ECRTS 2013

9-12 July
Paris, France

Work-in-Progress Proceedings

July 09–12, 2013
Paris, France

Edited by Linh Thi Xuan Phan

Proceedings
Work-in-Progress Session

of the 25th Euromicro Conference on Real-Time Systems (ECRTS'13)

July 09–12, 2013
Paris, France

Edited by Linh Thi Xuan Phan

Message from the Work-in-Progress Chair

Dear Colleagues:

Welcome to Paris, and to the Work-in-Progress (WiP) Session of the 25th Euromicro Conference on Real-Time Systems (ECRTS'13)! This session is dedicated to promising new and ongoing research on real-time systems and applications. I am happy to present eight excellent WiP papers that cover innovative research from a spectrum of topics, including multicore scheduling and timing analysis, functionality-aware task scheduling, real-time network architecture, dependable and real-time wireless communications, and component-based design and model-based development. I am confident that many of the research contributions we feature here will appear as full-fledged conference and journal papers in the near future. The proceedings from this session will be published online via the ECRTS 2013 WiP website (http://ecrts.eit.uni-kl.de/index.php?id=wip1).

The primary purpose of the WiP session is to provide researchers with an opportunity to discuss their evolving ideas and to gather feedback from the real-time community at large. Due to time constraints, the presentations in this session can only provide a brief overview of the new creative ideas and interesting approaches of the selected research contributions. Nevertheless, I hope that you all will enjoy this session, and that you will find the ideas presented interesting. Most of all, I hope you will participate in stimulating discussions, exchange your ideas, and provide valuable feedback to the authors.

I would like to thank the members of the WiP session technical program committee, Björn Brandenburg, Gabriel Parmer, Martin Schoeberl, and Insik Shin, for their hard work in reviewing the papers. I would also like to thank the authors for their interesting contributions and their confidence in ECRTS as a means to improve and advance their research. Last but not least, special thanks go to the ECRTS'13 organizers, Laurent George, Stefan M. Petters, and Gerhard Fohler, for their support and guidance.

Linh Thi Xuan Phan
University of Pennsylvania

Work-in-Progress Chair
25th Euromicro Conference on Real-Time Systems (ECRTS'13)
July 2013

ECRTS'13 Work-in-Progress Technical Program Committee

PROGRAM COMMITTEE

- Björn Brandenburg, Max Planck Institute for Software Systems, Germany
- Gabriel Parmer, George Washington University, USA
- Martin Schoeberl, Technical University of Denmark, Denmark
- Insik Shin, KAIST, Korea

WORK-IN-PROGRESS CHAIR

Linh Thi Xuan Phan, University of Pennsylvania, USA

Table of Contents

Message from the Work-in-Progress Chair……..……..……. i

Technical Program Committee……..……..……..…… ii

On-Demand Coherent Cache for Parallelised Hard Real-Time Applications……. 1
Arthur Pyka, Mathias Rohde, Sascha Uhrig and João Fernandes

Response-Time Analysis of Fork/Join Tasks in Multiprocessor Systems…….. 5
Cláudio Maia, Luís Nogueira, Luis Miguel Pinho and Marko Bertogna

Scheduling with Functional and Non-Functional Requirements: The Sub-Functional Approach……..……..……..……..…….. 9
Luca Santinelli, Wolfgang Puffitsch, Claire Pagetti and Frederic Boniol

High Level Modeling for Real-Time Applications with UML & MARTE……... 13
Julio Medina and Alejandro Pérez Ruiz

An Arbitrarily Precise Time Synchronization Algorithm Based on Ethernet Switch Serialization……..……..……..……..…….. 17
Fabrice Frances, Ahlem Mifdaoui, Xavier Codogni and Christian Fraboul

End-to-End Timing Challenges in Seamless Tool Chain Development for Vehicular Embedded Real-Time Systems……..……..……..…….. 21
Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin

Extending Real-Time Networks over WiFi: Related Issues and First Developments…… 25
Tony Flores Pulgar, Jean-Luc Scharbarg, Katia Jaffrès-Runser and Christian Fraboul

Towards Resilient Real-Time Wireless Communications……..……..…. 29
Jeferson Souza and José Rufino

On-Demand Coherent Cache for Parallelised Hard Real-Time Applications

Arthur Pyka*, Mathias Rohde*, João Fernandes†, Sascha Uhrig*
*Technical University of Dortmund, Germany
{arthur.pyka, mathias.rohde, sascha.uhrig}@tu-dortmund.de
†Honeywell International, Czech Republic
joao.fernandes@honeywell.com

***Abstract*—The efficient execution of parallelised hard real-time applications on many-cores highly depends on the accessibility of shared data. Concurrent accesses to the same memory induce a bottleneck for the application execution. Usage of coherent cache architectures dislocates the bulk of memory transfers to the responsible caches while ensures the coherence of shared data. For hard real-time systems, these cache coherence strategies are not suitable, since inter-cache communication corrupts the predictability of the timing behaviour.**

In this paper we propose our ongoing work on the On-Demand Coherent Cache (ODC^2) for hard real-time capable many-core systems. This strategy is based on marginal hardware extension and the use of common synchronisation techniques. ODC^2 provides caching of private as well as shared data while ensuring coherent accesses to shared data without inter-cache communication. We present a preliminary performance evaluation using a parallelised hard real-time 3D Path Planing application from the avionic domain.

I. Introduction

Parallelisation of hard real-time applications is a complex challenge. In the desired case, discovering of the possibilities for task and data parallelism leads to a parallelised code and a significant speedup when executed on a multi- or many-core platform. But even if a parallelised application is available, more challenges lie ahead.

The selection of a suitable platform has an enormous impact on the effectiveness of the parallelisation. Concurrent accesses of multiple cores to a shared memory can induce a bottleneck for the application execution and decrease the desired speedup. On platforms with a mesh or torus interconnect, the long distance between a shared memory and cores located at several nodes can increase the impact of the bottleneck. In such systems, private caches must be used to allow fast data access, while cache coherence mechanisms are needed to preserve the coherence of accesses to shared data.

Many cache coherence protocols have been introduced in the last decades. A large part of the mechanisms are based on ideas developed in the 1980 and are successfully used in various systems. These snooping or directory based protocols have been successively improved and evaluated [1]. Another method to provide coherent accesses to the main memory is based on the virtual memory management functionality of the cores [2]. Similar techniques are presented by Park et al. [3] and Ros et al. [4]. A technique closely related to this work is self invalidation. Lebeck et. al. [5] proposes dynamic self-invalidation, in which blocks are automatically selected to be invalidated.

While providing adequate solutions for various systems, these coherence strategies are not suitable for hard real-time systems. In such systems, the usage of cache memory is problematic. The execution of critical tasks has to follow strict timing restrictions, otherwise an intolerable system failure can occur. To ensure that timing conditions are met, a static timing analysis is used, allowing the system designer to estimate an upper bound for the worst case execution time (WCET) of an executed task. For the determination of proper memory access latencies, the timing analysis requires a suitable knowledge of the behaviour of the memory system including cache memories.

A profitable timing analysis of a cache memory is a challenging task for itself, dependent on the predictability of the cache behaviour. But in multi-core environments, when cache coherence mechanisms come into play, reliable predictions of a cache's timing behaviour are impractical. Timing analysis needs sufficient knowledge about the state as well as the content of block frames at a specific time. Having this information, cache hits and misses can be predicted in order to forecast memory access latencies, and the lack of this knowledge leads to a large WCET overestimation.

The essential operations of common cache coherence protocols restrain the analysability of the timing behaviour for different reasons. In the following, the effect of coherence protocols on analysability is described.

II. Analysability of Cache Coherence Protocols

Cache coherence protocols for multi-cores rely on various interaction between the caches. The state and the content of a cache line can be modified externally via actions triggered by other cores. The following three scenarios demonstrate possible interferences between caches resulting in corruption of content and mutated timing behaviour:

1) **Validity of cached data:**
   In both types (invalidation-based as well as update-based) of coherence protocols, the lifetime of a specific cache line is hard to predict:

- A cache line containing data which is shared with other cores can become invalid if another core intends to write on that data.
- Even though the lifetime of any *private* cache line is not directly affected by shared data, the state of the replacement policy (e.g. Least-Recently-Used) can be influenced by external invalidations of block frames holding shared data.

As a result, the lifetime of any cache line not only depends on internal modifications but also on unpredictable invalidations caused by other cores. This makes the prediction of the cache's content practically impossible.

2) **Latency of cache misses while accessing shared data:** A cache miss of shared data typically generates a read burst from the main memory. Since the data in the main memory could be outdated and the modified data could be present in another core's cache, a previous write back procedure of the possessing core may be necessary. Hence, the latency of cache misses at accesses to shared data is mutable since it is not known when the other core is able to handle the request.
3) **Latency of cache misses while shared data is hold by the cache:** The execution time of a specific task can be influenced by another core. If a core is holding a modified cache line with shared data, it can be forced to write back that cache line to re-establish the validity of data in the main memory. This intervention can delay a local cache miss, resulting in an unpredictable miss latency.

All three scenarios show, that holding shared data in a core's cache makes the latency of future cache accesses unpredictable. This is caused by external modifications of the cache's state and content. Common cache coherence mechanisms, invalidation- and update based, snooping- as well as directory-based, all rely on these modifications. A coherence mechanism, suitable for hard real-time systems must be defined by the absence of external intervention. The timing behaviour has to be predictable from the core's point of view.

Figure 1 demonstrates three scenarios of the same memory access sequence using a conventional cache coherence protocol. Assuming a 2-way set-associative cache with Least-Recently-Used (LRU) replacement strategy and the cache line access order A, X, B, A, where the cache lines A and B hold private and X shared data. All accesses concern the same set. In the case X is not externally invalidated (see scenario a) or X is externally invalidated after the access to B (see scenario b), B will evict A and the next access to A will lead to a cache miss. If X is externally invalidated before the access to B (see scenario c), the free block frame is used for B and A stays in the cache. In addition to that *validity of cached data* (1), the latency at loading X can vary depending on the presence of a modified X in another cache (2). Moreover, the invalidation of X in the scenarios b and c can conflict with loading A and B, respectively, leading to fluctuating latency of the corresponding load.

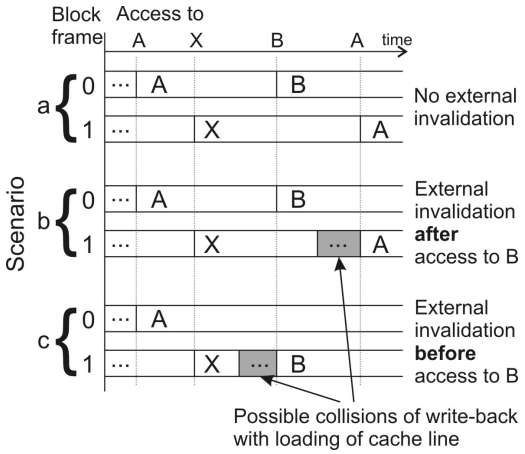


Fig. 1. Three scenarios of a 2-way set-associative cache using LRU replacement and the access sequence A-X-B-A where A and B are private and X shared data

```
..
-- Accesses to private data only --
..
lock(critical_section);
  enter_shared_mode();
  ..
  -- Accesses to shared data --
  ..
  exit_shared_mode();
unlock(critical_section);
..
-- Accesses to private data only --
..
```

Fig. 2. Example code snippet with switching ODC^2 mode inside a critical section

The *On-Demand Coherent Cache (ODC^2)* technique introduced by Pyka et al. [6] provides a predictable timing behaviour at the expense of an increased miss rate compared to the well-known MESI and MOESI coherence techniques. The mechanism is based on protected access to shared data and controlled invalidation of selected cache lines. Depending on the context of a cache miss, the corresponding block frame is regarded as private or shared. Block frames regarded as shared are invalidated automatically at specific execution points.

The ODC^2 technique is free of any interactions between caches, thus above mentioned scenarios does not appear. All modifications of a cache are induced by software and predictable as far as for incoherent caches. Apart from that, the mechanism can improve the availability of private data in set-associative caches, compared to conventional coherence mechanisms.

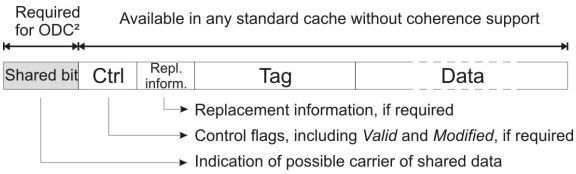


Fig. 3. Block frame of ODC^2 including the *Shared bit* which tags the block frame to be a possible carrier of shared data

III. The Time Predictable ODC^2

In this section a brief description of On-Demand Coherent Cache as presented in [6] is given.

The aim of ODC^2 is to provide a fast access to private as well as to shared data, the latter in a coherent way. To do this, shared data is hold only as long as necessary and is flushed back to main memory after the time of use. To identify instruction sequences that access shared data, the ODC^2 depends on critical sections or other synchronisation techniques which are anyway needed in parallel applications to protect accesses to shared resources and to show a deterministic behaviour. Therefore, these critical sections (using atomic lock-/unlock-operations) need to be extended by two operations that control the ODC^2 cache controller. The code snippet presented in Figure 2 shows an example of a critical section containing the control operations *enter_shared_mode()* and *exit_shared_mode()*.

An application using ODC^2 can take full advantage of caching (private) data in all code sections where no shared data is accessed. Therefore, ODC^2 supports two different working modes, the *private mode* and the *shared mode*. The activation and deactivation of the *shared mode* can be triggered either by accesses to cache control registers.
In private mode, the ODC^2 acts as a standard cache controller without any coherence functionality since in this mode no accesses to shared data are allowed. This mode is used as long as the processor has not entered a critical section.

At the time a critical section is entered, the shared mode is activated and the ODC^2 shows additional functionality. If a cache miss occurs, the loaded cache line is marked as shared by the *shared* bit. This additional information inside a block frame (see Figure 3) identifies the cache line as a potential carrier of shared data on which coherence actions are required. Seeing that some of the potential carriers can hold private instead of shared data, this private data is needlessly marked as shared. To minimise this negative effect, the address of an access is checked in shared mode. By doing this, it is possible to identify and to exclude specific memory sections from being marked (e.g. stack) or even restrict the marking to a special *shared* section.

At the time of deactivation of *shared mode*, all subsequent memory accesses are stalled and the cache controller performs a *restore procedure*. In case of a *write-back* write strategy, all cache lines marked as *shared* and *modified* are written back to the main memory. In case of a *write-through* policy, no additional write-back is required. Independent from the write policy, all cache lines marked as *shared* are invalidated. After this operation all modified shared data is flushed to the main memory and no shared data remains in the cache. Being free of shared data, the cache can perform future private data accesses more efficient. At the next access to the same shared data, it needs to be reloaded from the memory. Thus, shared data accessed by one processor is always consistent with the data accessed by other processors.

IV. Evaluation

The major requirement of a coherence technique suitable for hard real-time systems is a predictable timing behaviour. Performance improvements play a subsidiary role i. e. performance improvements with unpredictable timing behaviour are of no use for hard real-time systems. A common workaround used in current multi-core and many-core hard real-time systems is the application of non-coherent caches for private data only. Shared data has to be accessed without the help of caches.

Apart from the predictable timing behaviour, the ODC^2 has to provide a significantly improved performance compared to uncached accesses to shared data, to be a reasonable alternative. So, our evaluation is based on a comparison between the ODC^2 and the workaround using a hard real-time application from the avionic domain, executed on a mesh-based system.

A. 3D Path Planning Application

In this evaluation, a parallelised implementation of the *3D Path Planning* application from Honeywell is used. This application was provided in the context of the parMERASA project [7]. It is used for airborne collision avoidance that builds a path between an initial and goal waypoint, and its implementation is based on the Laplace's equation. The vehicle's surrounding space is represented by a 3 dimensional vector space, containing obstacles that must be avoided. The most computational intensive part of the application is highly suitable for parallelisation: the vector space can be divided into multiple compartments (i.e., 3-dimensional sub-grids) that can be processed in a pipeline scheme. All cores share the surrounding information (obstacle map and goal position) and cores with neighbouring compartments exchange data during computation, with inter-core synchronisation being done via locks and barriers. The code was extended with the required operations to control the ODC^2 protocol, whenever shared data is accessed (as described in Section III).

B. Evaluation Environment

The *parMERASA simulator*, a many-core simulation platform, developed in the parMERASA project [7], is used for this evaluation. It is based on the SoCLib simulation platform [8], a collection of SystemC hardware modules (processors, networks, memory, etc.). Our basic platform consists of multiple clusters, connected via routers in form of a mesh. A cluster itself is composed by different modules which are connected via a local crossbar.

For this evaluation, we set the platform to 3x3 mesh with one core per cluster, local memory in each cluster and a global

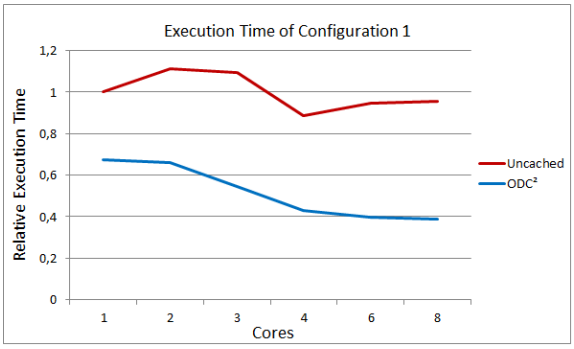


Fig. 4. Results of 3D Path Planning benchmark with configuration 1

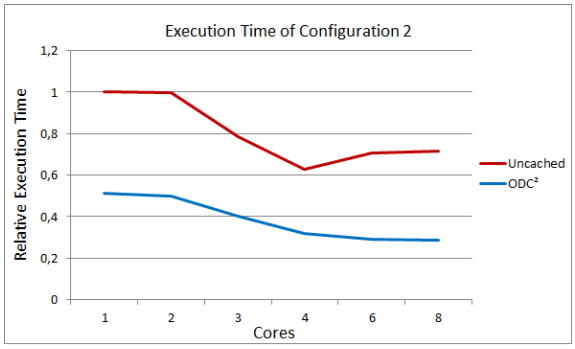


Fig. 5. Results of 3D Path Planning benchmark with configuration 2

memory (containing shared data and synchronisation variables) in the lower left cluster (first configuration), respectively in the middle cluster (second configuration). We evaluated platforms with 1 up to 8 cores involved in the parallel execution. The execution times of the ODC^2 are compared with the *uncached* execution. For ODC^2, private and shared data is cached while synchronisation variables remains uncached. Only private data is cached during the execution on the *uncached* platform.

C. Preliminary Results

In Figure 4 the execution time of the 3D Path Planning application executed with the first configuration is shown. The graph presents the results of parallel computation normalised to the *unshared* platform with 1 participating core.

