Licentiate Thesis Draft

Integrating Elastic Real-Time Applications on Fog Computing Platforms

Shaik Mohammed Salman
School of Innovation, Design and Engineering (IDT)
Mälardalen University
shaik.mohammed.salman@mdh.se

Main Supervisor: **Thomas Nolte**
Co-supervisors: **Alessandro V. Papadopoulos**
Saad Mubeen
Filip Marković
Industry Mentor: **Anders Wall**

Abstract

Real-time systems such as industrial robots and autonomous navigation vehicles integrate a wide range of algorithms to achieve their functional behaviour. In certain systems, these algorithms are deployed on dedicated computational resources and exchange information over a real-time network. With the availability of modern multi-processors, there has been a growing interest in transitioning towards an integrated architecture where these algorithms can be executed on a shared computational resource. The technology enabling such transition is focused around the virtualization of the computational resources that can provide spatial isolation and temporal partitioning to these software applications. Although many useful solutions such as resource reservations and hierarchical scheduling have been proposed to facilitate virtualization for real-time applications, the current state-of-the-art addresses mostly those applications whose tasks can be specified according to the periodic or the sporadic task model. As the computational demand of many control algorithms can be adjusted flexibly at runtime, for example, by changing their periods, they can be better modeled using the elastic task model, thereby reducing the pessimism inherent with the periodic or the sporadic task model.

Therefore, in this thesis, we first propose a scheduling framework for elastic real-time applications with reservations based on the periodic resource supply for uniprocessor systems and then extend this solution to a multiprocessor scenario where the reservation is based on the minimum parallelism supply form. Concurrently, since many existing software applications have been designed to run exclusively on dedicated single cores, we provide a systematic methodology to guide the migration of an existing real-time software application from a single-core to a multi-core platform with emphasis on architecture recovery of the existing software and its transformation for implementation on a multi-core platform. Furthermore, the advantages provided by cloud architectures for non-real-time applications has prompted discussions around the possibility of providing similar advantages to cyber physical systems such as industrial robots. Since virtualization is a key enabler for such architectures, we investigate the advantages

of a fog-based architecture over an existing architecture of the robot controllers and identify key research challenges that need to be addressed for its successful implementation.

Acknowledgment

Thank you all!

Shaik Mohammed Salman
Västerås, October, 2021

List of Publications

Papers included in this thesis[1]

Paper A: Shaik Mohammed Salman, Filip Markovic, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *Scheduling Elastic Applications in Compositional Real-Time Systems* In the proceedings of the IEEE 26th International Conference on Emerging Technologies and Factory Automation (ETFA 2021).

Paper B: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte *Multi-Processor Scheduling of Elastic Applications in Compositional Real-Time Systems* In the Journal of Systems Architecture, Dec 2021.

Paper C: Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. *A Systematic Methodology to Migrate Complex Real-time Software Systems to Multi-Core Platforms* In the Journal of Systems Architecture, Volume 117, 2021.

Paper D: Shaik Mohammed Salman, Vaclav Struhar, Alessandro V. Papadopoulos, Moris Behnam, Thomas Nolte. *Fogification of Industrial Robotic Systems: Research Challenges* In Proceedings of the Workshop on Fog Computing and the IoT.(IoT-Fog '19).

[1]The included papers have been reformatted to comply with the thesis layout.

Related publications, not included in this thesis

Paper 1: Salman, S. M., Sitompul, T. A., Papadopoulos, A. V., & Nolte, T. *Fog Computing for Augmented Reality: Trends, Challenges and Opportunities* IEEE International Conference on Fog Computing (ICFC), 2020.

Paper 2: Salman, S. M., Papadopoulos, A. V., Mubeen, S., & Nolte, T. *A Systematic Migration Methodology for Complex Real-time Software Systems* IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC), 2020.

Paper 3: Shaik, M. S., Struhár, V., Bakhshi, Z., Dao, V. L., Desai, N., Papadopoulos, A. V., Nolte, T., Karagiannis, V., Schulte, S., Venito, A., & Fohler, G. (2020). *Enabling Fog-based Industrial Robotics Systems.* In the proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2020.

