resources.

We have selected the GM (Slam Gmapping) component from the ROS repository for the map generation. The GM component receives the laser and the odometry data and generates the actual 2D map as well as the projection of the odometry data onto the map, thereby representing the localization of the robot in the environment.
For the navigation of the robot, the MB (Move Base) component was selected. This component subscribes for the actual 2D map and the localization of the robot, as well as for the robot infrared sensor data. The MB component input uses either a constant global map (Occupancy Grid) generated by the GM component, or a local map (orthogonal window surrounding the robot) using the infrared sensor data. The former approach is ideal for finding the optimal global plan, while the latter approach does not require transmission of a global map, saving up network and memory resources. Finally, the MB component publishes the navigation control signals, which guide the robot to its destination.

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 interconnection of the embedded PC of the robot and the sensor data, while it provides the ROS publish-subscribe mechanism. The MD publishes the sensor data (odometry and laser) 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.

The MS (Map Stitch) component composes two individual maps with the same resolution, to one map. It can operate with only two inputs at a time, meaning that for the integration of three individual maps, the MS component needs to be executed twice. It obtains the input data from a file generated by the Map Saver component.

### 4.3. Profiling of the SW components

The selected SW components have been profiled for two different HW platforms: 15–750 CPU (1st generation) and 15–2520 Mobile CPU (2nd generation) with specifications summarized in Table 3. Furthermore, during the profiling process, the ProMo tool automatically generates a performance model for each individual SW component. These resource models contain the cycle-accurate performance metrics extracted during the profiling. An example of the performance model of the MB component in a pseudo xml format is depicted in Fig. 7.

#### Table 2

<table>
<thead>
<tr>
<th>Scenario name</th>
<th>Period (ms)</th>
<th>Deadline (ms)</th>
<th>Message length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM:odom scenario</td>
<td>37</td>
<td>100</td>
<td>165</td>
</tr>
<tr>
<td>GM:laser scenario</td>
<td>79</td>
<td>100</td>
<td>1209</td>
</tr>
<tr>
<td>MB:odom scenario</td>
<td>37</td>
<td>100</td>
<td>165</td>
</tr>
<tr>
<td>MB:laser scenario</td>
<td>79</td>
<td>100</td>
<td>1209</td>
</tr>
<tr>
<td>GM:frame scenario</td>
<td>50</td>
<td>100</td>
<td>1999</td>
</tr>
<tr>
<td>GM:conversion scenario</td>
<td>50</td>
<td>100</td>
<td>159</td>
</tr>
<tr>
<td>MB:newgoal scenario</td>
<td>500</td>
<td>500</td>
<td>171</td>
</tr>
<tr>
<td>MB:frame scenario</td>
<td>50</td>
<td>100</td>
<td>1999</td>
</tr>
<tr>
<td>GM:map scenario</td>
<td>1000</td>
<td>1000</td>
<td>16777216</td>
</tr>
<tr>
<td>MB:nav scenario</td>
<td>50</td>
<td>100</td>
<td>144</td>
</tr>
<tr>
<td>GM:map stitch</td>
<td>60,000</td>
<td>60,000</td>
<td>16777216</td>
</tr>
</tbody>
</table>

#### Table 3

<table>
<thead>
<tr>
<th>HW Platforms used for profiling.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>i5 750</td>
</tr>
<tr>
<td>i5 2520</td>
</tr>
</tbody>
</table>

#### Fig. 7

Example of a ProMo performance model in pseudo xml format for the Move-Base (MB) component.

### 4.4. Scenario definition

As mentioned, the system executions are based on execution scenarios defined by the architect. In this case study, the behavior of the system is specified by 31 execution scenarios, where 30 of them represent the navigation and the map composition, while the last one provides merging of the three local maps to one global map. For navigation and map composition, 10 individual execution scenarios per robot are required (3 robots = 3×10).

Each scenario delivers a specific functionality (SW components) and is characterized by a triggering period and a deadline. The most crucial real-time scenarios are depicted in Fig. 8. In Table 2, we describe the periods and the deadlines of the 10 scenarios for each robot and the map-merging scenario (the periods and the deadlines are identical, thus it is not necessary to present them for each individual robot).

At this phase the system architect defines the execution scenarios by loading and instantiating the required SW components and interconnecting the provided and required interfaces within activity diagrams. Then he defines the scenarios triggering periods and the real-time deadlines. Following, we present the scenarios and the functionality as well as their real-time requirements.

The scenarios specifying the sensor data read and buffers update functionality are set to have a hard real-time deadline of 100 ms. Thus, the crucial scenarios MB:Nav and GM:Conversion (with a 50-ms triggering period) load the fresh sensor data at
least once every two iterations. The scenario GM::Odom specifies 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 the 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 specify transmission of the infrared odometry sensor positions in the 3D space to the GM and MB components, respectively.