An immense reduction of the execution time can be seen with ODC^2 compared to uncached. Using ODC^2, the application benefits from the higher parallelisation degree when increasing the number of cores. On the *unshared* platform, parallelisation benefits can be achieved only up to 4 cores. With more than 4 cores, the results starts obvert, caused by the higher amount of accesses to the shared memory. The exceptionally low execution time with one core is partly caused by the omitted synchronisation overhead, but mainly by the fast access to the shared memory that lies in the same cluster. To provide a fair chance for all cores to access the shared data without hindrance, in the second configuration the global memory is placed in the middle cluster. As seen in Figure 5, the parallel execution with up to 4 cores is considerably improved by the centralised memory. Still the execution with ODC^2 outperforms the uncached execution and is less sensible to layout variations.

V. Conclusion and Future Work

This paper presents a preliminary evaluation of the time predictable ODC^2 executing a parallelised hard real-time application from the avionic domain. The results fulfil the expectations of significant performance gain. While preserving the timing analysability of the system, ODC^2 offers improved performance for the parallelisation of hard real-time applications compared to uncached accesses.

Future work will focus on WCET analysis of the presented techniques together with more extensive evaluations on an increased number of cores, distributed on multiple clusters.

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no. 287519 (parMERASA).

References

[1] M. Tomasevic and V. Milutinovic, "Hardware approaches to cache coherence in shared-memory multiprocessors part 2," *IEEE Micro*, vol. 14, no. 6, pp. 61–66, Dec. 1994.

[2] X. Zhou, H. Chen, S. Luo, Y. Gao, S. Yan, W. Liu, B. Lewis, and B. Saha, "A case for software managed coherence in many-core processors," in *Proceedings of the 2nd Usenix Workshop on Hot Topics in Parallelism*. Usenix Assoc., 2010.

[3] J. Park, C. Jang, and J. Lee, "A software-managed coherent memory architecture for manycores," in *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 213–.

[4] A. Ros and S. Kaxiras, "Complexity-effective multicore coherence," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, ser. PACT '12. New York, NY, USA: ACM, 2012, pp. 241–252.

[5] A. Lebeck and D. Wood, "Dynamic self-invalidation: reducing coherence overhead in shared-memory multiprocessors," in *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, 1995, pp. 48–59.

[6] A. Pyka, M. Rohde, and S. Uhrig, "A Real-time Capable First-level Cache for Multi-cores," in *Workshop on High Performance and Real-time Embedded Systems (HiRES) in conjunction with HiPEAC'13, Berlin, Germany*, jan 2013.

[7] parMERASA - Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability, "http://www.parmerasa.eu/."

[8] "The SoCLib project. Mainpage." last download Nov. 17, 2012. [Online]. Available: http://www.soclib.fr/

Response-Time Analysis of Fork/Join Tasks in Multiprocessor Systems

Cláudio Maia, Luís Nogueira, Luis Miguel Pinho
CISTER-ISEP / INESC-TEC
Porto, Portugal
Email:{crrm, lmn, lmp}@isep.ipp.pt

Marko Bertogna
University of Modena
Modena, Italy
Email:{marko.bertogna}@unimore.it

***Abstract*—This paper proposes a model to analyse the response-time of parallel real-time tasks. The presented model is based on the fork/join model which is internally used by user-level frameworks to exploit the parallelism provided by the underlying architecture. The model considers tasks with fixed priorities and allows real-time jobs to generate an arbitrary number of parallel threads. Each parallel thread is scheduled at runtime by taking into account the timing properties of the job that spawns it.**

***Keywords*-Parallel Task Model, Job-level Parallelism, Real-Time**

I. Introduction

Nowadays most of the computer devices (e.g. PC, tablet, mobile phone) rely on multiple processors, and future generations of processors are expected to integrate thousands of simple processors into a single chip [1]. The turning point in the computer industry begun in 2001 when Sun Microsystems and IBM (in a separate effort) produced the dual-core processors, and later on in 2006 this type of processors became a mainstream technology powered by Intel and AMD.

The causes behind the paradigm shift are mostly concerned with the physical limitations of computer chips as an increase in the operating frequencies of the chips leads to an increase in power consumption as well as temperature. Therefore, and to overcome such limitations, instead of increasing the operating frequencies of the chips, chip manufacturers increased the number of computing units operating in parallel at lower frequencies.

The real-time and embedded systems domain was no exception to this shift. Industries such as automotive, aerospace, and avionic, design complex systems that require powerful hardware capable of supporting their functional and non-functional software requirements. It is therefore extremely important for each of these industries to incorporate new and state-of-the-art multiprocessor architectures in their products, not only because of the added computing capacity but also due to the size, weight, and power constraints of the hardware itself.

Traditional real-time applications are scheduled in a multiprocessor system by well-studied approaches, as it has been shown in a recent survey [2]. Nonetheless, the paradigm shift revealed, among others, that the problem of scheduling real-time tasks is no longer a problem of scheduling sequential tasks (i.e. without intra-task parallelism). New models for computing real-time tasks that consider *job-level parallelism* or *intra-task parallelism* should be employed to maximise the performance of the applications, and therefore the utilisation of the available processing capacity.

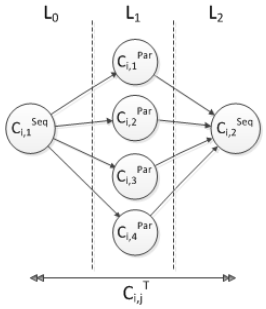


Figure 1. Job tree (DAG) of a job of Task τ_i

Frameworks such as Java Fork/Join [3] or OpenMP [4] help the application programmer divide the applications into sets of blocks which can then be scheduled in parallel in a multiprocessor architecture. Nevertheless, such parallelisation brings problems when application's timing constraints are considered, mainly because existing models are still restrictive (e.g. [5] and [6] transform a parallel task into a sequential task in order to apply well-known techniques used in traditional models).

The fork/join model is a model used by the above-mentioned frameworks to divide the applications into small blocks. In its basic form, the job of a task is composed of two sequential parts and a parallel part, as depicted in Figure 1. The first sequential part spawns several smaller units of execution that can be executed in parallel in order to exploit the inherent parallelism offered by multiprocessor architectures (in this paper these units are named *parallel jobs* or *p-jobs*). The number of parallel parts can be arbitrary large, as long as each parallel part is preceded by a sequential part and succeeded by another sequential part.

In this paper, the schedulability analysis of fork/join tasks from a response-time perspective is covered. We propose the

decomposition of a fork/join task into threads of execution, in order to improve the computation of the interference of each task on itself and on lower priority tasks. The proposed task model is composed of fixed-priority tasks where each real-time task can be a fork/join task, or a traditional sequential real-time task. The model assumes the graph of each task is known *a priori*, and the number of parallel jobs generated by each task can be greater than the number of cores in the platform.

The remainder of this paper is organised as follows. Section II presents the state of the art of parallel real-time tasks. Section III describes the system model. Section IV presents the possible approaches to perform response-time analysis of fork/join tasks. Finally, section V concludes the paper and presents the future work.

II. Related Work

Instead of considering a pure sequential task model, we consider the execution of parallel real-time tasks, i.e. job-level parallelism is allowed. In the domain of job-level parallelism, Goossens and Berten in [7] redefined a classification for different types of parallel tasks. Following this classification a job may be classified as rigid, moldable or malleable. A job is said to be rigid if the number of processors assigned to it is determined *a priori*, and this number does not change throughout job execution. A job is said to be moldable if the number of processors assigned to it is determined by the scheduler, but cannot change dynamically. Finally, a job is said to be malleable if the number of processors assigned to it is determined by the scheduler at runtime, and can change during the job's execution. Hence, a task is said to be: rigid if all of its jobs are rigid; moldable if all of its jobs are moldable; and malleable if all of its jobs are malleable. According to this classification, the parallel task model presented in this paper is considered to be composed of malleable tasks.

Malleable tasks were covered by Jansen [8], Collette et al. [9], and Korsgaard and Hendseth [10]. Jansen [8] focused on minimizing the makespan but without considering real-time constraints. Collette et al. [9] studied the problem of global scheduling of sporadic task systems on multiprocessors considering job-level parallelism. Korsgaard and Hendseth [10] proposed a sustainable schedulability test for malleable tasks scheduled with global Earliest Deadline First (EDF).

More recently, Lakshmanan et al. [5] and Saifullah et al. [6] focused on the study of scheduling fork/join tasks. Lakshmanan et al. [5] studied the scheduling of periodic real-time tasks that follow a fork-join structure on multiprocessor systems. In their proposed model, each parallel task is divided into a series of sequential and parallel segments, where all parallel segments must have the same number of threads and this number cannot be greater than the number of processors in the system. Moreover, the authors propose the task stretch transform algorithm in order to schedule fork/join tasks using traditional techniques. Saifullah et al. [6] present a synchronous task model for the scheduling of parallel real-time tasks with a fork-join structure. This model does not have any limitations on the number of parallel threads per segment and therefore is more general than [5]. The authors also proposed an algorithm to decompose the tasks into sequential tasks in order to use traditional schedulability analysis approaches.

III. System Model

We consider the problem of scheduling independent jobs on a system that comprises m identical processors with uniform memory access. In our model, a job is allowed to execute in more than one core at the same instant. A fully preemptive system is assumed where any job executing may be preempted at any instant and resumed later without any cost. At any given instant, the jobs with the highest priority among the ready jobs are the ones executing in the cores.

Let $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ denote the set of n periodic tasks. Each task τ_i in the task set τ is characterised by a period T_i, a worst-case execution time requirement C_i, and a relative deadline D_i. Each task releases an infinite sequence of jobs at periodic time intervals separated by at least T_i time units. Each job has an implicit deadline equal to $D_i = T_i$ and a worst-case execution time requirement equal to C_i.

During execution, a job of τ_i may spawn a set of k "sub-jobs", denoted by *parallel jobs* or *p-jobs*, $pJ_i = \{pJ_{i,1}, pJ_{i,2}, ..., pJ_{i,k}\}$. The parallel jobs are sequential threads that decompose the job's workload so that its execution can be performed in parallel, therefore having the advantage of being executed in different processors in the same time instant (see Figure 1). Thus, each job has a set of instructions that are executed sequentially, and may have a set of instructions that can be executed in parallel upon m processors, i.e. a sequential part and a parallel part.

Note that the worst-case execution time C_i is equivalent to the time it takes to execute a job of τ_i in a single processor without preemption, i.e. executing all p-jobs sequentially. Let $C_{i,s}^{Seq}$ be the sequential worst-case execution time of the s-th sequential part of task τ_i, and $C_{i,p}^{Par}$ be the worst-case execution time of the p-th p-job spawned by task τ_i . Then, $C_i = \sum_{s=1}^{h} C_{i,s}^{Seq} + \sum_{p=1}^{k} C_{i,p}^{Par}$, where h and k are the number of sequential and parallel parts of τ_i, respectively.

Each p-job instance $pJ_{i,p}$ inherits the timing properties from the job that spawns it. Thus, the p-th instance of a p-job is characterised by the same period T_i and relative deadline D_i of the parent job. In this model, parallel jobs are independent, and with the exception of the processors, there are no other shared resources or critical sections.

The task structure is represented by a directed acyclic graph (DAG), denoted as $G_j = (V, E)$, as depicted in Figure 1. Each element in the set of vertices V represents the sequential parts of a job and the p-jobs spawned during

the execution. Each vertex has an associated worst-case execution time. Each element in the set of edges E represents the communication path between two vertices, v_i and v_j in the set V, i.e. $v_i, v_j \in V$. The proposed model does not take into account any communication cost between any two nodes in the graph. Nevertheless, a partial order in the execution is imposed which is deemed correct from the relation that exists between a job and its spawned p-jobs.

The *minimum execution time* P_i of a job j is defined to be the longest execution path in the task graph from the root vertex to the leaves, i.e. the critical path length. Formally, P_i is defined as $P_i = \sum_{v \in L_l} max(C_{i,v}), l = 0, 1, ..., L$, where v represents the v^{th} vertex that is part of level L_l and L denotes the number of levels in the graph (see Figure 1).

The *utilisation* u_i of task τ_i is the ratio between the task's execution time and period, $u_i = \frac{C_i}{T_i}$. For the task set τ, the *total utilisation factor* is defined as $U(\tau) = \sum_{i=1}^{n} \frac{C_i}{T_i}$. For implicit-deadline sequential task sets, a necessary and sufficient condition for feasibility is $U(\tau) \leq m$ ([11]). Nevertheless, for fork/join tasks this condition is only necessary [5]. It is important to mention that the fork/join model allows a task to have a utilisation larger than 100% while assuring that the cumulative utilization of the task set is no greater than m. This property prevents the serialisation of certain task sets to the implicit-deadline sequential case, as it would deem these task sets unschedulable.

IV. Response-Time Analysis

The response-time of a job is the amount of time that elapses between the release of the job and its completion time. From a real-time systems perspective, guaranteeing that the response-time of the tasks in the task set does not exceed their deadlines for all possible arrivals of the tasks, assures that the system is schedulable.

Lakshmanan et al. [5] analyse fork/join tasks from a feasibility perspective and provides the best-case and worst-case fork/join task structure. The best-case task structure is composed of m p-jobs which can be executed in parallel by fully utilising the m cores provided by the platform, with a cumulative utilisation of the task set that does not exceed m. The worst-case structure is based on infeasible task sets with a cumulative utilization closer to 100%, regardless of the number of processors present in the platform. An example of such task sets is given by taking a fork/join task with an implicit deadline equal to the *minimum execution time* and a short parallel region spanning all the cores in the platform; when this task is scheduled along with a sequential task with an arbitrarily small utilization and a deadline equal to the parallel region of the fork/join task, a deadline can be missed with a cumulative utilisation close to 100%.

From a response-time analysis perspective, the worst-case response-time of a fork/join task does not only depend on the interference caused by other higher priority jobs, but also on the precedence constraints between serial and parallel stages

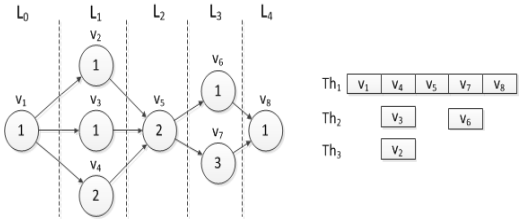


Figure 2. Decomposition approach example

of the task itself, as well as on the degree of parallelism of each stage and of the architecture.

In this paper we propose a global fixed-priority approach for fork/join tasks in which all the ready jobs/p-jobs are inserted in a global queue from where m processors pick the highest priority m jobs/p-jobs. In order to perform the response-time analysis of such tasks, we propose two possible approaches. The first approach considers for each job its execution time and the interference that it suffers from higher priority tasks. If the system is schedulable, the interference is bounded and the job execution time plus the imposed interference is always less than or equal to the job's deadline. If a job finishes its execution before its deadline, the available slack, given by the difference between the deadline D_i and the response time R_i, can be used to refine the computation of the worst-case interfering workload for other tasks, similarly to the method presented for sequential task sets in [12]. Moreover, a further refinement for fork/join tasks can be given by examining the response-time of each parallel stage of a task, allowing a tighter estimation of the interference imposed by such a task.

A second approach to response-time analysis is to consider a novel decomposition approach of a fork/join task, as depicted in Figure 2. In this approach, the fork/join task is decomposed into a set of threads of execution. There is a main thread of execution (Th_1 in the example) which is composed of all the sequential parts and the worst-case parallel part of each level L_i with an execution time of P_i. The remaining threads of execution (Th_2 and Th_3 in the example) are composed of sets of parallel jobs belonging to different levels in the graph, picking one parallel job from the remaining ones in each level L_i. The total number of threads of execution is given by the maximum out degree of all the sequential nodes in the graph. The algorithm that performs the decomposition is depicted in Algorithm 1.

Once the task decomposition into different threads is done, classic methods to bound the interfering contribution of each sequential thread can be applied, e.g., limiting the carry-in contributions to at most $m-1$ threads [13], limiting each thread interference to a task τ_i to at most $D_i - P_i + 1$ time-units [14], etc.

V. Conclusion

In this paper, we presented a model to analyse the response-time of fixed-priority fork/join tasks. Different

Algorithm 1 Task decomposition algorithm

```
v_i ← RootVertex(G_j)
function COMPUTE THREADS(v_i)
    Create new list
    backtrackNode ← v_i
    if v_i not visited then
        v_i ← visited
        add v_i to list
        VisitedNodes ← VisitedNodes + 1
    end if
    for each v_j in G_j such that v_i, v_j is an edge
and v_j is ordered in nonincreasing order by WCET
 do
        if v_j not visited then
            add v_j to list
            v_j ← visited
            v_i ← v_j
            VisitedNodes ← VisitedNodes + 1
        else
            if inDegree of v_j > 1 then
                v_i ← v_j
            end if
        end if
    end for
    if VisitedNodes < |V| then
        Compute threads ( backtrackNode )
    end if
end function
```

methods are proposed to improve the response-time analysis of such task systems, including the decomposition of each fork/join task into sequential threads of execution. Future work includes a complete schedulability analysis of such tasks, considering the worst-case situations that lead to the largest possible interference for each task. Moreover, we believe the proposed model can be easily adapted to support strict fork/join tasks, where nested parallelism is allowed, and other general parallel task models.

ACKNOWLEDGMENT

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within projects Ref. FCOMP-01-0124-FEDER-022701 (CISTER), ref. FCOMP-01-0124-FEDER-020447 (REGAIN) and ref. FCOMP-01-0124-FEDER-012988 (SENODS); also by FCT and by ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/88834/2012.

REFERENCES

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[2] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011.

[3] D. Lea, "A java fork/join framework," in *Proceedings of the ACM 2000 conference on Java Grande*, ser. JAVA '00, 2000, pp. 36–43.

[4] OpenMP, "Openmp," http://openmp.org/, Jun. 2011.

[5] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*, ser. RTSS '10, 2010, pp. 259–268.

[6] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real-Time Systems Symposium, IEEE International*, vol. 0, pp. 217–226, 2011.

[7] J. Goossens and V. Berten, "Gang ftp scheduling of periodic and parallel rigid real-time tasks," *CoRR*, vol. abs/1006.2617, 2010.

[8] K. Jansen, "Scheduling malleable parallel tasks: An asymptotic fully polynomial-time approximation scheme," in *Proceedings of the 10th Annual European Symposium on Algorithms*, ser. ESA '02, 2002, pp. 562–573.

[9] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Inf. Process. Lett.*, vol. 106, no. 5, pp. 180–187, May 2008.

[10] M. Korsgaard and S. Hendseth, "Schedulability analysis of malleable tasks with arbitrary parallel structure," *Real-Time Computing Systems and Applications, International Workshop on*, vol. 1, pp. 3–14, 2011.

[11] W. A. Horn, "Some simple scheduling algorithms," *Naval Research Logistics Quarterly*, vol. 21, no. 1.

[12] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, 2007, pp. 149–160.

[13] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, 2009, pp. 387–397.

[14] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of edf on multiprocessor platforms," in *Real-Time Systems, 2005. (ECRTS 2005). Proceedings. 17th Euromicro Conference on*, 2005, pp. 209–218.