Contents

I

Thesis

Chapter 1

Introduction

Integration of independent real-time applications on multiprocessor hardware platforms has been extensively studied with solutions ranging from reservation based scheduling to provide temporal partitioning[1, 2, 3] to cache partitioning [4, 5] and memory controller designs[6] to improve performance. Similar to single-core platforms, managing both overload and overrun situations [7] is also critical in realization of open real-time systems[8] on multiprocessor hardware platforms. For many real-time applications whose workload can be modelled as periodic or sporadic tasksets[9], the techniques of reservation [10] and hierarchical scheduling [11] can reduce the extent of the interference on co-executing applications while providing sufficient CPU time for the application to meet its timing requirements. Some real-time systems have temporal requirements that vary over time[7] and the computational needs of their algorithms may depend on the state of the environment the system is operating in [12]. In such cases, it may be necessary to describe the application behaviour through more *expressive* task models than the sporadic or the periodic task model for better resource usage[13]. For instance, the graph-based task models can be used to describe every possible job type that a task can release to improve resource utilization. Another task model that can be used to specify the varying temporal requirements is the elastic task model[14, 15], where the task parameters can be described as a range of values instead of a single value. This flexibility can be utilized at runtime to control the computation time allocated to the tasks depending on the existing system load. Moreover, the elastic task model and the associated load modification algorithms [15, 16] have been used to address the issue of overload management to ensure schedulability of soft real-time applications [7]. Combined approaches for overrun management through resource reservations and overload management through elastic task modelling and elastic compression algorithms have been proposed in the literature for systems executing

only a single application with multiple tasks [17, 18]. In the case of multiple independent applications, the focus has primarily been restricted to modifying the resource budget allocated to an application[19, 20, 21, 22, 23, 24]. As co-execution of independent applications on shared hardware is at the heart of the new distributed computing paradigms such as fog computing [25, 26]there is a need to support not only periodic or sporadic real-time workloads but also flexible workload models such as the elastic task model to address both overload and overrun scenarios for efficient resource utilization.

Concurrently, for some industrial real-time systems it may be beneficial to reuse much of their existing code even when transitioning to fog-based architectures. In many cases, the software is optimized for performance on single-core systems and new features are added in an incremental manner through the release of a new version. Redeveloping the complete application for a given architecture in such cases may not be an optimal choice. A systematic migration process that can ensure successful realization of the benefits offered by the new hardware while maximizing the reuse of existing code can be beneficial.

Furthermore, although fog and edge architectures have been proposed to bring the benefits of cloud computing to systems with real-time requirements [27, 28], their benefits over an existing architecture have not been investigated for a concrete real-time system.

In this context, the work in this thesis extends the current techniques for integrating elastic real-time applications on a shared hardware through a hierarchical scheduling framework where each elastic application is encapsulated in a reservation server while utilizing the elastic task compression and decompression algorithms for overload management within the reservation. The framework proposes the utilization of the periodic resource supply form reservation for uniprocessor systems and the minimum parallelism resource supply form combined with the periodic resource supply model for multiprocessor systems. Concurrently, since there are considerable differences between a single-core and a multi-core platform, we provide a systematic methodology to guide the migration of any existing controller software systems from the single-core to multi-core platforms with emphasis on architecture recovery of the existing software system and its transformation for implementation on a multi-core platform to meet its timing requirements. Furthermore, to investigate the advantages of fog computing architectures for real-time systems, we propose a fog-based architecture for an industrial robot controller software and compare it to an existing architecture of the controller while identifying some of the research challenges that need to be addressed for the fog architecture to be used in practice.

Thesis outline This licentiate thesis contains two parts. Part I is an overview of the thesis and is organized as follows. We first discuss background and related work to the thesis in Chapter 2. In Chapter 3, we provide an overview of the included papers and the contributions brought by each of them. In Chapter 4, we present conclusions and an outline of future work towards a doctoral thesis. Part II includes the collection of included papers.

Chapter 2

Background and Related Work

This chapter provides an overview of the relevant background and work related to contributions of the thesis.