The scenario GM::MapConversions transmits the projection of the odometry data on the composed map to the MB component for the navigation process.

The scenario MB::NewGoal represents the workload on the system when a new target/goal arrives from RVIZ during the navigation process.

The scenario GM::Map specifies generation of the 2D map of the robot local environment. The Map operation is the core of the GM component. The operation is triggered with a 1000-ms period. During one iteration, the 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 (100 ms). The scenario describes the feedback control loop for the robot wheels. The Nav operation of the MB component is triggered every 50 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 to the engine of the robot wheels (see Fig. 8).

The scenario MS::MapStitch merges the three individual maps composed by the three robots to one global map, executing the MapStitch operation of the MS component. The period as well as the deadline are set to 60 s.

The deadlines for the scenarios have been defined to guarantee that the system: (a) avoids collisions with the newly appeared objects, (b) generates a map every minute and (c) does not introduce a noticeable response delay when a new goal is set.

4.5. HW platform definition

As soon as the system architect has defined the system functionality, he composes the system architecture by specifying the HW platform and mapping the involved in the scenarios SW component instances on this platform (see Fig. 9). The system architect loads the HW components from the repository and by instantiating them he defines the HW platform. In this particular case study, two CPU types are employed for processing nodes on workstation’s side: the i5–750 and the i5–2520. Both incorporate 4 cores, meaning that they can execute many tasks simultaneously. The interconnection between the nodes is based on an ad-hoc wireless network operating at 54 Mbps in the IEEE 802.11g mode. As a design choice, wireless network infrastructure has been chosen instead of wired, since it features flexibility and practicality at a competitive data rate transfer. Because the robots move through the space transmitting data from the sensors, it is not practical to connect the processing nodes that lay on top of the robots with cables. Moreover, in some cases, when increased processing resources are required, the installation of additional processing node(s) under a wireless network is an easy task (no cables, switches), increasing the flexibility of the system. Finally, when an ad-hoc type of wireless network is adopted, the cost is reduced due to the fact that no additional equipment is required (router, switch).

In Fig. 9, Architecture A is a single-node architecture alternative. All the tasks are assigned on the i5–750 CPU node.

Architecture B deploys two processing nodes. The computationally expensive GM component, as well as the MS are mapped on the i5–750 CPU node. The other SW components are mapped on the i5–2520.

Architecture C incorporates three processing nodes. Each processing node executes the whole navigation and map creation process, while Node#1 and Node#2 execute an instance of the MS component.

Architecture D is based on four nodes. The Node#1, Node#2 and Node#3 execute the navigation and the map composition process. The three composed maps are subsequently transmitted to the Node#4 which executes the MS component.
As soon as the execution platform is defined, the system architect maps on this platform the SW component instances that are involved in the execution scenarios. Further details of the SW/HW mapping are presented in Fig. 9.

4.6. Performance evaluation

The evaluation of the architecture alternatives has been performed by applying both types of analysis: the schedulability (MAST + WNDP tool) and the simulation analysis (jSimMAST + NS3 tool). We first perform schedulability analysis to analyze whether each alternative is schedulable. The candidates that satisfy the real-time requirements are further applied to simulation analysis. The obtained performance predictions (schedulability and simulation analysis) are depicted in Table 6, which contains the processor and the network utilization, as well as the execution scenario delays.

The performance evaluation phase strives for the identification of possible bottlenecks. Moreover, it also obtains guaranteed worst-case and realistic average-case delays. Finally, by applying an additional robustness test to the architecture alternatives, it identifies the optimal architecture alternative regarding the performance of the real-time tasks, robustness and the cost of the composed system.

As we have already mentioned in Section 3, we first compute the network delays, and subsequently, we use these network delays in the schedulability and simulation analysis of the overall system (computation and network delays).

### 4.6.1. WNDP evaluation

We have evaluated our WNDP model by the requirements of the experimental autonomous navigation system setup described in Table 2, in which nodes communicate in a wireless unmanaged network.

For the transport layer, TCP is selected, whose implementation in Robot Operating System (ROS) mandates a TCP ACK for each segment, thus the transport layer length of sequence of segments (see Appendix), $seql^h = 1$. Due to the ACK overhead of TCP, the transport layer connections may be considered by the MAC layer as two-source information emitters. This approach is similar to Bruce et al. (2008), which focus on derivation of throughput. Other specifications taken from ROS are the segment payload limit, $G = 1, 448$ bytes, TCP ACK size, $ACK^p = 66$ bytes, the MPDU payload limit, $L = 2, 304$ bytes and OSI headers sizes, $h^p = 32$ bytes, $h^h = 20$ bytes and $h^{mac} = 34$ bytes. Tests are conducted for WLAN technology, IEEE 802.11g (IEEE, 2007) tuned at a contention window of $CW = 32$. Other PHY/MAC parameter values are listed in Table 4.