Scheduling with Functional and Non-Functional Requirements: the Sub-Functional Approach

Luca Santinelli, Wolfgang Puffitsch, Claire Pagetti and Frederic Boniol
ONERA, Toulouse
name.surname@onera.fr

***Abstract*—The problem of finding a feasible task scheduling with both functional and non-functional requirements has risen to complexities never experienced before. In this paper we propose a functional task classification stage through which tasks are grouped together following their functional properties. Then the task scheduling problem is reduced into a group scheduling problem where inter-group and intra-group ordering are experienced.**

I. Introduction

Finding valid schedules can become problematic for todays embedded systems with large task sets that could include both functional requirements in the form of dependencies between tasks, and non-functional requirements in terms of timing constraints. On the one hand, the dependencies defeat traditional schedulability tests that assume independence between tasks. On the other hand, exact approaches face limitations when being applied to task sets with thousands of tasks [1], [2], [3].

The complexity to find a schedule that fulfills all the functional and non-functional constraints becomes increasingly harder, in particular because the underlying problem is NP-hard [4], [5]. Although attacked from different perspectives, i.e. [6], the complexity problem remains hard for realistic real-time systems with large task sets.

In order to make such sets tractable, it is necessary to reduce the complexity of the scheduling problem. In this work we propose to group tasks according to their functional requirements, and perform scheduling on these groups of tasks rather than individual tasks. Reducing the number of items to be scheduled reduces consequently the complexity of the scheduling problem to be solved; therefore makes tractable real-time systems with thousands of tasks.

A. Related Work

There are several exact approaches to finding off-line schedules for task sets with precedence constraints. An early approach comes from [1], which starts from a heuristic solution that is consistent with the functional requirements and then uses a branch-and-bound algorithm to improve the solution with regard to the non-functional requirements. Other approaches combine the functional and non-functional requirements and try to find a solution that satisfies both in a single step. In [2], the Petri nets are applied to model the scheduling of dependent tasks. Ekelin [3] explores the use of constraint programming to solve scheduling problems, with several optimizations to speed up the search for a valid solution.

Chetto et al. [7] first considered the effect of precedence constraints between tasks on the dynamic priority scheduling problem. Spuri et al. [8] extended formal results on precedence constrained tasks to arbitrary timed tasks with preemption. Cucu-Grosjean et al. have tackled with the scheduling problem with a graph representation for the functional constraints [9], [10]. Those papers are inspired, among the others, by Clark [11] which outlines the task dependence problem together with the complexity of the scheduling problem when the dependences are dynamic, and by the work of Natale et al. [4] where end-to-end timing constraints are applied to guarantee distributed processes.

While these approaches can be fairly efficient for some cases, they cannot escape the fact that the underlying scheduling problem is a hard problem.

II. Problem Statement & Backgrounding

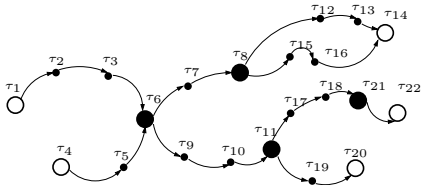
In this paper we tackle with the real-time scheduling problem by classifying tasks into groups according to their functional requirements. We call our approach *sub-functional scheduling*, as it is composed by the "functional grouping" and a series of future steps toward the schedulability analysis including both functional requirements and non-functional requirements, e.g. timing constraints.

Our sub-functional scheduling approach is developed assuming a) the functional constraints as data dependencies and precedence constraints described with graphs, b) off-line, non-preemptive and single-processor scheduling, c) mono-rate, synchronous tasks with the same period. In future work we intend to release some of the assumptions made.

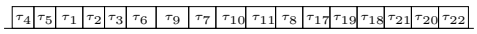
Example II.1. *Figure 1 shows our reference example to explain the approach under development. The* 22 *tasks have functional requirements described in Figure 1(a) with the precedence dependences as edges of the graph. Figure 1(b) shows an example of task scheduling compliant to the functional requirements.*

A. Functional Backgrounding

A real-time system can be seen as a task set $\Gamma = \{\tau_1, \tau_2, \dots \tau_n\}$ where each composing task τ_j is described



(a) Functional dependencies through data flow and graph representation



(b) An example of a possible task execution ordering for the tasks within the graph

Figure 1. Task functional dependence description and a possible scheduling.

by a tuple (a_j, C_j, T_j) with C_j being the task worst-case execution time, T_j the inter-arrival time of the task instances (jobs) and a_j the task activation which repeats at any of its instance. All the tasks of Γ are assumed with the same period to approach safety-critical avionic platforms. Recent works have shown that in safety-critical avionic platforms the strict time-oriented approach is applied with mono and multiple-rate tasks, [12], [13]. In our first stage we start with the mono-rate case to converge to the multiple-rate case in the future. Thus all the tasks share the same period T_j.

In our model, the precedence constraints (i.e. data dependences) between tasks are described as a directed acyclic graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ where $\mathbb{V}$ is the set of tasks Γ, $\Gamma \equiv \mathbb{V}$, and $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ is the set of edges which represents the precedence constraints between tasks.

Although classical real-time system models assume task execution as an infinite sequence of tasks instances (jobs), we keep our model within the finite directed graph scenario considering the tasks, not the jobs.

1) Graph Modeling: Given a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, an edge between two tasks τ_j and τ_k, $\tau_j \rightarrow \tau_k$, represents a precedence constraint between τ_j and τ_k.

A path $p(\tau_i, \tau_o)$ from task τ_i to task τ_o within a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is an alternating sequence $\langle \tau_i, \tau_i \rightarrow \tau_1, \tau_1, \ldots, \tau_n \rightarrow \tau_o, \tau_o \rangle$ of vertices and distinct edges where $\tau_i, \tau_1, \ldots, \tau_n, \tau_o \in \mathbb{V}$ and $\tau_i \rightarrow \tau_1, \ldots, \tau_n \rightarrow \tau_o \in \mathbb{E}$. We denote by P the set of all the paths in $\mathbb{G}$ and we conclude that two tasks τ_k and τ_j are *connected* if there is at least one path $p(\tau_k, \tau_j) \in P$. As the number of vertices and edges is finite and the graph is acyclic, P is a finite set; by $P(\tau_k, \tau_j)$ we denote the set of all the paths from τ_k to τ_j. $M(\tau_k, \tau_j)$ instead is the set of tasks belonging to all the possible paths from τ_k to τ_j.

The set of predecessors of a task τ_i is denoted by $preds(\tau_i)$ such that $preds(\tau_j) \stackrel{def}{=} \{\tau_k \mid \tau_k \rightarrow \tau_j\}$, and the number of successors $succs(\tau_i)$, as $succs(\tau_j) \stackrel{def}{=} \{\tau_k \mid \tau_j \rightarrow \tau_k\}$. The cardinality of a task is the sum of the number of predecessors and successors, $\|\tau_k\| = \|preds(\tau_k)\| + \|succs(\tau_k)\|$.

B. Functional Abstraction

The functional description implies a partial ordering between tasks, but usually allows more than one valid schedule. We can define equivalence among scheduling as follows: *two scheduling resulting from the same partial ordering are functionally equivalent*. Then the objective of the scheduling problem is to find in a total ordering of task such that complies to both functional and non-functional requirements.

Definition II.2 (Direct Dependence $\succ$)**.** *Given a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, a task τ_k directly depends on a task τ_j, $\tau_k \succ \tau_j$ if $\tau_j \rightarrow \tau_k$.*

The direct dependence relation can be extended transitively to a notion of *functional dependence*.

Definition II.3 (Functional Dependence $\rhd$)**.** *Given a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, a task τ_k* functionally depends *on a task τ_k, $\tau_k \rhd \tau_j$ if there exists at least one path $p(\tau_k, \tau_j) \in \mathbb{P}$ connecting τ_k and τ_j within $\mathbb{G}$.*

The functional dependence is a transitive relation between tasks; indeed, if $\tau_k \rightarrow \tau_j$ and $\tau_j \rightarrow \tau_r$, then $\tau_k \rhd \tau_r$. Thus the direct dependence is a stricter definition of dependence $\succ \subseteq \rhd$, which does not include the transitivity property.

Definition II.4 (Functional Independence $\overline{\rhd}$)**.** *The notion of independence is the opposite (the negation) of the dependence, $\overline{\rhd}$. Two tasks τ_j and τ_k are called independent, $\tau_j \overline{\rhd} \tau_k$ if $\nexists\ p(\tau_j, \tau_k) \in \mathbb{P}$.*

III. Sub-Functional Grouping

To simplify the scheduling problem with functional dependencies we propose to classify tasks according to their dependencies and create groups of tasks.

Definition III.1 (Grouping)**.** *Given a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, a grouping $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$ divides the task set into disjoint subsets such that $\forall \mathcal{G}_i, \mathcal{G}_j \in \mathcal{G},\ \mathcal{G}_i \cap \mathcal{G}_j = \emptyset$ and $\bigcup_{i=1}^{n} \mathcal{G}_i = \mathbb{V}$.*

The direct and functional dependence can be extended from tasks to groups of tasks by considering groups dependent if there is a dependence between any of their tasks:

$$\mathcal{G}_i \succ \mathcal{G}_k \Leftrightarrow \exists \tau_j \in \mathcal{G}_i, \tau_l \in \mathcal{G}_k, \tau_j \succ \tau_l \quad (1)$$

$$\mathcal{G}_i \rhd \mathcal{G}_k \Leftrightarrow \exists \tau_j \in \mathcal{G}_i, \tau_l \in \mathcal{G}_k, \tau_j \rhd \tau_l \quad (2)$$

The criteria for the functional grouping may be chosen arbitrarily, and some grouping are more helpful with regard to scheduling than others. In the following, we focus on two classes: a) *independence grouping* which exploits the notion of independence to partition the task set and b) *dependence grouping* which creates groups of dependence tasks.

A. Independence Grouping

Definition III.2 (Independence Grouping)**.** *We call a grouping $\mathcal{I}$ an independence grouping if only independent tasks belong to the same partition $\mathcal{I}_i$,*

$$\forall \mathcal{I}_i \in \mathcal{I}, \forall \tau_j, \tau_k \in \mathcal{I}_i, \tau_j \overline{\rhd} \tau_k. \quad (3)$$

This definition does not infer a unique grouping. A trivial independence grouping would be a grouping where each

Algorithm 1 Independence grouping algorithm

Input: Γ
Output: $\mathcal{I}_i$
1: $i \leftarrow 1, \mathcal{R} \leftarrow \Gamma$
2: **while** $\mathcal{R} \neq \emptyset$ **do**
3: $\quad \mathcal{I}_i \leftarrow \{\tau_j \mid preds(\mathcal{R}, \tau_j) = \emptyset\}$
4: $\quad \mathcal{R} \leftarrow \mathcal{R} - \mathcal{I}_i$
5: $\quad i \leftarrow i + 1$
6: **end while**

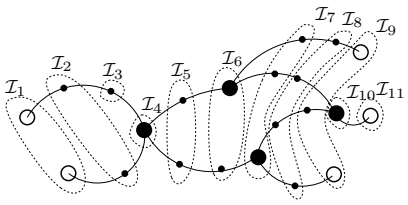


Figure 2. An Independence grouping example.

partition contains exactly one task. Obviously, such a grouping would not be particularly helpful. However, even when considering non-trivial cases, grouping according to independence allows for some ambiguity.

Algorithm 1 shows an algorithm to create an independence grouping by moving forward through the task graph. In the first step, it puts all tasks without predecessor tasks (begin tasks) into the same group. As none of these tasks have predecessors, it cannot be the case that $\tau_i \rhd \tau_k$ and the group consists only of independent tasks. The algorithm then removes these nodes and the related edges from the graph and creates a group which contains tasks without predecessors in the new task graph. The same reasoning as before applies, and all tasks in the group are independent. While removing tasks from the graph no relevant edges are removed; only the precedent tasks are evicted and the dependences among remaining tasks remain unaffected. Therefore, Algorithm 1 creates a valid independence grouping, and Figure 2 shows the result of applying the independence algorithm to Example 1. Among the other possibilities, there is also the backward approach to the grouping by moving backward to the graph.

B. Dependence Grouping

Instead of grouping independent tasks, we can group tasks that form chains of dependent tasks.

Definition III.3 (Chain)**.** *Two tasks τ_i and τ_k form a chain if τ_k is the only successor of τ_i and τ_i the only predecessor of τ_k, or if there exists a sequence of chains between τ_i and τ_k through intermediate tasks.*

$$\tau_i \rightsquigarrow \tau_k \stackrel{def}{=} (succs(\tau_i) = \{\tau_k\} \wedge preds(\tau_k) = \{\tau_i\}) \vee (\exists \tau_l, \tau_i \rightsquigarrow \tau_l \wedge \tau_l \rightsquigarrow \tau_k) \quad (4)$$

Definition III.4 (Dependence Grouping)**.** *A* dependence grouping $\mathcal{D}$ *is a partitioning of the task set such that chains of dependent tasks belong to the same group $\mathcal{D}_i$*

$$\forall \mathcal{D}_i \in \mathcal{D}, \tau_j, \tau_k \in \mathcal{D}_i, j \neq k \Leftrightarrow \tau_j \rightsquigarrow \tau_k \vee \tau_k \rightsquigarrow \tau_j. \quad (5)$$

Algorithm 2 Dependence grouping algorithm

Input: Γ
Output: $\mathcal{D}_i$
1: $i \leftarrow 1, \mathcal{R} \leftarrow \Gamma$
2: **while** $\mathcal{R} \neq \emptyset$ **do**
3: $\quad T \leftarrow \{\tau_i \mid preds(\mathcal{R}, \tau_i) = \emptyset\}$
4: $\quad$ **for** $\tau_j \in T$ **do**
5: $\quad\quad \mathcal{D}_i \leftarrow \{\tau_j\} \cup \{\tau_k \mid \tau_j \rightsquigarrow \tau_k\}$
6: $\quad\quad i \leftarrow i + 1$
7: $\quad$ **end for**
8: $\quad \mathcal{R} \leftarrow \mathcal{R} - \bigcup_{j=1}^{i} \mathcal{D}_j$
9: **end while**

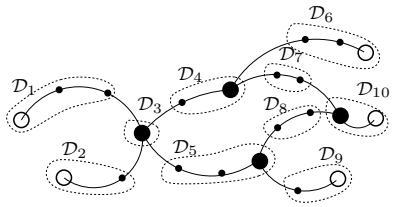


Figure 3. The only possible grouping with dependences.

Algorithm 2 computes the dependence grouping. It starts from the tasks without any predecessors and iterates over them creating a group for each of these, which includes the task and all those that form a chain with it. Afterwards, it removes the tasks that already are part of a group and continues until the task graph is empty.

Algorithm 2 is a *correct* dependence grouping since it explores the whole graph for the largest dependence groups, Equation (5). It dominates all the possible dependence grouping which start from intermediate or leaf tasks.

Figure 3 is for the result of the dependence grouping to Example 1 with 10 groups.

IV. Scheduling

The partial order established by the functional dependences is transformed into a total order by the scheduling.

A. Sub-Functional Scheduling

For a system of tasks Γ, a schedule S is a totally ordered set of activation times of all the tasks $S = \{a_j \in \mathbb{N}\}, \tau_j \in \mathbb{V}$ such that all the precedence constraints are satisfied [9]. $\mathbb{S}$ is the set of all the possible schedules. With the group decomposition, for a scheduling problem we can differentiate two "levels" of scheduling. There is the *inter-group* scheduling, as the ordering of groups, and the *intra-group* scheduling, as the ordering of the tasks composing each group.

Example IV.1. *Taking the reference example from Figure 1, we have that the first block of the graph composed by the tasks $\tau_1, \tau_2, \tau_3, \tau_4$ and τ_5 can results into 10 possible schedules:*

$$\begin{array}{ll} \tau_1 - \tau_2 - \tau_3 - \tau_4 - \tau_5 & \tau_1 - \tau_2 - \tau_4 - \tau_5 - \tau_3 \\ \tau_1 - \tau_4 - \tau_5 - \tau_2 - \tau_3 & \tau_1 - \tau_4 - \tau_2 - \tau_5 - \tau_3 \\ \tau_1 - \tau_4 - \tau_2 - \tau_3 - \tau_5 & \tau_1 - \tau_2 - \tau_4 - \tau_3 - \tau_5 \\ \tau_4 - \tau_5 - \tau_1 - \tau_2 - \tau_3 & \tau_4 - \tau_1 - \tau_5 - \tau_2 - \tau_3 \\ \tau_4 - \tau_1 - \tau_2 - \tau_5 - \tau_3 & \tau_4 - \tau_1 - \tau_2 - \tau_3 - \tau_5 \end{array}$$

All of them are compliant to the functional requirements, although they differ in terms timing.

1) Independence Scheduling: With an independence grouping as computed by Algorithm 1, the scheduling of groups is already fixed. The algorithm create groups greedily, such that groups must be executed in the same order as they were created for the algorithm. However, as all tasks are independent, the execution order of tasks within a group can be chosen freely.

Example IV.2. *From Example II.1 with a forward independence grouping it is* $\Gamma = \{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, \mathcal{I}_4, \mathcal{I}_5, \mathcal{I}_6, \mathcal{I}_7, \mathcal{I}_8, \mathcal{I}_9, \mathcal{I}_{10}, \mathcal{I}_{11}\}$ *where the group ordering is already decided as* $\mathcal{I}_1 - \mathcal{I}_2 - \mathcal{I}_3 - \mathcal{I}_4 - \mathcal{I}_5 - \mathcal{I}_6 - \mathcal{I}_7 - \mathcal{I}_8 - \mathcal{I}_9 - \mathcal{I}_{10} - \mathcal{I}_{11}$*, Figure 2. The total ordering is obtained selecting for each set an order for the composing tasks.*

This results into $2!2!1!1!2!2!3!4!3!1!1! = 13824$ *possible task combinations tough, still a complex problem.*

Considering the subgraph $\mathcal{I}_1 - \mathcal{I}_2 - \mathcal{I}_3$*, there are* 4 *possible orderings with independence grouping*

$$\tau_1 - \tau_4 - \tau_2 - \tau_5 - \tau_3 \qquad \tau_1 - \tau_4 - \tau_5 - \tau_2 - \tau_3$$
$$\tau_4 - \tau_1 - \tau_2 - \tau_5 - \tau_3 \qquad \tau_4 - \tau_1 - \tau_5 - \tau_2 - \tau_3$$

out of the 10 *possible without any grouping applied. This is the degree of flexibility we lose by applying grouping while reducing the complexity of the problem.*

The independence grouping reduces the possible task schedules, which in the worst case could eliminate all schedules that would satisfy the timing requirements. However, the grouping does not add any additional schedules, and is therefore safe with regard to the functional requirements.

2) Dependence Scheduling: With the dependence grouping paradigm, the ordering of tasks within a group is fixed by the functional constraints. In contrast, the scheduler has to find a suitable ordering of the groups of dependent tasks. Within this paradigm, the scheduling remains a mixture of functional and execution ordering with less flexibilities compared to the case without grouping. Like independence grouping, dependence grouping is safe with regard to the functional requirements, but may eliminate all schedules that would satisfy the timing constraints.