2.1 Background

2.1.1 Temporal Isolation in Real-time Systems

Many cyber-physical systems are composed of algorithms that require different amounts of computing times for different scenarios and are flexible with the frequency with which they perform the computations. For example, in the case of autonomous driving software, it was shown that the execution times for different functions such as perception and planning varied considerably within the measured samples[12]. For a robot manipulator, it was shown that periods of some of the tasks were dynamically adjusted while still achieving the control objectives [29]. The need for adjusting the frequencies of some of the tasks of a mobile robot to react to dynamically changing environments of the robot was highlighted in [30]. In some scenarios, the variability can be upper bounded and the application's computational demand can be modeled as a periodic or sporadic taskset while still ensuring schedulability. Even if computational demand was modelled with worst-case assumptions, it is possible that at runtime, the values may deviate from the assumed values, resulting in violation of the timing requirements. In such a scenario, it is desirable to limit the impact of the deviation to only the tasks violating the assumptions, rather than the entire application. For applications with dedicated processors, this is normally achieved through temporal isolation[7] techniques such as reservation servers[31] with each server providing the reserved computation time to its associated task. For virtualized hardware platforms, temporal isolation

is provided through hierarchical scheduling techniques[32, 33, 11]. Here each application has its own local scheduler to schedule the tasks while associated servers are scheduled on the physical processor by a system or global scheduler. This approach ensures that any deviation of an application from its assumed values can be limited to the application itself.

2.1.2 Overload Management in Real-time Systems

For some applications with varying computational demand, it may not be possible to bound the demand with periodic or the sporadic model without over-provisioning of the resources. In such cases, it may be more apt to consider mode-based schedulability techniques [34, 35, 36]. For scenarios where such an approach cannot be adopted, for example, due to difficulty in defining distinct modes, or when schedulability cannot be guaranteed within a mode due to insufficient computational resources, it may be necessary to introduce some form of overload management techniques[14, 7], where the computational demand is adjusted at runtime to keep the application schedulable.

The Elastic Task Model

The elastic task model [14, 15] was proposed to manage overload scenarios by allowing the temporal parameters of each task to be defined through a range of values. In case of an overload situation (either predicted or detected), the associated elastic compression algorithms[16] adjust the computational demand of the tasks by modifying the utilization of the tasks such that they are as close as possible to their desired utilization while keeping the application schedulable. Algorithms for applying the elastic task model in multiprocessor systems were proposed in [37, 38, 39, 40].

Overload Management in Hierarchical Scheduling

In hierarchical scheduling, mode changes and overload situations are managed by modifying the resource reservations [36, 41] and ensuring schedulability even during the transition states [42]. In cases where there is sufficient spare capacity, the bandwidth modifications can be applied without affecting the performance of co-running applications or by considering their Quality-of-Service requirements[24]. An alternative mechanism to manage overload compared to bandwidth modifications is to adjust an application's tasks such that the computational demand can be satisfied within the existing reservation without affecting the performance of the co-running applications. Such an approach has been considered in [18]

where the system model considered allows only one task per reservation server. In this thesis, we extend this concept to a hierarchical scheduling framework for not only applications that require a reservation less than that of a single-core but also for those applications whose computational demand requires a multiprocessor reservation.

2.1.3 Software Migration and Multi-Core Platforms

The software for real-time systems undergoes continuous changes to meet the demands of different stakeholders [43]. Software migration refers to modifications of existing software, for example, when changing the hardware platform, or when adopting a different architectural paradigm than the existing one, such as changing the programming language [44] or when moving from native server deployments to cloud-based deployments [45, 46]. In many cases, it is expected that the functionality post migration shall be equivalent to that prior to migration. Moreover, the migration should consider architectural transformations with evolvability for long life-cycle systems as a criteria[47]. To manage such requirements, Sneed [48] proposed a five-step re-engineering planning process for legacy systems. The author highlights the need for creating measurable metrics to justify the effort and the improvements achievable with the migration. As architectural paradigms such as fog and edge computing have been proposed to provide benefits of cloud computing to applications with real-time requirements, the real-time software may need to run on shared multi-processor platforms supported by techniques such as hierarchical scheduling and other overload management techniques discussed previously. For many of the existing real-time systems, the software was developed to run with optimal performance on single-core hardware platforms. Therefore, for some applications, a complete redevelopment may be necessary to enable execution of the software on multi-processor platforms, while for some applications, reuse of the existing code along with architectural modifications may be sufficient. Since execution on multi-processors requires consideration of aspects such as cache contention, shared memory buses and scheduling algorithms that take into account the availability of multiple processors, a systematic approach is necessary to guide the migration of the software for execution on multi-core platforms.