From the result reported in Table 5, the first observation is that the average delay is more than twice as large as the best delay. This is due to the "collisions" time overhead that incurs an extra delay of at least the same order as the best delay.

Another observation is the relatively large value of the worst delay. We underpin the effect of our mean-field approach on the two terms in Eq. (25). The first term strongly depends on the size of the largest MPDU involved in a given "collision". Because of the fact that MPDU sizes reflect only the mean payload generated at the transport layer, the first term of Eq. (25) has a low impact. The second term strongly depends on the number of sources. Our WNDP model reduces the transmitters of data segments and corresponding control segments to one virtual source per function. Thus the length of the longest schedule during the first-order "collisions" resolution is effectively reduced by an order of mean MPDU count per function. Consequently, the second term has a low impact as well. Given the conditions that lead to the worst-case delay, their occurrence probability is almost negligible. The occurrences represent system hyper periods, which is estimated to be 4,229 days at the function and our WNDP model levels. Lastly, the lower delays of IEEE 802.11g are due to its higher transmission rate.

### 4.6.2. Architecture evaluation

The network delays computed by the WNDP model, are integrated into the models of the four architecture alternatives. Performance prediction metrics are obtained for each architecture alternative (see Fig. 9), as depicted in Table 6. For each

<table>
<thead>
<tr>
<th>Network transmission</th>
<th>BCET</th>
<th>ACET</th>
<th>WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM:conversion scenario 1</td>
<td>0.768</td>
<td>2.112</td>
<td>6.464</td>
</tr>
<tr>
<td>MB:collision scenario 1</td>
<td>9.288</td>
<td>2.432</td>
<td>6.464</td>
</tr>
<tr>
<td>GM:conversion scenario 2</td>
<td>0.768</td>
<td>2.112</td>
<td>6.464</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSlotTime (μs)</td>
<td>9</td>
</tr>
<tr>
<td>SIFS (μs)</td>
<td>10</td>
</tr>
<tr>
<td>Data rate (bits/μs)</td>
<td>54</td>
</tr>
<tr>
<td>ACK (bytes)</td>
<td>14</td>
</tr>
<tr>
<td>PHY header (bits)</td>
<td>144</td>
</tr>
</tbody>
</table>
Table 6
Performance evaluation of four architecture alternatives (WCET).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpu utilization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core1</td>
<td>na</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core2</td>
<td>na</td>
<td>59.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core3</td>
<td>na</td>
<td>59.5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core4</td>
<td>na</td>
<td>23.4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core1</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core2</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core3</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core4</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core1</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core2</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core3</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core4</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core1</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core2</td>
<td>na</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Util.</td>
<td></td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slack</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenarios</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Odom1</td>
<td>na</td>
<td>3680 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Odom2</td>
<td>na</td>
<td>3681 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Odom3</td>
<td>na</td>
<td>3687 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Laser1</td>
<td>na</td>
<td>3685 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Laser2</td>
<td>na</td>
<td>3677 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Laser3</td>
<td>na</td>
<td>3687 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Odom1</td>
<td>na</td>
<td>3678 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Odom2</td>
<td>na</td>
<td>3679 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Odom3</td>
<td>na</td>
<td>3685 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Laser1</td>
<td>na</td>
<td>3687 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Laser2</td>
<td>na</td>
<td>3678 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Laser3</td>
<td>na</td>
<td>3688 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Frames1</td>
<td>na</td>
<td>3853 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Frames2</td>
<td>na</td>
<td>3862 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Frames3</td>
<td>na</td>
<td>3859 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Frames1</td>
<td>na</td>
<td>3851 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Frames2</td>
<td>na</td>
<td>3890 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Frames3</td>
<td>na</td>
<td>3856 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:GMCon1</td>
<td>na</td>
<td>10348 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:GMCon2</td>
<td>na</td>
<td>497 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:GMCon3</td>
<td>na</td>
<td>495 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:NewGoal1</td>
<td>na</td>
<td>14088 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:NewGoal2</td>
<td>na</td>
<td>491 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:NewGoal3</td>
<td>na</td>
<td>494 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Nav1</td>
<td>na</td>
<td>319278 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Nav2</td>
<td>na</td>
<td>508 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB:Nav3</td>
<td>na</td>
<td>506 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Map1</td>
<td>na</td>
<td>2676 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Map2</td>
<td>na</td>
<td>497 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GM:Map3</td>
<td>na</td>
<td>495 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS:MapStitch</td>
<td>na</td>
<td>3.44 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

architecture alternative (see Fig. 9). Table 6 depicts the schedulability and the simulation-based metrics (if available). More specifically, for each architecture alternative and for each type of analysis, resource utilization metrics like CPU and network utilization are depicted. Additionally, end-to-end delays, including the communication and calculation delays, are presented for each individual critical scenario of the system. The obtained performance metrics are compared to the requirements of the systems, leading to (a set of) optimal architecture alternative(s). In the sequel, the performance evaluation of the four composed architecture alternatives is presented.