Example IV.3. *The resulting dependence scheduling for Example II.1 derives from the following set of combinations* $\{\mathcal{D}_1, \mathcal{D}_2\} - \mathcal{D}_3 - \{\mathcal{D}_4, \mathcal{D}_5\} - \{\mathcal{D}_6, \mathcal{D}_7, \mathcal{D}_8, \mathcal{D}_9\} - \mathcal{D}_{10}$*, where the relative order between* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *does not affect the functional requirements as well as for* $\mathcal{D}_4$ *and* $\mathcal{D}_5$ *and* $\mathcal{D}_6, \mathcal{D}_7, \mathcal{D}_8$ *and* $\mathcal{D}_9$*. It results into* $2!1!2!4!1! = 96$ *possible ordering, Figure 3.*

Considering the partition $\{\mathcal{D}_1, \mathcal{D}_2\} - \mathcal{D}_3$*, with a dependence grouping, we have* 2 *possible ordering depending on which between* $\mathcal{D}_1$ *or* $\mathcal{D}_2$ *is scheduled first, thus*

$$\tau_1 - \tau_2 - \tau_3 - \tau_4 - \tau_5 \qquad \tau_4 - \tau_5 - \tau_1 - \tau_2 - \tau_3$$

as a subset of the 10 *possible without any grouping. We notice that with a dependence grouping we further lose scheduling possibilities in that particular configuration.*

V. Conclusions

In this paper we introduced the notion of grouping to classify tasks with respect to their functional requirements. We proposed two different grouping policies according to the notion of functional dependence and functional independence.

In future stages, we intend to combine the grouping with schedulability analysis for timing constraints in terms of latencies. Extensions to this work will also release some of the assumptions made i.e., non preemptability or mono-rate, to generalize both the grouping approach and the consequent timing analysis.

References

[1] J. Xu and D. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations," *IEEE Trans. Softw. Eng.*, 1990.

[2] E. Grolleau and A. Choquet-Geniet, "Off-line computation of real-time schedules by means of petri nets," in *Workshop On Discrete Event Systems, WODES*, 2000.

[3] C. Ekelin, "An optimization framework for scheduling of embedded real-time systems," Ph.D. dissertation, Chalmers University of Technology, 2004.

[4] M. D. Natale, M. Di, N. John, and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proceedings of RealTime System Symposium, (RTSS)*, 1994.

[5] S. Baruah and J. Goossens, "Scheduling real-time tasks: Algorithms and complexity," *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 2003.

[6] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Trans. Comput.*, 2008.

[7] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints," *Real-Time Systems*, 1990.

[8] M. Spuri and J. Stankovic, "How to integrate precedence constraints and shared resources in real-time scheduling," *Computers, IEEE Transactions on*, 1994.

[9] L. Cucu-Grosjean and Y. Sorel, "Non-preemptive scheduling algorithms and schedulability conditions for real-time systems with precedence and latency constraints," INRIA Rocquencourt, Tech. Rep., December 2004.

[10] L. Cucu, R. Kocik, and Y. Sorel, "Real-time scheduling for systems with precedence, periodicity and latency constraints," in *Proceedings of Real-time and Embedded Systems*, 2002.

[11] R. K. Clark, "Scheduling dependent real-time activities," School of Computer Science, Carnegie Mellon University, Tech. Rep., 1990.

[12] F. Boniol, H. Cassé, E. Noulard, and C. Pagetti, "Deterministic execution model on cots hardware," in *Proceedings of the 25th international conference on Architecture of Computing Systems*, ser. ARCS, 2012.

[13] M. Lauer, J. Ermont, F. Boniol, and C. Pagetti, "Latency and freshness analysis on ima systems," in *Emerging Technologies Factory Automation (ETFA) IEEE 16th Conference on*, 2011.

High level modeling for Real-time applications with UML & MARTE

Julio L. Medina and Alejandro Pérez Ruiz
Departamento de Electrónica y Computadores, Universidad de Cantabria. Santander, Spain
{julio.medina, alejandro.perezruiz}@unican.es

***Abstract*—This paper shows initial results and the research path in a methodology to use UML & the UML Profile for MARTE in the design of real-time applications. The modeling constructs used are those proposed in the High Level Application Modeling chapter of the MARTE standard. These elements are at a high abstraction level, and hence they need to be complemented with a number of constraints and rules of usage in order to get a consistent set of transformations to obtain code and analysis models automatically from them. The rules and patterns proposed in this effort are meant to address increasingly complex design intents. As a starting point in the methodology this paper shows some of the basic ones, concretely the simple independent tasking model, the passive protected data sharing, and the distributed end-to-end flows of linear execution. The models here defined are suitable to be transformed into both: schedulability analysis models and code generation models. These models are also represented in UML as a previous step to its execution, the profiling of its execution times, and the schedulability analysis.**

Keywords—code generation; modeling; UML; MARTE; model-based schedulability analysis; MAST; Ada; real-time.

I. INTRODUCTION

Model-based software development is progressively taking momentum in industry as one of the most promising software engineering approaches. It helps to create and keep assets of many kinds along the development process. It facilitates the separation of concerns, increasing the process efficiency, and finally empowering the quality of software.

For real-time applications, a model-based methodology can also help to simplify the process of building the temporal behavior analysis models. These models constitute the basis of the real-time design and the schedulability analysis validation processes. With that purpose, the designer must generate, in synchrony with the models used to generate the application's code, an additional parameterizable model, suitable for the timing validation of the system resulting out of the composition of its constituent parts. The analysis model for each part abstracts the timing behavior of all the actions it performs, and includes all the scheduling, synchronization and execution resources information that is necessary to predict the real-time qualities of the applications in which such part might be integrated. In the approach here presented, these analysis models are automatically derived from high level design models annotated with a minimum set of real-time features taken from the requirements of the application in which they are to be used. Following the generation of the application's code as a composition of the code of its constituent parts, the complete real-time analysis model of the application can also be automatically generated from the composition of the set of real-time sub-models that form it.

The research effort that this paper presents considers the model-based development of hard real-time applications, for which the definition of the corresponding schedulability analysis models is an automated result of a chain of tools and techniques used in a model driven engineering approach. Our previous efforts in this direction can be read in [1]. In this context, this paper proposes the concrete modeling elements at a high level of abstraction, useful to conceive and elaborate the system using the Unified Modeling Language (UML) [2]. This is a general purpose modeling language standardized by the Object Management Group (OMG). This is used in conjunction with its standard extensions for Modeling and Analysis of Real-Time and Embedded systems, namely the UML Profile for MARTE [3]. There are other model driven similar efforts from the software engineering perspective derived from the ASSERT project, in [4] for example UML is used, though not fully based in standard modeling extensions like MARTE.

The most widely known use of model based development techniques comprises the generation of code from structural models like class diagrams. With those automations an initial set of skeletons of the classes and structural packages that form an application is usually easy to obtain. Also some form of reverse engineering is available through the usage of specially formatted "comments" placed as textual marks surrounding the space in the code files for the "bodies" of the operations. The final implementation code is then inserted (usually typed by hand) between the textual marks that are managed by the code generators. A further refinement that generates both, specifications and bodies from models, are code generators that use state machines for modeling the behavior of the classes. This mechanism uses the operations of a class as message handlers that trigger the events between states. That way the messages from other objects can interact with the automaton of the class, though in a non-predictable order. Then, this kind of code generators is not consistent with the required scenario-based description of real-time activities used for schedulability analysis.

For this reason a different approach to the code generation is necessary if we want to keep both models in tune in a way as automated as possible. Our tactic for generating the code that goes inside the marks of the structural skeletons is the use of the behavioral models given for each operation of the class.

This work has been partially funded by the Spanish Government under grant TIN2011-28567-C03-02 (HI-PARTES). This work reflects only the author's views; the funding organism is not liable for any use that may be made of the information contained herein.

These models are usually made just for descriptive or documentation purposes, but there is no reason for not using them precisely as a specification. For this labor the more adequate modeling elements are activity diagrams. The formalization of the textual code inside actions may be either the standardized action language [5] of the OMG, or specific annotations made in the target language that specify the concrete actions to be performed.

In the context of the methodology proposed in this approach, this paper contributes to clarify the process to use from a software engineering point of view, and to define the input modeling formalisms, using UML and MARTE for expressing the needs of the designer.

The paper is organized as follows: Section 2 presents a global view of the approach and situates the contribution of this work-in-progress paper in its perspective. It also makes a brief summary of the challenges, and presents related efforts. Section 3 presents the concrete modeling elements and rules used for modeling applications compliant to (A) the simple independent tasking model, (B) the passive protected data sharing and (C) end-to-end flows of linear chains of execution. Finally some conclusions and next steps to follow in our envisioned model based engineering approach.

II. CONTEXT OF THE MODEL-BASED APPROACH

As early mentioned, here we use UML as modeling language and the UML standard extensions proposed by the MARTE profile for annotating the necessary real-time aspects at different levels of specification. A synthetic view of the approach is shown schematically in Fig. 1.

The initial model used to describe the application and its real-time features is constructed using the MARTE extensions for high level application modeling (HLAM). From this formalism, two model-to-model (M2M) transformations are used. One, indicated as M2M_A in Fig. 1, is used to create the UML representation of the analysis model. This transformation is used to create a model for each real-time situation under analysis together with the model of the processing resources, and the workload to consider. For this model the schedulability analysis modeling (SAM) capabilities of MARTE are used. The other transformation, M2M_C, is used to generate an intermediate model ad hoc for the code generation. The intermediate model, called UMLforCode in Fig. 1, is a typical

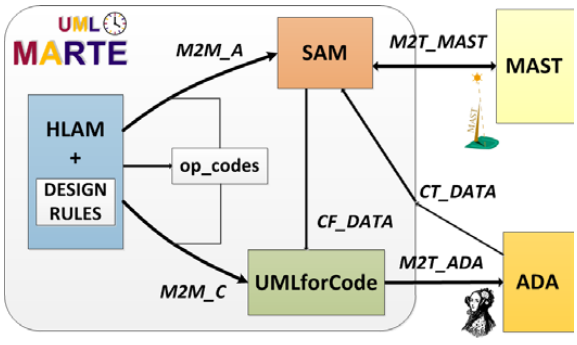


Fig. 1. Models and transformations used in our approach

UML object oriented generic model that comprises structural as well as behavioral information. The behaviors of the operations in this model are expressed by means of activity diagrams.

The model-to-text (M2T) transformation, denoted as M2T_MAST in Fig. 1, is used to generate the schedulability analysis models. It is part of our previous work [1]. An eclipse based tool [6] is available for the generation of analysis models, the invocation of the analysis tools, and the retrieval of results back into the modeling context. The tool then converts SAM models into the formalisms used by MAST [7] and then recovers its results back into the UML+MARTE model.

Another tool is also provided for generating Ada code from the UMLforCode object oriented generic model [8]. This is a model-to-text transformation, called M2T_ADA in Fig. 1. The code implemented out of the combination of M2M_C and M2T_ADA is consistent from the execution semantics point of view with the analysis models generated out of the combination of M2M_A and M2T_MAST. The transformation that generates the UMLforCode model includes the necessary instrumentation code that is used to measure and recover the approximate values for the worst, best and average execution times into the analysis model (a process called CT_DATA in Fig. 1). The op_codes table will help the transformations and tools to keep track of sections of code instrumented. Once the analysis is performed, scheduling analysis results are back annotated to the SAM models. These real-time configuration data include priorities (or relative deadlines) for the concurrent units, and priority ceilings (preemption levels or deadly floors) for shared resources. Called CF_DATA. in Fig. 1, these data are the configuration information in the UMLforCode generation model.

A. Design and analysis in the software development process

From a software engineering perspective a summary of the methodological steps to follow may be stated as:

(a) Introduction of design intent in UML using HLAM. The definition of the models to use for this step is the aim of the research work proposed by this work-in-progress paper.

(b) An initial schedulability analysis architectural validation may be done using speculative values for execution times using M2M_A in exploration mode, the extraction of schedulability analyses models with M2T_MAST, and the execution of the analysis tools (in this case MAST).

(c) Generation of UMLforCode model with M2M_C.

(d) Code generation (M2T_ADA) and execution in profiling mode (to use less speculative values in the analysis). Alternatively the code may be statically analyzed and executed with ad-hoc worst case execution time analysis tools.

(e) Generation of the SAM model with M2M_A, which now includes the recovered execution times (WCET)

(f) Extraction of final schedulability analysis models with M2T_MAST, and execution of the analysis tools (i.e. MAST)

(g) Recovery of analysis results in SAM and transposition of configuration data into the UMLforCode model.

(h) Generation of the final application code.

This paper proposes a way for HLAM models to be created so that all the transformations mentioned may work correctly. The transformations for generating UMLForCode (M2M_C) and analysis (M2M_A) models will be our next steps.

B. Related work

Following previous efforts that have studied the design of real-time systems using object oriented formalisms, we observe that most of them include the specification of the concurrency using structural models, usually at the design-for-implementation level. These dual structural-behavioral formalisms are made in the aim that this will help to realize schedulability analysis with the simple tasking model in mind and basic rate monotonic analysis (RMA) techniques later on. Unfortunately the complexity of the mechanisms used to generate the code makes this assumption not realistic, such as in ROOM [9], Octopus/UML [10], ACCORD/UML [12] [13], Comet [14], or the design model extremely constrained and monolithic such as in HRT-HOOD [15], OO-HARTS [16].

Being a syncretism of all those mentioned, and in order to ease the application of simple schedulability analysis techniques, the high level application modeling constructs in MARTE (see the HLAM section in [3]) also facilitate the use of structural models for the specification of the concurrency. But the interactions between them (including distribution) may take complex patterns that require a richer model for the analysis. Then, from the analysis perspective the end-to-end offset based analysis techniques scale far better to deal with these scenarios than the basic RMA tasking model. HLAM proposes two basic building blocks, the real-time unit: *RtUnit* and the passive protected unit: *PpUnit*. As for the behaviors in them (the code inside the marks), due to its natural complexity it is usually not just passive linear code that can be modeled as a computation time; instead they include delays, and interactions among objects and nodes, mostly when they become formed out of a composition of distributed operations (behavioral models). In these cases a state machine (the basic construct used in most of the analyzed approaches) is not directly transformable into an analysis model.

From the analysis perspective, the models that are required to apply the modern offset-based analysis techniques, are fundamentally scenarios. A scenario is an expression of the (worst case) expected or observable manifestation of the design intents (coded behaviors). This is the basis for coping with complexity that distinguishes RMA schedulability analysis techniques from those other strategies like the based on timed automata or synchronous languages.

As a modeling language for this domain, the scheduling analysis modeling section of MARTE (SAM) is also able to express that kind of scenario models, and then it is an adequate formalism to feed the corresponding analysis tools. Unfortunately these scenarios are not necessarily part of the initial specification of the system behavior. They are a means to express: the expected stimuli, the high level expected workload, and the end-to-end timing requirements, but they are usually not the basic data used for design intent or code generation drawn by the designers.

The creation of these (usually worst case) analysis oriented scenarios in tune with the final code is actually the main duty and a high responsibility of the real-time practitioner. In order to help in this labor the automation tools need the model used for code generation to have the behaviors of its operations expressed as scenarios. For this reason the adequate input models for the generation of the code inside the operations in the UMLforCode model are UML activities. This is why the tool that fills the code for the methods of the classes retrieves it from activity diagrams.

The use of scenarios has an additional benefit. This method helps to support the design of applications in terms of composable parts, which are closer in granularity to the concept of real-time objects than to the fully component-based software engineering (CBSE) interpretation of components. In a fully component-based approach, the creation of the analysis models would have to be made as a combination of both, structural elements plus their deployment. In an object-oriented model-driven approach, this later strong form of composability is in a higher level of abstraction, but still may benefit of the approach here described in order to assess a variety of non-functional properties, in our case of course the assessment of its timing properties by means of schedulability analysis.

C. Contribution of the effort here described

The contribution of this work-in-progress paper is in the clarification of the approach, the steps to follow in a software engineering process, and the initial identification of rules and concrete modeling elements in UML and MARTE so that suitable design models may be processed by the tools that the full model-based methodology presented comprises.

III. HIGH-LEVEL MODELING RULES

The basis for modeling with schedulability analysis in mind is the specification of three basic models, the platform, the logic of the application and the workload the system is expected to support. An initial set of modeling rules, which included those for describing the platform, was proposed in [1]. Here we enhance and extend it to address also code generation. For those terms in italics refer to the MARTE specification [3].

A. Modeling independent tasks

In cases where tasks are independent, the basic rules for describing the logic of the application are:

1. Each *RtUnit* have only one *schedulableResource* (thread) on it. Its behaviors (operations) may not be called from other *RtUnits*, and run under the scheduling parameters associated to that schedulable resource. Behaviors called in other passive classes run under the scheduling parameters of the calling *RtUnit*.
2. Each *RtUnit* has one and only one of its operations (UML BehavioralFeatures or behaviors) with the stereotype *RtFeature*. This has an *RtSpecification* (a comment stereotyped) in which at least the attribute *occKind*, has to be specified. This attribute indicates the *ArrivalPattern* (the triggering scheme) of the underlying task (usually a periodic pattern).

3. All the *RtUnits* deployed in a *processingResource* (a host) are handled by the same scheduler and use the same (or fully compatible) scheduling policy.
4. Each *RtUnit* whose *isMain* attribute is set to true, implies the presence of an execution host where the main service of the *RtUnit* is deployed.
5. The attribute *srPoolPolicy* holds the value *infiniteWait*

B. Modeling share data interactions

When tasks share passive data the *PpUnit* modeling construct is used, in this case these additional rules apply:

6. The *ExecKind* of *PpUnit* services is *ImmediatRemote*
7. All services of the *PpUnit* use the same protection protocol: *ImmediateCeiling* or *PriorityInheritance*
8. The *ConcurrencyPolicy* of *PpUnit* is *Guarded*.

The *concurrencyPolicy* of the kind *Concurrent* might be enabled in order to have the writer/reader *ConcurrencyKind* available, but this behavior requires additional capabilities from the analysis techniques to take really advantage of it, so in principle it is discouraged.

C. Modeling end-to-end flows

When tasks interact by triggering one another, chains of actions need to be ensemble. In this case the calling of the first action (task) in the chain determines the execution periodicity and end-to-end deadline. The following rules apply in this case:

9. The first restriction in rule 1 is here relaxed so that operations stereotyped as *RtServices* of *RtUnits* may be invoked by others using the SignalEvent semantics.
10. In this case, in order to have analyzable models, only the first calling operation in the chain may have an *ArrivalPattern* specified be means of its corresponding *RtFeature* and its *RtSpecification* comment.
11. Operations in an *RtUnit* that are not stereotyped as *RtServices* run in the context of the calling task. They are called passive and use the CallEvent semantics.

See [1] for additional rules that apply in general in specific phases of the development process.

The invocation of behaviors is made in activity diagrams. *sendSignalActions* are used for triggering *RtServices* and *callActions* for calling passive operations. The invocation of an *RtService* that holds an arrival pattern implies the initialization of the task (usually invoked in the main). This auxiliary code will be automatically inserted in the activity diagrams of the UMLforCode generation model. This will be done following the arrival pattern of the task (usually periodic).