2.1.4 Fog Computing for Real-time Systems

Fog computing is a distributed computing paradigm that complements the cloud platform by providing on-demand resources to applications that require low communication latencies [26]. The low latencies are achieved by introducing an intermediate layer of computational resources between the cloud and the edge

of the network of real-time systems. The intermediate layer of computational resources can be connected through a network such as TSN[27] and the computational hardware can be of any arbitrary capacity[28]. For real-time systems, it may be necessary to reserve resources within the fog computing hierarchy including those corresponding to communications [49]. Moreover, given its distributed nature, the fog architecture remains susceptible to faults[50]. Furthermore, it inherits similar challenges such as those for achieving predictable cloud computing and virtualization[51]. Therefore, given the complexity associated with fog computing solutions, a comparative analysis of existing system architectures of real-time systems with those based on fog-computing is necessary to evaluate the advantages that can be achieved through fog-computing architectures.

2.2 Related Work

2.2.1 Hierarchical Scheduling

In a hierarchical scheduling framework, the computational demand of an application is abstracted through a single interface that specifies the computation time to be reserved along with the period with which it should be provided[11]. The reserved computation time can be available to the application through different reservation servers such as the periodic server and deferrable server [32]. The mechanism for defining such an interface and the schedulability tests vary depending on the scheduling policies employed for both local as well as global schedulers.

For single processor systems, Davis and Burns[32] provided an exact schedulability test for a hierarchical system with fixed priority preemptive schedulers (FP) for both local as well as global scheduling. They evaluated the schedulability for the periodic, sporadic and the deferrable servers and showed that the periodic server provides better schedulability compared to the deferrable and the sporadic server models. Similarly, Zhang and Burns [33] provided schedulability analysis when the earliest deadline first scheduling policy (EDF) was used as the local scheduler while the global scheduling policy could either be FP or EDF. For compositional systems, Mok[52] proposed the regularity based resource supply model and provided schedulability conditions for applications with either FP or EDF as the local scheduling policy. Shin and Lee [53] proposed the periodic resource supply model to define guaranteed resource time to be provided to an application and provided schedulability conditions when using FP or EDF as the local scheduling policy. Easwaran et al. [54] generalized the periodic resource model and provide methods to generate the optimal bandwidth interfaces and improve resource usage. Dewan and Fisher[55, 56] proposed an algorithm to determine the optimal server

parameters for the periodic resource model. Kim et al. [57] provide a method to reduce the overhead associated with periodic resource model. Yang et al.[58] consider the scheduling of mixed criticality tasks within a periodic resource model with multiple estimates for the resource supply.

For multiprocessor reservations, Leontyev and Anderson proposed the minimum parallelism supply form to schedule soft real-time tasks with no utilization loss. Easwaran et al. [59] extend the periodic resource model through an additional parameter that specifies the maximum physical processors that can be used to provide the reserved computation time at any given time. They considered the schedulability of sporadic tasks within such a reservation using global EDF [9] scheduling policy. Furthermore, they provided a transformation to generate equivalent periodic tasks for the multiprocessor resource reservations (MPR) to schedule them at the system level. Burmaykov et al. [3] proposed a generalization of the MPR model to reduce the pessimism for bandwidth allocation and provided schedulability conditions for both global EDF as well as global FP scheduling policies. Yang and Anderson [60] provided conditions to preserve optimality of the minimum-parallelism supply form even for hard real-time tasks. Pathan et al. [61] propose an overhead aware interface generation method for multiprocessor reservations for global FP scheduling policy.