Architecture A is based on a single-node approach. Due to the lack of processing resources (1 CPU), the architecture fails to pass the schedulability test due to overloaded resources (see Table 6). Thus, Architecture A is rejected at the early phase as insufficient. In addition to the schedulability analysis, simulation-based analysis was applied in order to identify the ACET and the WCET values, since the schedulability analysis was not able to provide the WCET due to high resource utilization.

Architecture B meets all the real-time requirements, as presented in Table 6. The dual-node system has the advantage of the increased processing resources (vs. architecture A), while the data transmission over the network is limited, since the scenarios GM:MapStitch retraining the maps are hosted in the same node with the scenario GM:MapStitch that merges them. Thus, no need of map transmission through the network is required.

Architecture C meets the real-time constraints and defines an alternative solution to the Architecture B. The three-node vs. the dual-node architecture provides an extra processing resource, but it generates more network traffic, due to the increased data synchronization among the nodes. The extra processing third node/processing resource offers additional system availability in case that one of the other two nodes fails due to a HW incident.

Architecture D fails to meet the real-time requirements. More specifically, the scenario GM:MapStitch fails to meet the end-to-end deadline (60 s). By applying simulation analysis, we have found that the rest of the system deadlines are met, meaning that: (a) the transmission of the maps through the network infrastructure bounds the performance of this alternative and (b) this architecture could be a possible candidate for a system where the task of the 3 maps merging to 1 map is not crucial.

The analysis has shown that, out of the 4 architecture alternatives, only the Architectures B and C meet the system’s real-time
requirements. Architecture A fails to meet the requirements due to the lack of processing resources, while Architecture D is not sufficient due to the high network utilization which influences the end-to-end response delays in crucial scenarios. Architecture B features higher response delays in the majority of the execution scenarios (except the MS:MapStitch scenario). On the other hand, the Architecture C uses an extra processing resource being more costly than the Architecture B.

For more comprehensive analysis, a robustness test is applied to both architectures, in order to choose the optimal one. In our framework we define a robustness as the ability of the composed system to fulfill the non-functional requirements under overload conditions. In the current setup, the composition of one global map is being performed every minute. An increased global map composition frequency provides a fresher global map to the user of the system. As a robustness test, the map stitching frequency was set 5% higher, meaning the period of 60,000 ms is reduced to 5714 ms. The analysis of the obtained performance predictions has shown that in Architecture B, the CPU utilization has been increased, but without any violation of the real-time deadlines. In Architecture C, the additional network traffic (increased map data transmission period) lead to raise of the network utilization but without causing any real-time deadline violation. Both architectures B and C manage to meet all the real-time requirements and pass the robustness test. Architecture B features lower cost since that it deploys two nodes in contrast to the three nodes incorporated by architecture C. On the other hand, architecture C offers higher system availability compared to architecture B and lower resource utilization, due to the additional node. The dynamic mapping of the SW components to HW nodes allows the reconfiguration of the system on-the-fly, when HW incidents occur (broken communication between two nodes), thus increasing the system availability. With this process a three-node system (architecture C) can be transformed to a two-node (architecture B) and still meet the system requirements, enabling high system availability. An architect may choose architecture B when the primary system objective is the cost and architecture C when the architect strives for high system availability. In this particular case study, we select for implementation the architecture B, because the cost objective is higher in hierarchy than the availability objective.

4.7. Validation

Our prediction method was validated by implementing the optimal Architecture B and measuring the actual latencies of the most critical scenarios MB:Nav, GF:Map and MS:MapStitch. Following this, we compared the predicted end-to-end latencies of the critical scenarios to the actual end-to-end delays of the implemented system.

The system requires the availability of 3 individual robots that navigate through the space and compose 3 individual maps. Since we factually only used one robot, we captured the transmitted laser, odometry and TF topics, and saved them into three different transmission and storage data repositories in ROS (rosbags). These three rosbags are then transmitted providing the topics to the nodes that subscribe for these topics, thereby maintaining and mimicking the communication and storage behavior of 3 robots.

For the worst case, the predicted vs. actual latency deviations have shown to be within a 6% range, while for the average case the deviations are even lower, as presented in Table 7. In order to compute the actual worst-case and average-case latencies of the two scenarios, we have executed the 3-robot system navigation 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 latencies during that operation hour.