The constraining rules described here for this HLAM input model are meant for ensuring (i) analyzability by means of schedulability analysis (ii) consistency between the analysis models and the generated code, (iii) the minimum usage of tools and transformations, and (iv) compliance with executable versions of UML, fUML [17] and the future standard for a Precise Semantics of UML Composite Structures [18].

IV. Conclusions and Future Work

This paper presents a model-based software engineering methodology for the development of real-time applications. Some steps in the necessary chain of tools have been realized and this paper shows some of the basic steps missing. It addresses the simple independent tasking model, the passive protected data sharing, and the distributed end-to-end flows of linear execution. The models compliant to the rules here defined are suitable to be transformed into both: schedulability analysis and code generation intermediate models. Next steps include, the high level transformations into these intermediate models, experiments, rules to handle interrupts, and tooling support for the complete iterative engineering process.

References

[1] J. Medina and A. Garcia Cuesta. Model-Based Analysis and Design of Real-Time Distributed Systems with Ada and the UML Profile for MARTE. In Proc. of the 16th International Conf. on Reliable Software Technologies-AdaEurope 2011, LNCS 6652, pp 89-102.

[2] Object Management Group. Unified Modeling Language version 2.4.1, OMG document formal/2011-08-06, 2011.

[3] Object Management Group, UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, version 1.1, OMG doc. formal/2011-06-02, 2011.

[4] Silvia Mazzini, Stefano Puri, and Tullio Vardanega. An MDE Methodology for the Development of High-Integrity Real-Time Systems. Proceedings of Design, Automation and Test in Europe. Nice, France. April 20-24, 2009.

[5] Object Management Group. Action Language for Foundational UML (Alf), Concrete Syntax for a UML Action Language. OMG document ptc/2010-10-05, 2010.

[6] http://mast.unican.es/umlmast/marte2mast

[7] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake, MAST: Modeling and Analysis Suite for Real-Time Applications, in Proc. of the Euromicro Conference on Real-Time Systems, June 2001.

[8] http://mast.unican.es/umlmast/uml2ada/

[9] Bran Selic and Jim Rumbaugh. Using UML for Modeling Complex Real-Time Systems. Rational white papers, http://www.rational.com/products/whitepapers/UML-rt.pdf, March 1998

[10] Domiczi, R. Farfarakis and J. Ziegler. Octopus Supplement Volume 1. Nokia Research Center. http://www-nrc.nokia.com/octopus/supplement/index.html, 1999.

[11] Laila Kabous. An Object Oriented Design Methodology for Hard Real Time Systems: The OOHARTS Approach. Doctoral Theses, School Carl von Ossietzky, Universität Oldenburg. 2002

[12] F. Terrier, G. Fouquier, D. Bras, L. Rioux, P. Vanuxeem and A. Lanusse. A Real Time Object Model. Presented in TOOLS Europe'96. Paris, France. Prentice Hall, 1996

[13] A. Lanusse, S. Gerard and F. Terrier. Real-Time Modeling with UML: The ACCORD Approach. In Selected papers from the 1st. Int. Workshop on The Unified Modeling Language UML'98: Beyond the Notation. Mulhouse, France, June 3-4, 1998. Pp. 319-335. ISBN:3-540-66252-9.

[14] Hassan Gomaa. Designing Concurrent, Distributed and Real-Time Aplications with UML. ISBN 0-201-65793-7, Addison-Wesley, 2000.

[15] Alan Burns, Andy Wellings. HRT-HOOD, a structured design method for hard real-time ADA systems. ISBN 0 444 82164 3. Elsevier, 1995

[16] Mazzini S., D'Alessandro M., Di Natale M., Domenici A., Lipari G. and Vardanega T. HRT-UML: taking HRT-HOOD into UML. In Proc. of 8th Conference on Reliable Software Technologies Ada Europe, 2003

[17] Object Management Group. Semantics of a Foundational Subset for ExecutableUML Models (fUML), v1.1, OMG Document: ptc/2012-10-18. http://www.omg.org/spec/FUML/1.1

[18] Object Management Group. Precise Semantics of UML Composite Structures, Request For Proposals. OMG Document: ad/11-12-07

An arbitrarily precise time synchronization algorithm based on Ethernet Switch serialization

Fabrice Frances, Ahlem Mifdaoui,
ISAE/DMIA
University of Toulouse
Toulouse, France
{*fabrice.frances* | *ahlem.mifdaoui*}*@isae.fr*

Xavier Codogni, Christian Fraboul
ENSEEIHT/IRIT
University of Toulouse
Toulouse, France
{*xavier.codogni* | *christian.fraboul*}*@enseeiht.fr*

***Abstract*—It turns out that trying to play a worst-case traversal time (WCTT) scenario on a real experimentation platform is a Real-Time problem with extremely tight constraints. When two packets (with the same destination) arrive to two different input ports of a network switch within a time frame of only a few nanoseconds, the order of these packets in the output port queue will reflect this small nanoseconds arrival difference. Moreover, failing to emit packets within this tiny time frame will exhibit a different scenario than expected, potentially so radically different in farther places of the network that the behavior of the whole system seems affected by a butterfly effect. As we were trying to achieve the most precise clock synchronization we could with standard hardware, we have had the idea to turn this butterfly effect to our benefit and develop an arbitrarily precise time synchronization algorithm that only requires a standard Ethernet switch connecting the two hosts to synchronize and a third host on the network that will serve as a synchronization helper.**

***Keywords*-clock synchronization; Ethernet switch;**

I. Introduction

When trying to exhibit Worst-Case Traversal Time (WCTT) scenarios on a *real* network experimentation platform, we were faced with the problem of playing scenarios containing synchronized emission of frames in several End-Systems that do not share a common clock: most often, these worst case scenarios consist in simultaneous arrivals of frames in a switching network element, for example Ethernet frames on an Avionics Full-Duplex Ethernet Switch (AFDX).

Let's consider the simplest example of two frames (A and B) emitted by two different End-Systems, when these frames arrive to their connecting Ethernet switch and have the same destination. We assume t_A and t_B are the arrival times of the last bit of these frames in an ideal FCFS switch running in Store and Forward mode. Then we can observe that when $t_B < t_A < t_B + Size_B/C$ (where C is the capacity of the ouput port), then frame B will be retransmitted first starting from time t_B (plus a small technological latency considered as 0 for this demonstration), whereas frame A will have to wait until $t_B + Size_B/C$ before starting to be retransmitted. From a scheduling point of view, we can see that the same scenario of trafic would be observed on the output port of the switch, whatever the actual value of t_A is in interval $]t_B, t_B+Size_B/C]$, i.e. retransmission of frame B starting at t_B, and retransmission of frame A starting at time $t_B + Size_B/C$ (thus frame A will have to wait for the end of retransmission of a part of frame B). However from a latency point of view, the worst latency we can observe is when t_A is *nearly* equal to B (in this case we get the expected result where frame A has to wait for the full retransmission of B). So we can see that if we want to play the worst-case latency scenario, we have to be *extremely* precise and have t_A very close to t_B (but not $t_A < t_B$ otherwise frame A would arrive first and be served without latency).

From this on, our quest was thus to find a way to synchronize our network End-Systems as much as we could. As we didn't want to invest in specialized hardware like GPS clocks and instead propose a solution easily reproducible by any researcher, we tested the existing clock synchronization algorithms. First, Network Time Protocol [1]: the ntpd daemon is present in every Unix distribution. However, NTP is targeted for 1 ms synchronization at best, clearly not enough for our requirements.

So, we then tested the Distributed Clocks of the Precision Time Protocol [2][3], but again we couldn't reach a tight enough synchronization: as the 1588 Working Group claims, microsecond precision shall be attainable with PTP, but only with dedicated hardware. There exists IEEE1588 Ethernet interfaces with integrated PTP, but this is not the case of most standard Ethernet interfaces found in PCs, where PTP has to be run in software. So, the software implementations we tested were not able to give a better than 10 μs synchronization. And studying the algorithms used in PTP revealed that the slave synchronization always rely on message exchanges sent on an Ethernet network, without taking into account the random latency effect of an eventual traffic arriving simultaneously on a switch.

Finally, we noticed that even the Robust Absolute and Difference Clock (RADclock) [4] has not taken benefit of the synchronization hardware that is available for free in every Ethernet network: the Switch.

II. The foundation idea: Ethernet Switch serialization

Whether the switch is in Store and Forward or in Cut-Through mode, frames relayed to the same output port have to be serialized (two frames cannot be emitted at

the same time). When two frames arrive in a perfectly simultaneous timing on two input ports, this serialization process is usually a side-effect of the behavior of a centralized entity that takes the frames arrived in the input ports and relay them to the appropriate output queue. Were the relaying process be distributed among several entities (one per input port), then frames would also be serialized into the output port queue with a many-writer/single-reader scheme.

Since we were trying to have several End-Systems emit simultaneous frames and we had observed that a very small variation on the emission time drastically affects the order of serialized frames after the first switch, why not interpret this sequencing order as a proof that one frame arrived later? A general setup can now be given: all what is needed to synchronize two End-Systems is to have them connected to two input ports of an Ethernet switch with a FCFS policy (this is the most common policy in small and medium-sized switches), and an observer on a third output port. The two End-Systems just need to try to send a frame to the observer at the same time, and the observer will tell them which one was first or second. This is the only thing the observer is able to say, it is binary information: this End-System's frame arrived first or second. For precise synchronization, it is useless to extract quantitative timing information: if the two End-Systems are nearly synchronized, the two frames will be serialized and arrive in a burst, one after the other, with no extra delay other than the standard InterFrame Gap (IFG) of Ethernet.

Now, with such a binary information (the order of frames is "A, then B" or "B, then A"), we will adjust one of the End-Systems' clocks by an increment of time, and repeat the process, iteratively dividing the increment of time by a modified dichotomy. This is the point of the *arbitrarily precise* expression which sounds like an hyperbole: of course an infinite precision would only make sense with an ideal switch and ideal End-Systems able to adjust frame emission times with an infinite precision. But still, the algorithm proposed here has no limitation by itself: the precision will only be limited by the actual End-Systems and switch used, and precision can be *arbitrarily* improved by using faster End-Systems and/or switches when they become available.

III. Informal description of the protocol

The protocol is composed of only two types of messages: messages sent by the clients to the synchronization helper host (the observer mentioned above will take the role of a synchronization server in this protocol), and messages sent by the synchronization server to the clients. The synchronization clients will never exchange messages together; communications take place only between a client and the synchronization server (helper).

Messages sent by the clients contain a timestamp value, denoted t_n^{client}, and n which is a sequence number starting from 0 (hence n identifies the messages sent by the clients). Messages sent by the synchronization server to a client host contain a tuple of four values (Δ_n, δ_n, t_{n+1}^{server}, $n+1$), Δ_n being interpreted as a request for clock adjustment, δ_n a notification of the current precision, and the last two values forming a request to emit a client message numbered $n+1$ at time t_{n+1}^{server}.

A. Initialization steps for a very rough synchronization

These first steps are required to initialize communication exchanges between the hosts to synchronize together and the synchronization helper. They are also used to reduce the large clock difference that might exist between the two client hosts at startup.

1) Step 1: Start server process on observer host C, it waits for two messages coming from client hosts A and B.

2) Step 2: Start client processes on hosts A and B; they send a client message to server C. The message contains the local time on the client host, t_0^{client}. This is just to let the synchronization server have a rough idea of what time it currently is on the client hosts.

3) Step 3: On receipt of each of these first client datagrams, the synchronization server calculates an approximated clock difference between the client and the server: $\Delta^{client} = t_{receipt}^{server} - t_0^{client}$

Please note that t_0^{client} *is not* the accurate time of packet emission on the client. It is a timestamp written by the client in the message sent to the server. This timestamp is obtained by reading the current local clock prior to building and sending the datagram.

Conversely, $t_{receipt}^{server}$ *is not* the accurate time of packet receipt on the server. It is the current time read on the server after the message read call returns. The thread that executes this blocking datagram read is resumed after a non-predictable amount of time due to slice execution of the currently active thread, followed by slice execution of other more prioritized threads.

However, the approximation on Δ^{client} is anticipated to be lower than one second on non-overloaded hosts.

4) Step 4: The synchronization server C now plans a roughly synchronized emission (from both clients) to occur at time $t_0^{server} = t_{current}^{server} + c$

c is a constant delay bigger than the error in the approximated Δ^{client}, e.g. two seconds.

Thus, synchronization server C sends a request to each client, asking for a clock adjustment of $\Delta_0 = -\Delta^{client}$ so that time on both clients becomes *roughly* equal to time on server C. In the same request, it also asks for a message emission at time t_1^{server}. When the client has adjusted its clock as requested, this t_1^{server} time is interpreted as a local client time, i.e. the client considers that $t_1^{client} \approx t_1^{server}$.

B. The arbitrarily precise synchronization scheme

At the beginning of next step, a client has already received a request to send a datagram to the server at time t_n^{server}, with n being an iteration number (n is 1 when only the initialization steps above have been executed).

The following step is repeated until the desired synchronization precision is reached. The currently attained

precision is denoted δ_n and has been transmitted in the last server message alongside with the request for clock adjustment and the requested time for the next client message emission. Thus the value of δ_0 has been sent to the clients in the last step of the initialization phase. Chosing a good value for δ_0 will have an impact on the number of steps needed to attain a defined precision. However, it is useless to select a very small value like 1 ms, because in this case there is a possibility that the altered dichotomy has to do hundreds of 1 ms increments in the same direction. In the other hand, starting with a large value (like 1 second) only requires 20 steps to reach the 1 μs precision.

1) Repeated step: Both clients *actively* wait for time t_n^{client} to happen (i.e. with an active loop) and then immediately send the requested datagram to the server. The two datagrams are received as Ethernet frames in the switch that connects the clients, and since the destination of the two datagrams is the same, the two Ethernet frames are serialized for retransmission on the output port that leads to the destination. The serialization process will be further detailed in next section.

The synchronization server will thus receive the two messages in one of the two possible orders (either the message from A followed by the message from B, or the message from B followed by the message from A). The received order tells which client host has its clock in advance compared to the other client host. So the synchronization server prepares a new request for clock adjustment: the client host whose message arrived first will be requested a clock adjustment of δ^{n+1}, while the other will be requested a clock adjustment of 0.

In the same message, the synchronization server also ask both clients to plan their next message emission at time $t_{n+1}^{server} = t_{current}^{server} + c$.

The last parameter of the message is the precision delta that will be associated to the next iterated step. This δ^{n+1} is calculated with the following rule: if the order of reception is the same as the one observed during the previous *nearly-synchronized* emission, then δ stays the same (i.e. $\delta^{n+1} = \delta^n$) ; but if the order of reception is reversed, then δ is divided by two. The rationale for this altered dichotomy is discussed in next section.

IV. Rationale for the protocol efficiency

The rationale behind the foundation idea is that the latencies between each of the synchronizing client hosts and the switch's relaying entity are equal for both clients. More precisely, the delay of interest comes from the following sequence of events:

- a read of the client's internal clock that determines the end of the active wait loop,
- the write of the client synchronization message, which is a system call that provides the protocol datagram to the UDP/IP send stack, including the final Ethernet driver, which in turn provides the Ethernet frame to the hardware Ethernet card (or interface),

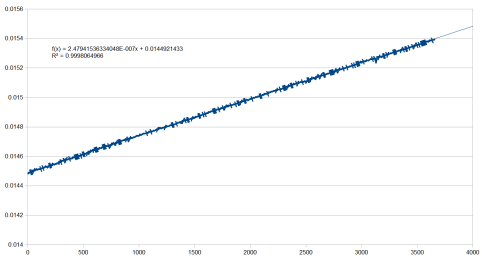


Figure 1. Measurement of clock difference (in seconds) between two non-synchronized hosts, over 3600 seconds (1 hour)

- the emission of the Ethernet frame,
- the propagation on the link that connects to the switch, delaying the reception of the emitted signal on the switch's input port,
- the switch algorithm that senses the input ports for incoming frames and decides at which point a frame can be relayed to an output port's queue (e.g. as soon as the destination address has been received, in Cut-Through mode).

The synchronization precision that our protocol will be able to reach is directly affected by variations in any of these points, so it is worthwhile explaining how jitter will be controlled. Also, it must be noted that the sequence of timely-controlled events has been reduced to a minimum: in other real-time distributed algorithms, large latencies with uncontrolled jitter exist in the network receiver stack and in the delivery of a received message to an application thread. Our solution fully removes these two sources of latency.

A. Controlling latencies from the client host to the switch

First, we assume the two clients have the same hardware/software combination: the same protocol client program is run on the same hardware and operating system. We will thus assume that the execution delay, between the read of the client's internal clock and the I/O command sent to the hardware interface by the Ethernet driver, is *constant* and identical on both clients. This assumption does not seem unrealistic, even if execution of other processes on a synchronizing client will introduce variations: we will try to reduce these interactions by implementing an active wait loop around the internal clock read, and no system call between this read and the datagram write, in order to reduce eventual thread switches. Of course, we will guard against clock skew. Our measures have shown very good consistency in stable conditions of temperature, as can be seen on Figure 1, showing a constant clock drift.

Also, the execution time cannot be guaranteed to be exactly the same though, because of possible different content in the memory caches. However, the tight active wait loop will also help in keeping these memory caches filled with the desired content. Still, random hardware interrupts (disks or other hardware sources) might happen

in the synchronizing clients: we will assume that these random events will be *quite infrequent* and we will protect against these events with the *altered dichotomy* algorithm.

Secondly, the length of the cables that connect the client hosts to the switch will also be assumed the same, even if this parameter has a smaller impact on the overall latency. Finally, the TxC (Transmit Clock) of the Ethernet interfaces will be considered equal: any clock skew between these TxC will be compensated by the receiving switch.

We could also argument that a significant difference in the delay that separates the read of the client's internal clock and the arrival of the frame in the corresponding switch's input port is acceptable for our Real-Time application (playing a worst case traffic scenario): the same difference will exist when synchronously emitting the scenario, so what is really important for us is how we can have a fine control over the arrival of frames in the switch.

In conclusion, the only remaining source of uncontrolled latency is the one present in the switch, before detecting an arrival of a frame and handling that frame (mainly relaying it to an output port). It is expected that some switches will scan their input ports in a loop, giving potential order inversion when frames arrive in a small time window, but we prefer to consider this switch behavior as a black box so that the protocol remains generic.

B. The altered dichotomy algorithm

The altered dichotomy scheme has been designed to account for transient variations, e.g. additional latency in the client execution, due to some random event (disk interrupt for example). Also, even if no network application is executed at the same time as the synchronization protocol, there are always a few packets sent by daemons from time to time. In the emission protocol stack on the client host, such packets could delay a client synchronization message, affecting the order of arrival of the synchronized messages on the switch (and thus on the synchronization server). In such a case, the clock adjustment might take one erroneous direction which will hopefully be compensated by two half moves in the other direction (a normal dichotomy would never compensate a wrong move). This is exemplified by Figure 2, the upper part shows the case where the clock adjustment is wrongly halved, and the lower part shows the case where the clock adjustment should have been halved. The previous adjustment is depicted in order to show the last direction and amplitude of adjustment. In both cases, the next two adjustments will compensate the erroneous one.