Khalilzad et al. [21, 22, 41] proposed a feedback based adaptive resource reservation scheme that adjusts the bandwidth of resource supply for variable tasks to minimise deadline misses. Groesbrink et al.[24] consider a similar approach to manage variable tasks where the bandwidth is modified such that each server gets a minimum guaranteed supply, while any left-over spare capacity is used to satisfy the requirements of applications whose bandwidth should be increased.

2.2.2 Software Migration

Many software processes have been proposed to address software evolution[47] which is concerned with modifications of the software to accommodate different requirements such as business needs, adoption of newer algorithms, change of hardware platforms. Menychtas et al. [62] presented a framework called *ARTIST*, a three-phase approach for software modernization focusing on migration towards the cloud. They categorised the migration into three main phases, *Pre-migration, Migration and Modernisation* and *Post-migration*. During the pre-migration phase, they proposed a feasibility study to address the technical and economic points of view. During the migration and modernisation phase, the actual migration is carried out and finally during the Post-migration phase, the system is deployed and validated. Erraguntla et al. [63] discussed a three phase migration method consisting of analysis, synthesis and transformation phases to migrate single-core

to multi-core parallel environments. During the analysis and synthesis phase, the design of the existing software is recovered while recommendations for the multi-core environment are made during the transformation phase of the migration method. They also provided a reverse engineering toolkit called *RETK* for the analysis and synthesis phases. Battaglia [64] presented the *RENAISSANCE* method for re-engineering a legacy system. The method focuses on planning and management of the evolution process. Forite et al. [65] proposed the *FASMM* approach to better manage the migration and to record and reuse the knowledge gained during the migration in other projects. Reussner et al. [66] and Wagner [67] proposed model-driven approaches to software migration. Her, the approach requires reverse engineering the system using automated tools and capturing the information in modelling languages and from there on, use model-driven techniques for further maintenance of the system. Most of these works provide a general approach to software migration. In this thesis, we tailor these approaches for migration of real-time systems to multi-core platforms.

2.2.3 Fog Computing for Industrial Systems

Authors of [68, 69] have discussed fog-based solutions for general robotic systems and highlighted the advantages of using fog-based architecture for such applications. While Hao et al. [70] provided a generic software architecture for fog computing, Faragardi et al. [71] provided a time predictable framework for a smart factory integrating the fog and cloud layers. Skarin et al. [72] developed a test bed to study the feasibility of a fog-based approach for control applications, while Pallasch et al. [73] and Mubeen et al. [74] showed the feasibility of using an edge based solution for combining cloud and field devices. Ning et al. [75] considered fog computing in the context of smart traffic management. Barzegaran et al.[76] provide an industrial use-case with electric drives as fog nodes.

Chapter 3

Research Contributions

This chapter introduces the goals of the thesis in Section. 3.1 and provides an overview of the research process in Section. 3.2. An overview of the thesis contributions and a mapping of the contributions of this thesis to the sub-goals is then discussed in Section. 3.3.

3.1 Research Goals

The overall goal of the thesis is to propose and evaluate solutions to integrate independent elastic real-time applications on fog computing platforms while satisfying their temporal requirements. To achieve this, we define the following sub-goals:

RG_1: Provide a solution to schedule elastic real-time applications on virtualized single-core and multi-core platforms.

The fog layer in the fog computing paradigm is expected to host a wide spectrum of independent applications including those with real-time requirements. As the resource demand of the applications can vary between those that need only limited computation time on a single core and those that require multiple processors for achieving their functional behaviour, we formulate the research goal RG_1 to address the scheduling of the applications on both single-core as well multi-core platforms.

RG_2: Propose solutions to guide the migration of existing real-time software applications from a single-core to multi-core platforms.

Since many of the existing real-time applications are optimized for single-core and networked system architectures, it may be necessary to transform their architectures for multi-core platforms before they can be deployed in a

fog computing architecture. To support such transformation, we formulate the research goal RG_2.

RG_3: Evaluate the advantages of a fog-based architecture for industrial robotic systems and identify research challenges.