The comparison of the delays depicted in Table 7 and in Fig. 10 leads to some interesting observations. The WCET predicted by the schedulability analysis is higher than both the actual execution and the simulation analysis. For the experiment, this means that both the actual execution and the simulation analysis did not reach the WCET during the measuring and analysis of the system, respectively. Hereby, the experiment confirms what was already mentioned at the beginning of this paper: the simulation analysis cannot guarantee the reachability of the WCET prediction, in contrast to the schedulability analysis which provides the WCET due its formal approach.

In the GM:MapScenario, we notice that the WCET prediction by the simulation analysis is lower than the WCET of the actual execution time. This occurs because during the simulation analysis, the system did not reach and thus execute the scenario that imposes the absolute worst-case delay. The latter highlights the danger of building a system based on WCET predictions extracted only from simulation. Simulation cannot act as a reliable source for building hard real-time systems and should be combined alongside with the schedulability analysis. The joint simulation/schedulability analysis method provides with: (a) reliable prediction of the worst- and average-case execution time of the system’s scenarios and (b) identification of possible bottlenecks.

5. Discussion

In this section we present important hints and guidelines for optimizing the mapping of the tasks on the executing distributed platform using the ProMARTES framework. These
guidelines emerged from the experiments during the case study of the 3 autonomously navigating robots.

We have identified that the mapping of tasks on the CPU cores plays a decisive role in the obtained performance of the system. Therefore, we present two guidelines referring to optimized mapping. When two or more computationally expensive tasks are mapped on the same core, then the high amount of task interleaving leads to increased WCET of the scenarios based on these tasks. We have found that it is beneficial to map the identified heavy tasks on separate CPU cores so that heavy task interleaving is avoided. Moreover, in the actual system implementation, we have identified that when tasks are mapped statically on specific cores, then the WCET of the scenarios based on these tasks is reduced. This is explained by the fact that when a task migrates to another core, the L1 cache data need to be transferred to the new core, thereby causing cache misses.

Another important performance aspect is that scenarios with low-latency tasks, that are mapped on different nodes, show high execution times due to the disproportional network transmission delay, compared to the architectures where the low-latency tasks are mapped on the same node. By comparing the architecture alternatives B and C, and focusing on the GM::Laser scenarios, we identify that in Architecture B, where the involved MD and GM components are mapped in different nodes, the WCET is 59 ms and the ACET is 21 ms. In Architecture C, where the MD and the GM components are mapped on the same node, the WCET and the ACET is around 12 ms and 6 ms, respectively. Hence, this shows that low-latency tasks executed on the same node lead to clearly reduced end-to-end delays.

The wireless ad-hoc network analysis of the system has revealed a number of interesting issues derived from our tests made on IEEE 802.11g WLAN technology. First, delay performance varies considerably between network interface cards (NICs) from one vendor to another. This is due to vendor-specific implementation of the communication drivers of the NICs. We have also tested a mixed vendor scenario, in which NICs from different vendors are used. In this case, the network drops to even lower rates of transmission due to incompatibility between NICs from different vendors. For instance, when NICs with names AR5212 and IN6205 are connected, the transmission rate is dropped to 11 Mbps in case of IEEE 802.11g. The incompatibility phenomenon becomes well visible when a comparison is made using all NICs from AR5212. In the comparison, the transmission rate stays at 54 Mbps, thus imposing less communication delays. Another important observation is on shielding our test bed from interference with other wireless networks. In an open environment, there are always interfering signals disturbing the communications, so that the network resorts to lower transmission rates. This effect holds even for tests made during night time.

We have also observed that an architecture, in which the critical tasks are data- and execution-independent from the performance of all other tasks, forms a very robust system 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. For instance, a possible failure of the task that provides the Navigation process with the odometry data does not influence the critical MB::Nav scenario, but only decreases the delivered quality of the operation.

6. Applicability of the ProMARTES method

This section discusses the applicability of the ProMARTES framework on systems with different and diverse characteristics, scope and properties from the stated case study. First, we assess the presented case study to a superset of systems as they are referred in the literature. Then, we examine the formality of ProMARTES applicability to system architectures and platforms that are partly out of our research scope (no hard deadlines, no predictability).

In this paper, we have applied the ProMARTES framework on the real world problem of 3 autonomously navigating robots. This is a component-based system consisted of multiple processing nodes connected through wireless network links. It integrates 33 end-to-end deadlines and the communication among the components is being performed through publish-subscribe mechanism. The 3 robots navigating system can be considered as a Service Oriented Architecture (SOA) system and superset of SaaS (Service as a Robot), since that the application components provide services to other components. However, it cannot be generalized that ProMARTES framework is tailored for all the available SOA-based systems. Depending on the system specification, the ProMARTES framework may need a few adaptations or assumptions in order to support the modeling and the analysis. For instance the cycle-accurate performance attributes can be utilized only for systems or platforms that are based on native programming languages such as C/C++ and Fortran.