V. Conclusion

This is a Work In Progress, the protocol is still in development as we are still working on the best way to trigger the emission of frames on the End-Systems so as to make the most of current hardware/operating system. But a number of ideas make this work promising, not only for our needs of synchronizing a network platform that aims to play worst case scenarios of traffic in Real Time, but also for any other distributed real-time application as soon as the nodes are connected with standard Ethernet technology:

- the use of the natural serialization that takes place in standard Ethernet switches, in order to provide binary information on which client host is late or in advance,
- the reduction of the number and scope of uncontrolled sources of latencies: emission latency is controlled and message delivery latency in the destination observer (synchronization helper) is not an issue. Moreover, the observer does not need to be connected to the same switch as the synchronizing clients: once client messages have been serialized by the first switch, they can cross any number of cascading switches before reaching the observer. Conversely, the observer could be integrated in the switch.
- the robustness to transient errors with an original altered dichotomy that brings further confidence in the capability of our algorithm to give the most precise synchronization, with standard operating systems and no extra hardware.

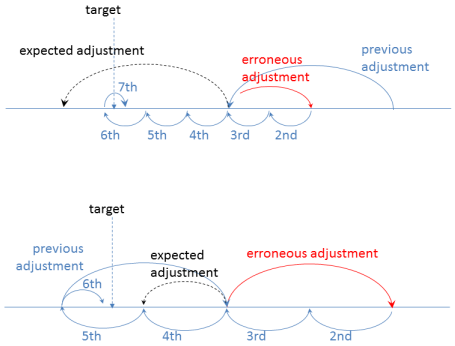


Figure 2. Robustness of the altered dichotomy algorithm in case of a wrong decision

References

[1] David Mills, Jim Martin, Jack Burbank, William Kasch, *Network Time Protocol Version 4: Protocol and Algorithms Specification*, RFC 5905, ISSN: 2070-1721

[2] *Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, PTP, IEEE 1588-2002 standard

[3] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, PTP v2, IEEE 1588-2008 standard

[4] Julien Ridoux and Darryl Veitch, *Ten Microseconds Over LAN, for Free (Extended)*, IEEE Transactions on Instrumentation and Measurement (TIM) vol. 58(6), pp. 1841-1848, June 2009

End-to-end Timing Challenges in Seamless Tool Chain Development for Vehicular Embedded Real-Time Systems

Saad Mubeen*†, Jukka Mäki-Turja*† and Mikael Sjödin*
* *Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Västerås, Sweden*
† *Arcticus Systems AB, Järfälla, Sweden*
saad.mubeen@mdh.se

***Abstract*—Often, there exists mismatch among tools that are used for structural, functional, and execution modeling of vehicular embedded real-time systems in the industry. Building a seamless tool chain to support model- and component-based development of these systems with different, and sometimes independent, tools is challenging. Within this context, we investigate the challenges related to modeling, analyzing, and exchanging end-to-end timing information. We target domain specific models like EAST-ADL supplemented by the Timing Augmented Description Language; and component and execution models that are already used in the industry such as the Rubus Component Model.**

***Keywords*-Automotive embedded real-time systems; timing model; component-based real-time systems; model- and component-based development; timing analysis.**

I. Introduction

The industrial requirements on embedded real-time systems are constantly evolving. With the flexibility offered by software, the complexity of system designs and the amount of advanced computer controlled functionality in products is increasing. Historically, developers of embedded real-time systems have used low level programming languages to guarantee full control of the system behavior. Hence, many embedded real-time systems have become overly complex and hard to manage during functionality or technology shifts. The variety of functionality in today's embedded real-time systems requires development methods and tools that support flexible and efficient development.

Within the business segment of construction-equipment vehicles (and similar segments for heavy special-purpose vehicles), model-based development of software architectures for embedded real-time systems has had a surge the last few years. The idea is to use models to describe functions, structures and other design artifacts. This is in contrast to the previously used text documents. Benefits that are sought by this transition in design technology include simplified communication amongst engineers and other stake holders, use of precise and unambiguous notations to describe complex features, faster turn-around times in early design phases, possibilities to automatically perform timing analysis and derive test cases, and possibilities to automatically generate code.

In practice, existing tools and languages for model-based software design for embedded real-time systems imposes many hinders with respect to information flow between different abstraction levels and project phases. In industry, productivity is hampered by incompatible tools and file formats, in conjunction with the need for non-trivial, manual and tedious translations between different model formats. Moreover, these translations are done in ad hoc fashion making the result of the translation unpredictable and potentially altered in terms of semantics. Thus, there is a strong need to investigate how to work with existing modeling languages and tools in an effective and efficient way. A solution must entail possibilities to make tools inter-operable to allow automated (and semi-automated) translations between modeling languages and tools with preserved model semantics.

A. Motivation and contribution

The motivation for this work comes from the industrial needs at the partner companies. Different tools are used for *structural*, *functional*, and *execution* modeling of vehicular embedded real-time systems. The translations between different models of the systems are done manually and in ad hoc fashion. Hence, there is a need to develop a seamless tool chain that should support *structural*, *functional*, and *execution* modeling of vehicular embedded real-time systems.

In this paper we identify and discuss the challenges related to modeling, analyzing, and exchanging the end-to-end timing information during the development of a seamless tool chain for these systems. We focus on the models and related tools that support model- and component-based development of these systems in the segment of construction-equipment vehicles such as EAST-ADL [1], Timing Augmented Description Language (TADL2) [2], Rubus Component Model [3] and Rubus-ICE [4].

B. Outline

The rest of the paper is organized as follows. In Section II, we discuss the background and related work. In Section III, we discuss the research challenges. Section IV discusses the current work.

II. Background and related work

A. Structural and functional modeling of vehicular real-time systems

The *structural modeling* is concerned with the structure definition of requirements and high-level architectural objects. Whereas, the *functional modeling* refers to the structured way of representing software functions for the system to be modeled. In this work, we consider the structural and functional modeling support of EAST-ADL

which supports domain-specific modeling concepts for modeling product lines of automotive control systems. Basically, it is an architecture description language that tends to describe, capture and model the engineering information of the automotive electronic systems in a standardized way. It describes the functionality of the vehicle at four vertical levels of abstraction starting from requirements capturing to the system implementation as shown in Figure 1.

B. Execution modeling of vehicular real-time systems

The *execution modeling* [5] is concerned with the modeling of run-time properties and/or requirements (e.g., end-to-end deadlines and jitter) of software functions. For example, in resource-constrained and safety-critical embedded systems, it is simply not enough to have a high-level view of the system in the models. Instead, the models need to capture what goes on at the execution level. The modeling of these systems should extend down to the execution level to allow precise control of resource utilization, avoid violation of timing requirements when the system is executed, and certify systems toward safety standards [6]. In this work, we focus on the component-based software development technologies that provide execution modeling support in vehicular domain and are actually used in the industry such as the Rubus Component Model.

C. Abstraction levels during the development of vehicular real-time systems

In this work, we consider four abstraction levels that are described by EAST-ADL. These levels are shown in the Figure 1.

1) Vehicle or end-to-end level: At the vehicle level, requirements, functionality and features of the vehicle are captured in an informal (often textual) and solution-independent way. Basically, this level captures the information regarding what the system should do [7]. In the segment of construction-equipment vehicles, this abstraction level is better known as end-to-end level because features and requirements on the end-to-end functionality of the machine or vehicle are captured in an informal way.

2) Analysis level: At the analysis level, the requirements are captured in a formal way. Functionality of the system is defined based on requirements and features without implementation details. A high-level analysis may also be performed for functional verification.

3) Design level: The artifact developed at the analysis level is refined into design functions at the design level. The resulting artifact at this level also contains middleware abstraction and hardware architecture. In addition, software functions to hardware allocation may be present.

4) Implementation level: At the implementation level, the design-level artifact is refined to software-based implementation of the system functionality. The EAST-ADL methodology defines the system at this level in terms of AUTOSAR elements. However, in this work, our focus is on using the Rubus Component Model and its development environment Rubus-ICE at the implementation level. Hence, the artifact at this level consists of the software architecture of the system defined in terms of Rubus components and their interactions. We choose Rubus instead of AUTOSAR at the implementation level because of industrial needs at the partner companies.

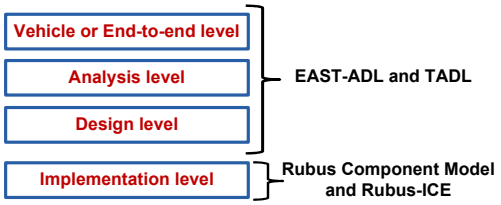


Figure 1. Abstraction levels considered during the development

D. The Rubus concept

Rubus is a collection of methods and tools for model- and component-based development of dependable embedded real-time systems. Rubus is developed by Arcticus Systems [4] in close collaboration with several academic and industrial partners. Rubus is today mainly used for the development of control functionality in vehicles by several international companies [8], [9], [10], [11]. The Rubus concept is based around the Rubus Component Model (RCM) and its development environment Rubus-ICE which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable, timing analyzable and synthesizable control functions in resource-constrained embedded systems. The timing analysis supported by Rubus-ICE includes distributed end-to-end response-time and delay analysis [12], [13].

E. AUTOSAR, TIMMO, and TADL

AUTOSAR (AUTomotive Open System ARchitecture) [14] is an industrial initiative to provide standardized software architecture for the development of software in the automotive domain. It can be viewed as a standardized distributed component model [15]. In AUTOSAR, the application software is defined in terms of Software Components (SWCs). The virtual function bus handles the distribution of SWCs, their virtual integration and communication at design time. Furthermore, it hides the low-level implementation and communication details at the design time. AUTOSAR provides same interfaces and services to the connected SWCs irrespective of the type of communication (intra- or inter-ECU).

TIMing MOdel (TIMMO) [16] is a large EU research project and serves as an initiative to provide AUTOSAR with a timing model. It describes a predictable methodology and a language Timing Augmented Description Language (TADL) [17] to express timing requirements and timing constraints in all design phases during the development of automotive embedded systems. TIMMO-2-USE [2] is the follow-up project to TIMMO. It defines TADL2 language that includes a major redefinition of TADL.

F. AUTOSAR vs RCM

When AUTOSAR was being developed, there was no focus placed on its ability to specify and handle timing-related information such as real-time requirements and properties. On the other hand, such requirements and capabilities were taken into account right from the beginning during the development of RCM. AUTOSAR describes embedded software development at a relatively higher level of abstraction compared to RCM. The software component in RCM more resembles to the runnable entity compared to AUTOSAR SWC. The runnable entity is schedulable part of AUTOSAR SWC.

As compared to AUTOSAR, RCM clearly distinguishes between the control flow and the data flow among the software components in a node or Electronic Control Unit (ECU). AUTOSAR hides the modeling of the execution environment. On the other hand, RCM explicitly allows the modeling of execution requirements, e.g., jitter and deadlines, at an abstraction level close to the functional specification while abstracting the implementation details. The Sender Receiver communication mechanism in AUTOSAR is very similar to the pipe-and-filter communication mechanism for component interconnection in RCM.

In conclusion, AUTOSAR is more focussed on the functional and structural abstractions, hiding the implementation details about execution and communication. Whereas, RCM is all about modeling, analysis and synthesis of the execution environment of software functions. Basically, AUTOSAR hides the details that RCM highlights.

III. Research Challenges

There are several different types of challenges that are faced during the development of a seamless tool chain to support model- and component-based development of embedded real-time systems in the segment of construction-equipment vehicles. In this section, we identify and discuss only those challenges that are concerned with the modeling, analyzing, and exchanging of the end-to-end timing information.

A. Mismatch between design and implementation levels

When RCM is used instead of AUTOSAR at the implementation level, there exists an incompatibility between the design and implementation levels. We believe, the main reason behind this incompatibility is the concept of virtual function bus in AUTOSAR. At the implementation level, EAST-ADL relies on virtual function bus for the distribution of software components, their virtual integration and communication. Further, virtual function bus hides the low-level implementation and communication details. At the design time, the components are considered at the same level irrespective of the communication they need, i.e., intra- or inter-ECU.

In RCM, there is no concept of virtual function bus. It differentiates between intra- and inter-ECU communication among its software components. It uses network interface components for inter-ECU communication; otherwise, the components communicate with each other via data and trigger ports. Hence, the communications should be explicitly modeled when RCM is used at the implementation level. Moreover, the timing related information on the communications should be explicitly specified in order to perform the end-to-end timing analysis at the implementation level [18].

The problem is that the design-level model does not differentiate between intra- and inter-ECU communications, whereas these communications are explicitly modeled at the implementation level when RCM is used. The timing-related information on communications is also explicitly available at the implementation level when RCM is used. One of the main challenges is to make the design and implementation levels compatible with respect to communications and the end-to-end timing information.

B. Refinement and translation of timing requirements and constraints

The timing requirements and constraints on vehicle features that are captured at the top level may be refined and broken down into more than one requirement and constraint at the lower levels. For example, a timing constraint specified on the braking system feature of the vehicle requires the brakes to be applied within three milliseconds from the time when the brake paddle is pressed. This timing constraint may be refined into more than one constraint at the lower levels. At the implementation level, these (sub) constraints may be specified on several event chains that may be distributed over several ECUs that may be connected to one or more networks. The timing requirements and constraints should be unambiguously refined and translated along all abstraction levels without any loss of timing information.

EAST-ADL supplemented by TADL2 supports the refinement and translation of timing requirements and constraints along all abstraction levels. However, the EAST-ADL methodology assumes that the implementation level is handled by AUTOSAR. When AUTOSAR is replaced by RCM at the implementation level, the refinement and translation of timing information between the design and implementation levels does not hold. Within this context, the challenge is to unambiguously refine and translate the timing requirements and constraints between the design and implementation levels with preserved semantics.

C. Tracing of timing requirements and their verification

Another challenge that we have identified is the need to support traceability of timing requirements from the implementation-level entities to the vehicle-level entities, i.e., following the bottom-up approach. The tracing of timing requirements is important to perform full coverage analysis of the requirements and their verification. Often, a timing requirement at vehicle level may be broken down into several timing requirements at the implementation level. If implementation-level timing requirements are satisfied, the corresponding timing requirement at the vehicle level is considered verified. Hence, the traceability of timing requirements among all abstraction levels should be supported by the tool chain.

The traceability of timing requirements is supported by EAST-ADL from the implementation-level entities to the vehicle level entities. The support for traceability does not hold when AUTOSAR is replaced by RCM at the implementation-level. When RCM and Rubus-ICE are used at the implementation level, the tracing of the timing requirements from Rubus components to the design-level entities arises as another challenge.

The support for traceability of timing requirements is also important for change management. For example, the user of the tool chain may be interested in finding out how do changes in timing requirements, constraints, or budgets at the higher abstraction levels impact on the entities at the lower abstraction levels. This type of support in the tool chain may also be useful to perform design-space exploration during the development of the systems.

D. Raising the end-to-end timing analysis at higher abstraction level

The safety-critical nature of many vehicular embedded real-time systems require evidence that each action by the systems is taken in timely manner. For this purpose, the end-to-end response-time and delay analysis [19], [13] should be supported by the tool chain. In order to perform the timing analysis, the end-to-end timing model[1] should be extracted from the architecture of the system under development. The Rubus-ICE tool suite supports the end-to-end response-time and delay analysis.

When RCM is used at the implementation level, another challenge is to raise the end-to-end timing analysis support provided by Rubus-ICE to the design level (i.e., lifting the analysis one level above). For this purpose, the end-to-end timing model should also be provided at the design level. The analysis framework of Rubus-ICE supports the extraction of end-to-end timing models at the implementation level. However, raising these timing models one level above at the design level is another challenge that we have identified.

IV. Current Work

Currently, we are conducting questionnaire and interviews at the partner companies to identify the patterns, styles of expression, and subsets of the full expressiveness of EAST-ADL that are used by the designers during the development of embedded real-time systems in the segment of construction-equipment vehicles. During the identification of these patterns, styles and subsets, we consider only first three abstraction levels which are vehicle, analysis, and design. After their identification, they will be integrated with the Rubus-ICE at the implementation level for the development of seamless tool chain. During the integration and implementation, we will attack the timing related challenges that we discussed above.

Currently, we are also identifying the most suitable use case at the partner companies for the verification and validation of the tool chain. We specified several requirements on the selection of the use case. That is, it should be a distributed real-time system and it should employ, at least, one CAN bus for communication among ECUs. Whereas, each ECU should have at least one mode and three software components (i.e., two components for network input and output interfaces and at least one component implementing the functionality).

[1] [18] should be referred for the details about end-to-end timing model

Acknowledgement

This work is supported by the Swedish Knowledge Foundation (KKS) within the projects FEMMVA and SythSoft. The authors would like to thank the industrial partners Arcticus Systems and Volvo Construction Equipment (VCE), Sweden.

References

[1] "EAST-ADL Domain Model Specification, Deliverable D4.1.1," http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[2] "TIMMO-2-USE," http://www.timmo-2-use.org/.

[3] K. Hänninen et.al., "The Rubus Component Model for Resource Constrained Real-Time Systems," in *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[4] "Arcticus Systems," http://www.arcticus-systems.com.

[5] J. Mäki-Turja, K. Hänninen, and M. Nolin, "Towards efficient development of embedded real-time systems, the component based approach," in *International Conference on Embedded Systems & Applications (ESA), 2006*, June 2006.

[6] "ISO 26262-1:2011: Road vehicles Functional safety. http://www.iso.org/."

[7] "Hans Blom et. al. EAST-ADL- An Architecture Description Language for Automotive Software-Intensive Systems. White paper, Version M2.1.10, 2012, http://www.maenad.eu."

[8] "BAE Systems Hägglunds," http://www.baesystems.com/hagglunds.

[9] "Volvo Construction Equipment," http://www.volvoce.com.

[10] "Mecel," web page, http://www.mecel.se.

[11] "Knorr-bremse," web page, http://www.knorr-bremse.com.

[12] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for Holistic Response-time Analysis in an Industrial Tool Suite: Implementation Issues, Experiences and a Case Study," in *19th IEEE Conference on Engineering of Computer Based Systems (ECBS)*, April 2012, pp. 210 –221.

[13] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Computer Science and Information Systems, ISSN: 1361-1384*, vol. 10, no. 1, 2013.

[14] "AUTOSAR Techincal Overview, Version 2.2.2. AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1, The AUTOSAR Consortium, Aug., 2008," http://autosar.org.

[15] H. Heinecke et al., "AUTOSAR – Current results and preparations for exploitation," in *Proceedings of the 7th Euroforum Conference*, ser. EUROFORUM '06, May 2006.

[16] "TIMMO Methodology , Version 2," *TIMMO (TIMing MOdel), Deliverable 7*, October 2009, The TIMMO Consortium.

[17] "TADL: Timing Augmented Description Language, Version 2," *TIMMO (TIMing MOdel), Deliverable 6*, Oct. 2009, The TIMMO Consortium.

[18] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extraction of end-to-end timing model from component-based distributed real-time embedded systems," in *Time Analysis and Model-Based Design, from Functional Models to Distributed Deployments (TiMoBD) Workshop*. Springer, October 2011, pp. 1–6.

[19] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, April 1994.