Abundance of SOA systems and communication platforms are based on Java and Python programming languages. For such systems, the ProMARTES framework cannot provide cycle-accurate performance metrics and only processing delay can be obtained by the system profilers. Nonetheless, late years JNI (Java Native Interface) has become a common practice enabling the use of routines developed in native languages (C/C++ libraries) by Java code. In such case ProMARTES is able to provide cycle-accurate performance models for the native libraries supporting highly-accurate performance analysis.

It has to be mentioned that distributed systems that utilize Java as programming language, focus more on the portability of the code and less on the low-level scheduling policies and performance for meeting hard deadlines. Moreover, the unpredictable performance of applications developed in Java, due to the task execution on top of the JVM and the lack of a direct native interface, assesses pure Java as a not recommended programming language for the development of hard real-time systems.

An example of a large scale distributed system that is based on Java, is the well known Hadoop framework, which is used for distributed storage. Incorporating many different nodes for saving data, providing simultaneously high availability rate and low processing time, Hadoop framework is a challenging case study for performance analysis. Unfortunately, due to the non-native
source code substance of this framework, its analysis cannot be influenced by cycle-accurate performance models. As we have already mentioned in the Section 2 (“Related work” section), Bulej et al. (2012) have proposed methods for the modeling and analysis of such systems. Similar work has been conducted by Woodside et al. (2014); Faisal et al. (2013) and Koziolek et al. (Becker et al., 2009; Brosch et al., 2012) where they utilize LQNs, QPNs for the modeling and related solvers for the performance analysis. Their case studies discuss Java-based servers that communicate through wired network connections, thus supporting the modeling and analysis of the Hadoop framework. ProMARTES is also able to provide the analysis of such a system, but at this current development phase it would be required an additional extension for profiling and generating performance models for Java-based applications, while obtaining execution time delays and therefore reducing the performance prediction accuracy.

7. Conclusion

In this paper we have proposed an accurate performance analysis method for CB-RTDS, that is supported by the ProMARTES toolkit, which integrates a number of analysis automation tools. The method supports all phases in the component design and the analysis life-cycle: (a) profiling and modeling of SW components at cycle-accurate level, (b) generation of MARTE-compatible system models based on the automatically generated resource models and (c) evaluation of the system models by both schedulability and simulation analysis techniques, providing useful metrics. Examples of such metrics are predicted latencies, throughput, resource usage and robustness. The proposed method was evaluated and validated on a real-world problem of 3 autonomously navigating robots generating 3 individual maps, which are finally merged in a global map.

The method features multiple benefits for CB-RTDS profiling, modeling and performance analysis. First, the profiling method provides cycle-accurate performance metrics obtained by the PMU (Performance Monitor Unit) of the attached CPU. Second, the automatically generated resource models comply with the commonly used UML-MARTE profile. Third, the integration of a formal and a simulation network analysis tool, that both support the low OSI layers, enables highly accurate network performance predictions. Both network analysis tools support different protocols (WLAN, Ethernet, Optic Fiber, Coaxial Cable), packet collisions and possible retransmissions. Beyond a certain threshold of network load, delays in contention-based WLANS increase with the network load. Delay can be bounded by prioritizing network access through parameter configuration. Fourth, the method incorporates two different types of performance analysis techniques: simulation and schedulability analysis, enabling guaranteed WCET, as well as identification of possible bottlenecks and ACET predictions. Finally, the ProMARTES framework is wrapped in the Eclipse Papyrus IDE, so that an architect can easily design the SW/HW architectures even in graphical form and then automatically convert the design into feasible system models.

This paper has stated three principal research questions at the introductory section. Below, we present the method aspects addressing these research questions, while focusing on the consequences (positive & negative) of each aspect.

RQ1. In order to support the performance analysis of CB-RTDS, a hybrid analysis method was proposed. This method integrates both highly accurate computation and communication delays. More specifically, the computation delays are based on low-level performance metrics, increasing the accuracy. The communication delays are computed separately (prior to computation), supporting intrinsic network configuration and protocols. The final system analysis model integrates the WCET of each operation with the worst-case transmission time of the data through the network. Since the network delays do not consider the exact triggering moment of the operations (pre-calculation of the worst-case situation), the method is conservative when simulation analysis is applied, as all events compete simultaneously for network resources.

RQ2. Our performance analysis mechanism has been established, based on the integrated approach of aiming at a guaranteed WCET, evaluating ACET and computing detailed time-line data in a simultaneous way. This mechanism integrates both schedulability and simulation-based analysis. The former provides early identification of WCET, while the latter features ACET and detailed time-line data. We first apply schedulability analysis for a fast pre-selection of attractive system architecture alternatives that do meet hard real-time requirements (WCET). Then, the selected architecture alternatives are applied to simulation-based analysis, thereby obtaining the ACET and the detailed time-line data metrics.