Extending real-time networks over WiFi: related issues and first developments

Tony Flores Pulgar, Jean-Luc Scharbarg, Katia Jaffrès-Runser, Christian Fraboul
University of Toulouse, IRIT / ENSEEIHT
2 Rue Charles Camichel - F-31061, Toulouse, FRANCE
firstname.lastname@enseeiht.fr

***Abstract*—The flexibility of wireless connectivity is appealing in the context of real-time industrial networks. This paper discusses the use of wireless communication protocols to interconnect remotely located fieldbuses. The focus of this paper is to analyze the feasibility and design issues related to this type of hybrid real-time network architecture. Investigations are presented by addressing an interconnection through the well-mastered WiFi technology. On an example architecture, we discuss the impact of the different distributed medium access protocols available (DCF, EDCA) on the real-time flows. We outline the main design issues related to these choices and illustrate them on a first case study where remotely located CAN buses are interconnected through an IEEE802.11g network in DCF mode. Using this very simple and cost-effective architecture, we show as a first result that transmitting soft real-time data over such an architecture is feasible.**

***Keywords*-Hybrid real-time networks, interconnection, IEEE802.11, DCF, EDCA**

I. Introduction

Industrial fieldbus technologies are widely rolled out to offer real-time communication capabilities on the factory floor. A large set of protocols offer deterministic and timely bounded transmissions using tailored medium access schemes and architectures (e.g. PROFIBUS, PROFINET, TTEthernet, etc.).

Recent developments for industrial communications consider introducing wireless transmissions into the global network architecture [1][2]. First studies have assessed the capabilities of mainstream wireless technologies such as WiFi (IEEE802.11 [3]), Bluetooth or ZigBee (IEEE802.15.4) [2] for real-time communications. In parallel, new real-time wireless protocols have been designed [4][5][6]. Recently, two TDMA-oriented solutions (WirelessHART and ISA100.11a) have been commercialized for factory automation applications [6]. The main pitfall of wireless communications is of course the increased unreliability the medium suffers from due to interference and pathloss compared to shielded wires.

Among others, one of the interesting benefits of wireless transmissions is to provide a cost-effective network to interconnect distant heterogeneous or homogeneous legacy fieldbuses. The focus of this paper is to discuss this use case of wireless communications.

A wireless interconnection will benefit architectures where several fieldbuses, located far from each other, need a backhaul network to exchange data. Depending on the application, this data may be time-sensitive in the hard or in the soft sense. Either legacy wireless technologies such as WiFi (IEEE802.11 technology) or dedicated wireless protocols such as WirelessHART may be chosen, depending on the nature of the traffic exchanged between the remote buses.

For hard real-time data, a dedicated reliable wireless solution has to be picked, while for soft real-time data, a cheaper and probably less reliable wireless technology can be chosen. This paper focuses on this last case, where of-the-shelf WiFi controllers are used to interconnect remote real-time buses. Several Medium Access Control (MAC) protocols are available with the IEEE802.11 technology. A first discussion presents the benefits and issues related to these MAC protocols to carry soft real-time data on an example hybrid network architecture.

This discussion is then illustrated with a first case study where remotely located CAN buses [7] are interconnected through an IEEE802.11g network using a decentralized MAC protocol relying on CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). Using this very simple and cost-effective architecture, we show that transmitting soft real-time data over such an architecture is feasible, provided some design issues are correctly accounted for (e.g. gateway policies, wireless network load, interference, ...).

Section II illustrates the hybrid architecture we consider with an example network. Then, Section III discusses the problem of using WiFi to interconnect the remote real-time networks using different distributed IEEE802.11 MAC protocols. Next, Section IV presents the first results of extending CAN networks over IEEE802.11 DCF. Finally, Section V concludes this paper.

II. Target interconnection architecture

An example of the type of hybrid architecture of interest in this paper is described in Figure 1. In this example, four real-time buses are interconnected through a wireless local area network that follows the mainstream IEEE802.11 standard [3].

To interconnect the real-time buses of Figure 1, four dedicated wireless gateways are implemented composed of a real-time controller and a wireless IEEE802.11 interface. Additional wireless transmitters are represented in this architecture, emitting pure wireless traffic with non real-time guarantees. All wireless transmitters may be connected in ad hoc mode (distributed medium access) or using a central controller localized in an Access Point (AP). Such an AP is not represented in Figure 1 but may be necessary

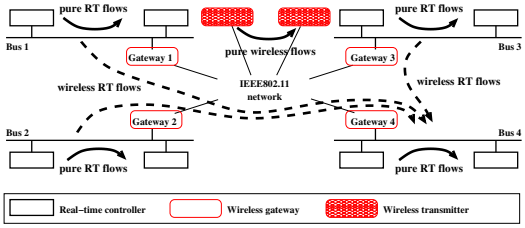


Figure 1. Interconnection architecture overview

for the study of centralized medium access versions of IEEE802.11 or even enhanced distributed ones.

A. Transmitted flows

Three types of flows are depicted in Figure 1:

- **pure RT flows** are periodic real-time flows local to the real-time bus. They *never* transit on the wireless network.
- **pure wireless flows** are *non real-time* Poisson flows local to the wireless network. They are competing for the medium with the wireless real-time flows emitted by the gateways.
- **wireless RT flows** are periodic real-time hybrid flows transmitted between controllers connected to different real-time buses: they transit on both wired and wireless networks via the gateways and are time-constrained.

Unique to pure and wireless RT flows, all frames have a critical delay which is the maximum allowed duration between their generation and reception times at their destination controllers.

III. Interconnection via WiFi

A. IEEE 802.11 MAC protocol overview

IEEE 802.11-2012 [3] defines several standards to offer a wireless connectivity at transmission rates ranging from 11Mbps (e.g. legacy versions such as IEEE802.11b) up to 600Mbps (IEEE802.11n). Bandwidth increase is due to improvements at the physical layer (OFDM, larger bandwidth, MIMO transmissions, etc.). Next generation physical layers are expected to increase data rates beyond 600Mbps (cf. IEEE802.11ac).

The fundamental medium access in IEEE 802.11 is a Distributed Coordination Function (DCF) which is a distributed random access scheme based on CSMA/CA. Additional protocols are defined to meet specific requirements but all use the service provided by DCF as shown on the MAC architecture in Fig. 2. For instance, the Point Coordination Function is a centralized protocol where an Access Point (AP) provides a contention-free medium access. Quality of Service (QoS), where channel access is differentiated for different classes of service, is introduced with the Hybrid Coordination Function (HCF) either in a distributed manner (EDCA protocol) or in a centralized manner (HCCA protocol). Finally, wireless meshed networks architectures can be handled in a controlled fashion with the MCCA protocol.

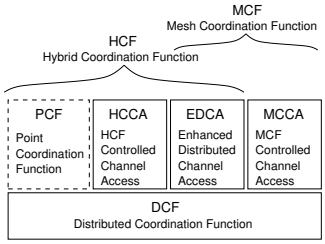


Figure 2. IEEE802.11 MAC (source: [3])

This paper discusses the use of the distributed medium access protocols (DCF and EDCA) since they are widely deployed and available for of-the-shelf designs. Thus, since these access are random based, only soft-real time data can be envisioned on this type of architecture. For more real-time constrained data, it is clear that further studies using centralized HCCA or even MCCA protocols should be performed.

B. Distributed medium access with IEEE802.11

This section introduces the main characteristics of the distributed MAC protocols of IEEE802.11, namely DCF and EDCA protocols. It discusses their implementation aspects in the context of real-time bus interconnection.

In DCF mode: a station performs carrier sensing to detect ongoing communications. If the channel is free for a period of time called Distributed InterFrame Space ($DIFS$), it transmits its frame immediately. If the channel is sensed busy, it defers its transmission until the end of the current transmission. Then, the station selects a random backoff b following an exponential backoff scheme. If the medium is idle for a $DIFS$ period of time, the backoff is decremented every *aSlotTime* duration. The backoff interval time is decremented as long as the channel is idle and is frozen as the node detects a transmission. At the end of this transmission, when the channel remains idle during $DIFS$, decrementation resumes. As b reaches zero, the transmission is attempted immediately by the station. After a successful transmission, the receiver sends an ACK after a duration called Short Inter Frame Space ($SIFS$) If no ACK is received by the emitter after an Extended Interframe Space ($EIFS$), the transmission is tried again.

A new backoff b is then uniformly chosen in the range $[0, w-1]$ where w is the contention window. This window depends on the number of failed attempts experienced by the current transmission. At the first attempt w is equal to the minimum contention window CW_{min}. Each unsuccessful transmission involves the multiplication of w by 2 until a maximum value of CW_{max} is reached.

DCF mode doesn't guarantee any prioritized access for real-time wireless frames. Thus, the real time traffic created by the gateways is directly competing with the

pure wireless flows of non-real time nodes. In order to achieve a soft real-time guaranty, the interconnection policies implemented at the wireless gateways have to be designed with the aim of reducing the number of collisions of RT flows that occur on the wireless medium with DCF mode. This can be arranged by selecting, in a timely manner, the RT frames to be encapsulated in the wireless RT frames as we will investigate in Section IV.

In EDCA mode: QoS is being introduced on top of DCF so as to provide different levels of priority to wireless transmissions. A wireless node requests a transmission opportunity (TXOP) which is an interval of time when a particular quality-of-service (QoS) station (STA) has the right to initiate frame exchange sequences onto the wireless medium. A TXOP is defined by a starting time and a maximum duration. The TXOP is obtained by a station by successfully contending for the channel in EDCA. During an EDCA TXOP, a STA may initiate multiple frame exchange sequences.

EDCA defines four access (or traffic) categories ranging from background (lowest priority), best effort, video to voice traffic (highest priority). All nodes emitting flows in the *same* access category AC contend for TXOPs together using their own set of EDCA parameters which are given by the triplet (AIFS[AC], $CW_{min}[AC]$, $CW_{max}[AC]$). The AC with the highest priority has the lowest parameter values among all ACs so as to have a higher probability to access channel first if contending with lower priority AC data frames. If there are frames with different ACs waiting in the output queue of at the same STA, collisions between them are resolved within the STA so that the data frames of higher priority compete for the channel. Remaining lower priority frames behave as if there was an external collision on the medium.

Interconnecting real-time buses using EDCA will be more challenging than using DCF because several options are available. First, besides defining encapsulation policies, the wireless RT flows have to be properly allocated to ACs. This can be done statically by the gateways. Another more subtile adaptation resides in the development of a dynamic adjustment of the EDCA parameters for each AC based on end-to-end delay statistics and missed-deadline rates, using monitoring. This last issue is an ongoing work which is not further addressed in this paper.

IV. A FIRST STUDY: CAN OVER DCF

Interconnection of a real-time bus with IEEE802.11 in DCF mode is investigated in the following. The real-time bus of interest is the well known Controller Area Network (CAN) [7] standard.

A. Investigated architecture

CAN is one of the mainstream standards for embedded communications. Despite the fact that it has been originally developed for automotive communications, CAN has found its place in factory automation applications to handle sensor-actuator communications because of its ease of use and the low cost of its controllers. We recall that CAN medium access is based on CSMA/CR (with Collision Resolution): the start of frame transmissions on the bus are synchronous. When two or more stations start a transmission simultaneously, the one with the smallest frame identifier wins and the others stop their transmission. This mechanism guarantees strict priority order on identifiers. It implies limitations on the bandwidth and the maximal length of the bus (e.g. 1 Mbps for 40 meters). Bit-stuffing is used to avoid the transmission of long sequences of bits with identical value. The computation of the frame length has to take into account these additional bits. In this paper, we use the upper bound given in [8]. The length C of a frame carrying x bytes of data is:

$$C = (55 + 10 \times x) \tag{1}$$

In this example, all wireless nodes (gateways, pure wireless emitters) function in ad hoc mode using DCF medium access protocol following the specification of the OFDM-PHY layer of 802.11g (20 MHz channel spacing). Since WiFi access points are static, we can consider that they are located at a distance where they can operate at the highest rate of 54 Mbps. We assume proper channel assignment has been performed so as to mitigate inter-node interference. In this ideal case study, transmissions are error-free and the transmission duration (in μs) at 54 Mbps of a MPDU of x bytes is derived according to [3]:

$$d(x) = 20 + 4 \left\lceil \frac{22 + 8(34 + x)}{216} \right\rceil \tag{2}$$

Following timing parameters are assumed: $SIFS = 16\mu$s, $DIFS = 34\mu$s, $EIFS = 78\mu$s, $CW_{min} = 16$ and $CW_{max} = 1024$.

B. Wireless gateway policies

As stated earlier, for DCF mode, interconnection policies at the gateway are crucial to ensure a timely distribution of wireless RT frames. Interconnection is done by encapsulating CAN frames of the wireless RT flows into IEEE802.11 frames. Encapsulation is chosen to facilitate the addressing of flows in the global network, and thus a basic static switching table is stored at the gateways to route wireless RT flows.

A gateway encapsulation strategy is characterized by the following parameters:

- The maximum number $N_{\ell,m}$ of CAN frames which can be encapsulated by gateway ℓ for destination gateway m in one IEEE802.11 frame,
- The maximum time $Wmax_j$ a frame of a wireless RT flow fH_j can wait in its source gateway.

C. Results

Table I summarizes the strategies analyzed in this study on the illustrative configuration of Figure 1. Each CAN bus carries 4 pure RT flows. Each bus 1, 2 and 3 are respectively the source of 8 wireless RT flows each which are all transmitted to bus 4. These wireless RT flows have a period and a critical delay of 10ms each. The size in bytes for each of these 8 flows are equally distributed between 2, 4, 6 and 8 data bytes.

The encapsulation strategies of Table I are considered, where uniform timers are assigned to the flows. T0 denotes a basic strategy where one CAN frame is encapsulated in exactly one wireless frame. T2, T3 and T4 are purely timed strategies where CAN frames wait for at most 2, 3 and $4ms$ respectively.

T0	$N_{\ell,m} = 1$	$Wmax_j = 0\ ms\ \forall j$
T2	$N_{\ell,m} = 100$	$Wmax_j = 2\ ms\ \forall j$
T3	$N_{\ell,m} = 100$	$Wmax_j = 3\ ms\ \forall j$
T4	$N_{\ell,m} = 100$	$Wmax_j = 4\ ms\ \forall j$

Table I
SIMULATED STRATEGIES

A quantitative analysis of the proposed bridging strategies has been conducted. This analysis is based on simulations. Therefore, a home-made simulation tool has been developed using QNAP2 [9]. Table II gives results concerning the utilization of the wireless medium. The study is conducted with different numbers of identical pure wireless flows fW (between 4 and 7), all of them with an average period of 2ms and for packets of 200 data bytes. Simulations are conducted for multiple instances where offsets of real-time flows are randomly chosen between 0 and their period.

		T0	T2	T3	T4
$\frac{\sum fH_j}{fW_1}$		4.8	1.9	1.4	1.2
4 fW flows	% Col	2.2	0.8	0.8	0.7
	AvgD	250	169	162	158
5 fW flows	% Col		1.3	1.2	1.1
	AvgD		199	189	184
6 fW flows	% Col		2.0	1.8	1.6
	AvgD		242	227	220
7 fW flows	% Col			2.7	2.5
	AvgD			287	274

Table II
IEEE802.11 RESULTS

The first line of the table shows the relative number of IEEE802.11 frames generated by wireless RT flows. The number of IEEE802.11 frames decreases when the value of the timer increases. Indeed, the average number of wireless RT frames encapsulated in an IEEE802.11 frame increases when the timer increases. Table II also shows the percentage of collisions (% Col) and the average delays (AvgD) of pure wireless flows for each strategy and each simulated scenario (i.e. number of pure wireless flows). Empty values mean that the wireless network is saturated and transmission delays diverge. It can be noticed that these percentages of collisions and delays are deeply linked to the number of contending IEEE802.11 frames.

Table III presents the rates of wireless RT frames which miss their deadlines. More precisely, each value in Table III is the average number of wireless RT frames that miss their deadlines when 10^6 of such frames are generated and transmitted. The main result of Table III is that it is still possible to reach a reasonably low value of missed deadlines (up to 200 every 10^6 frames) with the use of DCF when a limited number of pure wireless flows (up to 7) is present in the network.

It can be inferred from these results as well that there is a trade-off between the time spent waiting at the gateway due to $Wmax_j$ and the time spent contending for the wireless medium. In this very simple example, it can be seen that the best trade-off is obtained with T3

nb. of fW flows	T0	T2	T3	T4
4	1	0	0	0
5		0	0	3
6		21	19	20
7			200	200

Table III
MISSED DEADLINES FOR WIRELESS RT FLOWS

V. CONCLUSION

The motivation of this paper is to highlight that IEEE802.11 technology is a credible solution for interconnecting remote field buses when soft real-time data are exchanged between these buses. Future work needs to discuss the improvements provided by enhanced medium access protocols of IEEE802.11, including QoS capabilities and/or centralized mechanisms. Even though IEEE802.11 is the mainstream wireless technology, it is topical as well to evaluate the benefits of a dedicated wireless real-time protocol such as WirelessHART or ISA100.11a.

REFERENCES

[1] F. De Pellegrini, D. Miorandi, S. Vitturi, and A. Zanella, "On the use of wireless networks at low level of factory automation systems," *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 2, pp. 129–143, 2006.

[2] A. Willig, K. Matheus, and A. Wolisz, "Wireless technology in industrial networks," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1130–1151, 2005.

[3] "802.11-2012 - IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks
Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2012.

[4] W. Ng, C. Ng, B. Ali, and N. Noordin, "Performance evaluation of wireless controller area network (wcan) using token frame scheme," *Wireless Personal Communications*, pp. 1–27, 2013.

[5] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, and M. Nixon, "Wirelesshart: Applying wireless technology in real-time industrial process control," in *RTAS'08*, april 2008, pp. 377–386.

[6] S. Petersen and S. Carlsen, "WirelessHART versus ISA100.11a: The format war hits the factory floor," *Industrial Electronics Magazine, IEEE*, vol. 5, pp. 23–34, Dec. 2011.

[7] Bosch, "CAN Specification version 2.0," Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart, 1991.

[8] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller Area Network (CAN schedulability analysis: refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[9] Simulog, "Qnap2," http://www.simulog.fr.

Towards resilient real-time wireless communications

Jeferson L. R. Souza and José Rufino
University of Lisboa - Faculty of Sciences
LaSIGE - Navigators Research Team
Email(s): jsouza@lasige.di.fc.ul.pt, ruf@di.fc.ul.pt

***Abstract*—The use of wireless networks on environments with real-time constrains requires timeliness and dependability guarantees to allow the execution of time-restricted networked operations. The current wireless standards and state-of-the-art solutions pay no or little attention to dependability aspects of communications, which are fundamental to secure any timeliness guarantee. This paper presents an innovative standard-compliant solution, dubbed *Mediator Layer*, which extends the MAC sublayer with additional components that can be incorporated within networked simulators and implemented in currently available hardware platforms. Thus, the *Mediator Layer* enhances and complements the services traditionally offered by wireless network standards, providing a set of fundamental abstractions useful for both system design and application programming.**

***Index Terms*—wireless sensor and actuator networks, real-time communications, dependability, timeliness, resilience.**

I. Introduction

The use of wireless networks on environments where communications may have real-time requirements is a current trend [1]. This trend is guided by the needs to reduce system size, weight, and power consumption (SWaP) without lessen timeliness and dependability guarantees. Industrial, vehicular, and aerospace environments are waiting for a solution to address an effective and efficient real-time support on wireless communications. For example, [2] discusses a set of pressing challenges on the use of wireless sensor networks (WSNs), more specifically wireless sensor and actuator networks (WSANs), in industrial automation, where dependability and real-time guarantees are a must.