RQ3. The cycle-accurate component performance properties obtained from the Profiling phase, are used throughout all composition and analysis phases, to achieve sufficient performance prediction accuracy. At the Architecture Composition phase, the cycle-accurate attributes of the performance models are translated into the accurate execution time models (see Eq. (16)). These time models take into account the latencies caused by potential cache misses thereby enabling high prediction accuracy.

The overall performance analysis method is supported by the ProMARTES toolkit, which has a number of limitations that require further research. The component performance models can only be obtained for a Linux-based operating system, while they require the physical presence of actual HW platforms. Moreover, the ProMARTES toolkit does not support automated generation of the behavior models during the profiling, leaving this notorious task to the component developer. ProMARTES does not support GPU-based and operating system tasks, which may decrease the performance prediction accuracy. Last but not least, since this work is part of our proposed DSE framework (Bondarev et al., 2007, 2006; Triantafyllidis et al., 2013, 2012), it should be noted that the current version of the ProMARTES toolkit (open source available at Triantafyllidis, 2013) does automatically generate architecture alternatives, enabling automated architecture optimization (Triantafyllidis, 2015).

Appendix

Application to transport layer mapping

Payload of the reference application is transmitted in transport layer packets, termed as data segments, each one with an application payload limit that we denote by $G_k$ bytes. The delivery of data segments is acknowledged by transport layer ACKs, termed as control segments. Thus an application payload, $m_k$, is split into $\left\lceil \frac{m_k}{G_k} \right\rceil$ segments, of which $\left\lfloor \frac{m_k}{G_k} \right\rfloor$ are full while the last one carries $m_k - \left\lfloor \frac{m_k}{G_k} \right\rfloor \cdot G_k$ application payload. If the transport connection is reliable, a transport layer ACK is needed for each segment separately or together for a sequence of segments. Let transport layer header, size of a transport layer ACK and length of sequence of segments be denoted by $h^{IP}$, $ACK^{IP}$ and seq$^{IP}$, respectively.

Transport layer to MAC layer mapping

Similarly each segment is passed to DLL, which establishes a connection with an adjacent network node. Its MAC sublayer forms MPDUs, each one with a payload limit that we denote by $L_k$. In case of larger segments, they are split into $\left\lfloor \frac{L_k}{L_k} \right\rfloor$ MPDUs, where $g_k$ is the payload reaching DLL — it includes the segment with its header, $h^{IP}$ and IP header, $h^{IP}$. Out of these MPDUs, $\left\lfloor \frac{L_k}{L_k} \right\rfloor$ are full while the last one carries $g_k - L_k \cdot \left\lfloor \frac{L_k}{L_k} \right\rfloor$ segment payload. In this
case, segment payload is transmitted over the MAC sublayer, so that $\tau_{\text{mac}}(g_k)$ and $\tau_{\text{mac}}(\text{ACK}_{k})$ are the times to transmit the segment and its acknowledgement.

**Mean MPDU size**

The three delay types are defined for the reference application. In general, the transport layer interleaves data segments and control segments to ensure an ordered data delivery service. At a given time, a source will have either of these two types of segments riding the channel. Time exclusiveness property allows us to assume that both segment types do not compete against each other for channel, rather segments from other sources. Thus, our model has a virtual source that can emit both types of segments. A model source represents corresponding application source, so that our system is composed of model sources. Each model source contends for the channel with a strength that we measure by the corresponding application period and the mean size of MPDU emitted. The contention strength is needed to quantify the MAC arbitration, which has a key contribution to delay variations — hence, required to estimate an average case.

The mean MPDU size, $E(l_k)$, is determined as follows. Given application message, $m_k$, our model source fragments it into a sequence of MPDUs. Let $d_{\text{data}}$ and $n_k$ denote total data sent in this sequence and the length of this sequence, then

$$E(l_k) = \frac{d_{\text{data}}}{n_k} \tag{29}$$

$d_{\text{data}}$ is composed of data segments and control segments, with each one including $h_{\text{dp}}$. As they are passed downwards, $h_{\text{dp}}$ and $h_{\text{mac}}$ are also added to them. A control segment is a data segment with zero payload: $\text{ACK}_{\text{dp}} = h_{\text{dp}}$. The process of $m_k$ fragmentation into MPDUs must consider the data segment payload limit. Let $f(z_k, L_k)$ and $e(z_k, L_k)$ map the data segment payload, $z_k$, and the MPDU payload limit, $L_k$, to total data sent in the related MPDUs and their count, respectively.

$$f(z_k, L_k) = z_k + h_{\text{dp}} + h_{\text{mac}}.$$

$$e(z_k, L_k) = \left\lfloor \frac{z_k + h_{\text{dp}} + h_{\text{mac}}}{L_k} \right\rfloor.$$