So far, a lot of work has been done, proposing new medium access control (MAC) protocols [3]–[10], modifications on the existent standards [11]–[13], and abstract models [14] trying to enhance the reliability on wireless communications. However, all these works focus their analyses in the temporal aspects of the frame transmission service, paying no or little attention to the dependability aspects of communications, which are fundamental to secure any timeliness guarantee. An interesting conclusion draw in [2] is the necessity of improvements on the existent standards face to requirements of safe and secure networked operations.

Thus, this work-in-progress presents an innovative solution dubbed *Mediator Layer* [15], which is capable to enhance the dependability and timeliness of MAC sublayer services. In addition, a set of constructs useful to the programming of distributed applications, such as reliable communications, node failure detection, membership and clock synchronization may be offered immediately above the MAC sublayer. In this sense, our solution enhances and complements the services traditionally offered by wireless network standards, providing a set of fundamental abstractions useful for both system design and application programming. A prototype of the *Mediator Layer* is being integrated in the NS2 simulator [16] and in a commercial off-the-shelf (COTS) platform [17], using the IEEE 802.15.4 standard as a case study.

The presentation of our advances is organized as follows: Section II presents the system model. Sections III and IV describes our work-in-progress solution, including the incorporation of the *Mediator Layer* in the NS2 simulator and in the hardware platform. Finally, Section V presents some conclusions and future directions of this work.

This work was partially supported: by the EC, through project IST-FP7-STREP-288195 (KARYON); by FCT/DAAD, through the transnational cooperation project PROPHECY; and by FCT, through the project PTDC/EEI-SCR/3200/2012 (READAPT), the Multiannual Funding Program, and the Individual Doctoral Grant SFRH/BD/45270/2008.

II. System model

In this section we provide a brief description of our system model, which establishes a base foundation for our design and simulations. Our system model is formed by a set of wireless nodes $X = \{x_1, x_2, \ldots, x_n\}$, being $1 < n \leq \#A$, where A is the set of all wireless nodes using the same communication channel. A wireless node is a networked device capable to communicate with other wireless nodes. The set of nodes X itself defines a node relationship entity dubbed wireless network segment (WnS), which is established by all wireless nodes within $X \subseteq A$ that use a given communication channel and share a single hop communication range space.

For any WnS we use the following assumptions:

1) the communication range of X, i.e. its broadcast domain, is given by: $B_X = \bigcap_{j=1}^{n} B_D(x_j),\ \forall x_j \in X$, where $B_D(x_j)$ represents the communication range of node x_j;
2) $\forall x \in X$ can sense the transmissions of one another;
3) $\forall x \in A,\ x \in X \iff B_D(x) \bigcap B_X = B_X$ or, as a consequence of node mobility, $x \notin X \iff B_D(x) \bigcap B_X \neq B_X$;
4) $\exists x \in X$ which is the coordinator, being unique and with responsibility to manage the set X;
5) a network component (e.g. a node $x \in X$) either behaves correctly or crashes upon exceeding a given number

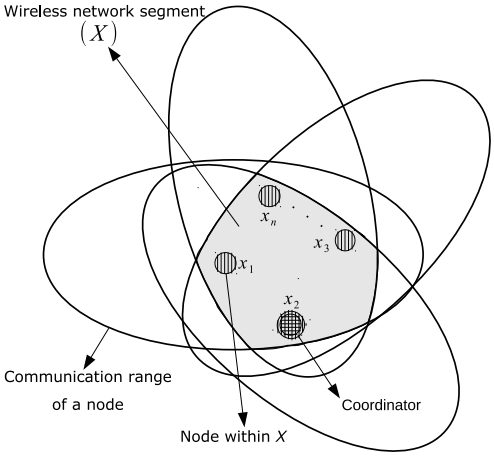


Fig. 1: Representation of a wireless network segment

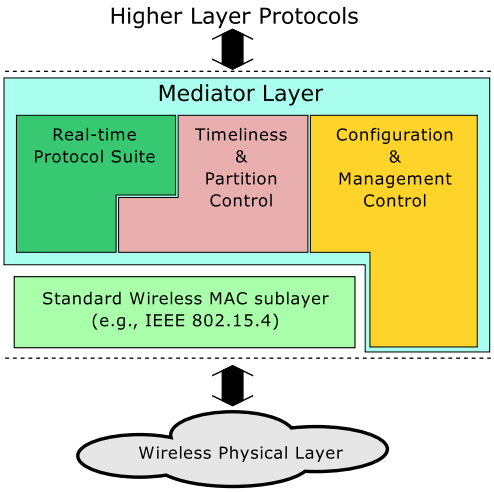


Fig. 2: The Mediator Layer and its components

of consecutive omissions (the component's *omission degree*, f_o) in a time interval of reference[1], $\mathcal{T}_{rd}$;

6) failure bursts never affect more than f_o transmissions in a time interval of reference, $\mathcal{T}_{rd}$;

7) omission failures may be inconsistent (i.e., not observed by all recipients).

Assumptions 1 to 3 define the physical relationship between nodes within the WnS. Our system model characterizes the relationship between nodes at MAC sublayer, where nodes must be in the communication range of each other to communicate, and are able to sense one another (assumption 2). Mobility may drive nodes away of the WnS (assumption 3).

In the context of network components, an omission is an error that destroys a data frame. Establishing a bound for the *omission degree* (assumptions 5 and 6) provides a general method for the detection of failed components. If each omission is detected and accounted for, the component fails once it exceeds the *omission degree bound*, k. The *omission degree* is thus a general measure of the reliability of network components with respect to accidental/intentional transient errors.

Figure 1 presents a graphical representation of a wireless network segment. In this figure we can see the communication range of each node within X, evidencing the intersection between all communication ranges of all nodes, which delimits the broadcast domain of X. One node within X assumes the role of coordinator (assumption 4). The management activities of the coordinator comprises the assignment of the current communication channel in use by the WnS, the allocation of guaranteed slots for frame transmissions, and so on.

[1]For instance, the duration of a given protocol execution. Note that this assumption is concerned with the total number of failures of possibly different wireless nodes.

III. Towards resilient real-time wireless communications

Our approach to enhance the dependability and timeliness of wireless communications consists of an extensible component layer, dubbed *Mediator Layer*, build around a standard MAC sublayer. No modification is required to existing standards, being the *Mediator Layer* standard-compliant solution easily implemented in currently available COTS platforms. In this sense, the *Mediator Layer* approach significantly differs from other solutions described in the literature [3]–[14].

The *Mediator Layer* provides enhanced frame transmission and management services through a minimal set of fundamental components (draw in Fig. 2), which handle the actions required to secure dependability and timeliness in communications.

The *Real-Time Protocol Suite* (Fig. 2) component is responsible for handling frame transmissions, which can be data or management frames. Different protocols serving requests with different types of requisites, ranging from unreliable unicast to reliable broadcast, can be incorporated within this component. This augments the applicability of wireless networks to broader class of applications, including those with mixed-criticality requirements. Our intuition is that replacing a classical timeout-based protocol design approach with a combination of positive and negative confirmations (acknowledgements) contributes to reduce protocol worst case termination times, and thus service timeliness. Timeout-based techniques may still be used but only to detect node crash failures.

The *Timeliness and Partition Control* (Fig. 2) component deals with the temporal aspects related to the frame transmission service. Controlling and monitoring the timing of the actions within the *Mediator Layer* relies on an internal *Time Service* that ensures the temporal awareness of all networked operations, including the occurrence of temporary network

partitions (i.e., network inaccessibility [18]) or even timeliness violations of a specific transmission protocol. An example is the monitoring and verification of a deadline associated to a data request, which may be disturbed by inaccessibility incidents. The dynamic control of inaccessibility periods [15] allows protocol execution to be aware of the real duration of inaccessibility incidents, and to (self-)adapt its execution to actual network operating conditions. In particularly, optimal timeout values can be used in the dimensioning of protocol timers, which contributes to enhance the timeliness guarantees.

The *Configuration and Management Control* (Fig. 2) component manages and controls the configuration of all parameters of the MAC sublayer and the internal parameters of the *Mediator Layer*, respecting application requirements, resource limitations, and environment restrictions. Such requirements, limitations, and restrictions should be defined in an profile that is utilised to adjust the aforementioned parameters. Autonomous methods may be utilized by the *Configuration and Management Control* components, making configuration procedures self-adaptive, self-managed, and self-controlled.

IV. Building *Mediator Layer* components

In order to illustrate how to improve the dependability and timeliness properties of a standard MAC sublayer, we select a set of key mechanisms that needs to be included in the *Mediator Layer* for that purpose.

The first mechanism we address introduces a minor extension in the standard frame check sequence (FCS) mechanism. The native FCS mechanism silently discards every frame received with errors, meaning omission errors will not be perceived, nor monitored nor accounted for. Conversely, the extension specified in Algorithm 1 provides a management indication of the status of the received frame to the *Mediator Layer*, even if the frame was received with errors. The management indication highlighted in line 15 signals: the instant the frame was received, represented by the variable $time\_stamp$, as obtained in line 6; the frame header, represented by the variable $frame\_header$, which is extracted in line 7; and the FCS error status represented by the variable fcs_error. This simple extension of the native standard mechanism enriches the monitoring capabilities of the *Mediator Layer*.

The second mechanism we address takes profit of the FCS extension mechanism to account for omission errors. The accounting of omission errors is expressed as a pseudo-code presented in Algorithm 2. The status of each frame received by a node is signalled through a MAC management indication in line 7. If a frame was received with errors, the number of consecutive omission errors, O_{degree}, is incremented by one (line 9). Otherwise, i.e., whenever a frame is received without errors, the value of O_{degree} is cleared (line 11). Should the omission degree bound, k, be exceeded, a management indication is provided (line 14). This is an indication that the communication channel in use has failed and that a channel switch action should be issued to MAC sublayer management entities.

Algorithm 1 Extending the FCS mechanism

```
1: Initialization phase.
2: fcs_error ←false;
3: Begin.
4: loop
5:     when Channel.indication(frame) do
6:         time_stamp ←MLA.get.time();
7:         frame_header ←MAC.get.header(frame);
8:         if MAC.FCS.check(frame) is OK then
9:             fcs_error ← false;
10:            MAC.indication(frame);
11:        else
12:            fcs_error ← true;
13:            MAC.discard(frame);
14:        end if
15:        MAC.Mgmt.indication(time_stamp, frame_header, fcs_error);
16:    end when
17: end loop
18: End.
```

Algorithm 2 Accounting local omission errors

```
1: Initialization phase.
2: O_degree ← 0;
3: k ← The value of the omission degree bound depends on environment
   conditions. The default value utilized within the IEEE 802.15.4 standard
   is k = 3.
4: current_channel ← It represents which is the current channel in use.
5: Begin.
6: loop
7:     when MAC.Mgmt.indication(time_stamp, frame_header, fcs_error)
       do
8:         if fcs_error = true then
9:             O_degree ← O_degree + 1;
10:        else
11:            O_degree ← 0;
12:        end if
13:        if O_degree > k then
14:            MLA.Mgmt.indication(time_stamp,current_channel,
               O_degree_exceeds_k)
15:        end if
16:    end when
17: end loop
18: End.
```

The real-time operation of nodes within the WnS is also monitored by a node failure detection service, which is part of the *Real-time Protocol Suite* component. The node failure detection is presented in Algorithm 3. This algorithm resides within every node of a WnS, establishing a distributed and decentralized node failure detection service.

Each node locally accounts omissions from other nodes of the WnS as follows: an indication representing the status of a received frame is detected (line 6). The source node index of the received frame is extracted from the $time\_stamp$ and $frame\_header$ variables (line 7). If a valid node index is found (e.g., the node source identifier in the frame header matches the foreseen time slot represented here by the $time\_stamp$ variable). If the frame was received with errors, the omission degree of this node is incremented by one (line 10), otherwise it is reseted to zero (line 12). When the omission degree of a given source node of a received frame exceeds k, a local indication is generated by the node failure detection service (line 15), which can be utilized by a membership

Algorithm 3 Node failure detection

* The detection of a node's crash is intentionally ommitted

1: Initialization phase.
2: $k \leftarrow 3$;
3: $node_{index} \leftarrow$ *It represents the index of a node.*
4: Begin.
5: **loop**
6: **when** *MAC.Mgmt.indication*$(time\_stamp, frame\_header, fcs\_error)$ **do**
7: $node_{index} \leftarrow$ *MLA.Mgmt.getNode*$(time\_stamp, frame\_header)$;
8: **if** $node_{index}$ *is valid* **then**
9: **if** $fcs\_error = \text{true}$ **then**
10: $O_{degree}[node_{index}] \leftarrow O_{degree}[node_{index}] + 1$;
11: **else**
12: $O_{degree}[node_{index}] \leftarrow 0$;
13: **end if**
14: **if** $O_{degree}[node_{index}] > k$ **then**
15: *MLA.FD.indication*$(time\_stamp, current\_channel, node_{index}, O_{degree}[node_{index}]\_exceeds\_k)$
16: **end if**
17: **end if**
18: **end when**
19: **end loop**
20: End.

service to help the update of the view that represents the active nodes within the WnS.

As a proof-of-concept the *Mediator Layer* has been implemented and incorporated within both NS2 simulator [16] and a COTS platform [17], using the IEEE 802.15.4 standard as a case study. The results achieved so far have shown that these mechanisms allow to achieve optimal latencies with respect to the detection of channel failures (Algorithm 2) and node failures (Algorithm 3). Any violation of the omission degree bound is also detected as soon as it occurs.

This happens because frame omissions are monitored and detected at the lowest level of communications (Algorithm 1). The effectiveness of this approach is illustrated in Fig. 3. In Fig. 3 the blue color represents the number of frames (621) received without any error (i.e., indicated with or without the presence of *Mediator Layer*), while the orange color represents the number of frames (270) received with errors, and detected by *Mediator Layer*.

V. Conclusion and Future Work

This paper presented an extensible component layer dubbed *Mediator Layer*, which enhances the timeliness and dependability properties of wireless communications. A dependable and timely service at lowest level of the network protocol stack helps the higher level protocol designers to keep their solutions as simple as possible, using the easy-to-design building block approach enabled by the *Mediator Layer*. Our standard-compliant approach has been implemented and incorporated within the NS2 network simulator and a COTS platform.

Future directions include, but are not limited to: adding protocols to the *Real-Time Protocol Suite* component to cover the requirements imposed by real-time environments; finishing the implementation and incorporation of the *Mediator Layer* within the NS2 network simulator and the COTS platform; performing analyses and evaluation of the *Mediator Layer* approach in such both simulator and COTS platform face to different error conditions and temporal requirements of environments and applications; and defining relevant real-time metrics to evaluate the wireless communications with regard to application requirements and environment restrictions.

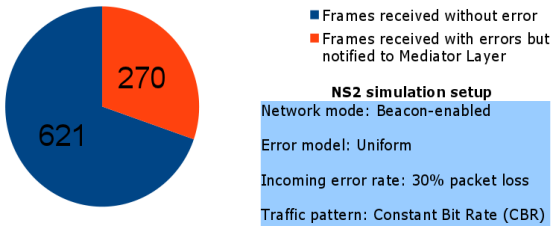


Fig. 3: Results obtained with the extension of the FCS mechanism implemented within NS2 simulator

References

[1] T. Stone, R. Alena, J. Baldwin, and P. Wilson, "A viable COTS based wireless architecture for spacecraft avionics," in *IEEE Aerospace Conference*, 2012, pp. 1–11.

[2] J. Åkerberg, M. Gidlund, and M. Björkman, "Future research challenges in wireless sensor and actuator networks targeting industrial automation," in *9th IEEE INDIN*, July 2011.

[3] A. Sahoo and P. Baronia, "An energy efficient MAC in WSN to provide delay guarantee," in *15th IEEE LANMAN*, June 2007.

[4] I. Aad, P. Hofmann, L. Loyola, F. Riaz, and J. Widmer, "E-MAC: Self-organizing 802.11-compatible MAC with elastic real-time scheduling," in *IEEE MASS*, October 2007.

[5] E. E-López, J. V-Alonso, A. M-Sala, J. G-Haro, P. P-Mariño, and M. Delgado, "A WSN MAC protocol for real-time applications," *Personal Ubiquitous Computing Journal*, January 2008.

[6] P. Bartolomeu, J. Ferreira, and J. Fonseca, "Enforcing flexibility in real-time wireless communications: A bandjacking enabled protocol," in *IEEE ETFA*, September 2009.

[7] X.-Y. Shuai and Z.-C. Zhang, "Research of real-time wireless networks control system MAC protocol," *Journal of Networks*, April 2010.

[8] T. Zhou, H. Sharif, M. Hempel, P. Mahasukhon, W. Wang, and T. Ma, "A novel adaptive distributed cooperative relaying MAC protocol for vehicular networks," *IEEE Journal on Sel. Areas in Comm.*, Jan 2011.

[9] M. Sha, G. Hackmann, and C. Lu, "Arch: Practical channel hopping for reliable home-area sensor networks," in *17th IEEE RTAS*, April 2011.

[10] X. Zhu, S. Han, P.-C. Huang, A. Mok, and D. Chen, "MBStar: A real-time communication protocol for wireless body area networks," in *23rd ECRTS*, July 2011.

[11] Yu-Kai, Ai-Chun, and Hui-Nien, "An adaptive GTS allocation scheme for IEEE 802.15.4," *IEEE Trans. on Parallel and Distributed Systems*, May 2008.

[12] M. Hameed, H. Trsek, O. Graeser, and J. Jasperneite, "Performance investigation and optimization of IEEE 802.15.4 for industrial wireless sensor networks," in *IEEE ETFA*, September 2008.

[13] A. Koubâa, A. Cunha, M. Alves, and E. Tovar, "i-GAME: An implicit GTS allocation mechanism in IEEE 802.15.4, theory and practice," *Springer Real-Time Systems Journal*, August 2008.

[14] F. Kuhn, N. Lynch, and C. Newport, "The abstract MAC layer," in *23rd DISC*, September 2009.

[15] J. L. R. Souza and J. Rufino, "An approach to enhance the timeliness of wireless communications," in *5th UBICOMM*, Lisbon, 2011.

[16] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Springer, 2009.

[17] ATMEL, *ATMEL AVR2025: IEEE 802.15.4 MAC Software Package - User guide*, ATMEL Coorporation, May 2012.

[18] J. L. R. Souza and J. Rufino, "Characterization of inaccessibility in wireless networks - a case study on IEEE 802.15.4 standard," in *3th IFIP IESS*, September 2009.