Thus,

$$d_{\text{data}} = \frac{m_k}{G_k} \cdot f(G_k, L_k) + \frac{m_k}{G_k} \cdot e(G_k, L_k) + \frac{m_k}{G_k} \cdot e(0, L_k).$$

$$n_k = \frac{m_k}{G_k} \cdot f(G_k, L_k) + e(G_k, L_k) + \frac{m_k - G_k}{G_k} \cdot L_k.$$

The time to transmit MPDU payload $E(l_k)$ is given by $\tau_{\text{mac}}(E(l_k))$, where

$$\tau_{\text{mac}}(l) = \left[ \frac{l}{L} \cdot f(L) \right] + \left[ \frac{l}{L} \right] \cdot L - l.$$

$$tt(L) = \frac{8}{8^{\text{ACK}} - 8} + \frac{8}{8^{\text{ACK}}} + SIFS + \frac{8}{8^{\text{ACK}}} + t_{\text{IP}}^{\text{phy}}.$$
Kostas Triantafyllidis received his Computer Engineering Degree at the Technical University of Thessaly, Greece in 2010. He is currently a PhD candidate at the Eindhoven University of Technology in the Video Coding Architecture (VCA) group since 2011. His research interests include profiling, modeling, analysis, design space exploration and multi-objective optimization of component-based real-time distributed systems (CB-RTDS).

Waqar Aslam received M.Sc. in Computer Science from Quaid-i-Azam University Islamabad, Pakistan. In 2014, he obtained PhD at the System Architecture and Networking Research group at Eindhoven University of Technology, The Netherlands, where currently he is an affiliated researcher. His research interests lie in performance modeling of distributed systems with focus on Quality of Service management of wireless networks.

Egor Bondarev received his MSc degree in robotics and informatics from the State Polytechnic University, Belarus Republic, in 1997. In 2009 he obtained his PhD degree in Computer Science at Eindhoven University of Technology (TU/e), The Netherlands in the domain of real-time systems with a focus on performance predictions of component-based systems on multiprocessor architectures. Currently, he is a Research Scientist at the Video Coding and Architectures group, TU/e, focusing on such research areas as multi-modal sensor fusion, photorealistic 3D reconstruction of environments and SLAM systems. Egor Bondarev is involved in several European research projects and currently he is a TU/e project leader in the projects both addressing challenges of ultra-high definition devices, distributed coordination, Quality of Service in networked systems and schedulability analysis in real-time systems.

Peter H. N. de With graduated in Electrical Engineering (M.Sc., ir.) from Eindhoven University of Technology and received his PhD degree from University of Technology Delft, The Netherlands. From 1984–1997 he worked for Philips Research Eindhoven, where he worked on video compression and chaired a R&D cluster for programmable TV architectures as senior TV Systems Architect. From 1997–2000, he was full professor at the University of Mannheim, Germany, Computer Engineering, and chair of the group Digital Circuitry and Simulation. From 2000–2007, he was with LogicAMG in Eindhoven as a principal consultant Technical SW and distinguished business consultant. He was also part-time professor at the University of Technology Eindhoven, heading the chair on Video Coding and Architectures. In the period 2008–2010, he was VP Video (Analysis) Technology at CycloMedia Technology. Since 2011, he is assigned full professor Video Coding and Content Analysis at Eindhoven University of Technology and appointed scientific director Care & Cure Technology and theme leader Smart Diagnosis in the University Health program. De With is a national and international expert in video surveillance for safety and security and has been involved in multiple EU projects on video analysis and surveillance projects with the Harbor of Rotterdam, Dutch Defense, Bosch Security Systems, TRH-Security, etc. He is board member of Dutch Institute of Technology for Safety and Security (DITSS) and R&D advisor to multiple companies. De With is Fellow of the IEEE, has (co-) authored about 50 journal papers and book chapters and over 300 conference papers on video coding, analysis, architectures, 3D processing and their realization. He is the (co-)recipient of multiple awards, such as the SMPTE team Award for the DV standard, a Philips Innovation Award, various IEEE CES Chester Sall paper awards, VCIP and BMV paper award, best IEEE Transactions paper awards, and Eurobest paper award.

Johan J. Lukkien is head of the System Architecture and Networking Research group at Eindhoven University of Technology since 2002. He received M.Sc. and PhD from Groningen University in the Netherlands. In 1991 he joined Eindhoven University after a two years leave at the California Institute of Technology. His research interests include the design and performance analysis of parallel and distributed systems. Until 2000 he was involved in large-scale simulations in physics and chemistry. Since 2000, his research focus has shifted to the application domain of networked resource-constrained embedded systems. Contributions of the SAN group are in the area of component-based middleware for resource-constrained distributed systems and schedulability analysis in real-time systems.

PANORAMA and OMEGA utilisation of HHD and 3D imaging.