Industrial robots are widely used in many different environments, especially in the automotive industry. The system architecture of some of the existing robot controllers follows a networked approach where the software is distributed over multiple single-core platforms. To investigate the advantages of fog computing architecture over such an existing architecture, we formulate the research goal RG_3. Furthermore, we extend the scope of this goal to identify research challenges associated with fog computing architectures to enable realization of such an architecture in practice.

3.2 Research Process

The thesis results were developed following the hypothetico-deductive method [77] and consisted of the following four steps.

- Problem Definition - Understand field of topic through literature review, systematic survey, state-of-art and state-of-practice study and defining the scope of the problem.
- Idea Development - Iterative development of solutions to the defined problem.
- Implementation - Converting the idea and the theory into an artefact.
- Evaluation - Evaluate the idea and its implementation and draw conclusions and identify its limitations.

Problem Definition In this thesis, we define the problems for our research to achieve the research goals described in Section. 3.1. The problems were based on literature review and inputs from industrial experts.

Idea Development and implementation The solutions to the identified problems were iteratively refined by evaluating the solutions using different research methods. A mapping of the different research methods used to achieve the research goals is provided in Table. 3.1.

Evaluation The evaluation of the solutions developed for each of the research goals was done through a comparative analysis of existing state-of-art solutions for RG 1, and state-of-practice for RG 3. The evaluation of the methodology developed for RG 2 was evaluated using a survey-based approach, which is described in detail in Paper C (Chapter. 7).

Research Methods	**Research Goals**
State-of-Art Study	RG1, RG 2, RG 3
State-of-Practice Study	RG 2, RG 3
Simulation	RG 1
Case study	RG 2, RG 3
Survey	RG 2

Table 3.1. Mapping of Research Methods and Research Goals

	RG_1	RG_2	RG_3
C_1	X		
C_2	X		
C_3		X	
C_4			X

Table 3.2. Mapping between the Contributions C_1 through C_4 and the research goals RG_1 through RG_3.

3.3 Technical Contributions

Here we outline the technical contributions of the thesis and then summarize each of the individual contributions. A mapping of the research goals with the technical contributions is shown in Table.3.2.

C 1: A reservation based scheduling framework for executing elastic real-time applications on a uniprocessor system.

C 2: A reservation based scheduling framework for executing elastic real-time applications on a multiprocessor system.

C 3: A systematic methodology to migrate from a single-core to a multi-core architecture with maximum software reuse and minimal re-engineering effort for real-time systems.

C 4: A fog-based architecture for industrial robotic systems and identification of research challenges.

C 1: A reservation based scheduling framework for executing elastic real-time applications on a uniprocessor system.

This contribution addresses the research goal *RG 1* and is provided in Paper A. Here, we consider a system model where an application is modeled according to the elastic task model[14] and is assumed to execute within a reservation server. The reservation server is assumed to be available to the application according to the periodic resource supply form (PRM) [11]. The application also includes a rate monotonic fixed priority scheduler or an Earliest-Deadline-First dynamic priority scheduler. The design parameters of the reservation server, i.e., the budget and the period of the server are determined by considering the initial desired utilization and periods of the application tasks. Whenever an application task requests a change in its period during runtime, the reserved server bandwidth will have to be updated. Such a bandwidth update can be performed without affecting the bandwidth of other reservations if there's sufficient spare capacity in the system. However, in the absence of sufficient spare capacity, there may be no option but to adjust the bandwidth of all the other co-running servers. To minimise such bandwidth modifications, a utilization modification algorithm, a variant of the original elastic compression algorithm[14], is integrated within the application to generate a new set of periods for the application's tasks while ensuring that they satisfy the constraints defined for each task. Since the resource supply is assumed to be provided according to the PRM form, there exist a possibility that in the worst-case, the resource is unavailable for the application for a specific duration. If any of the tasks have their periods that are less than this is unavailable duration, there may be no but to adapt the bandwidth of the server. However, if no tasks have their periods less than or equal to the unavailable duration, and the newly generated periods satisfy a utilization based schedulability test, the server bandwidth can remain unchanged, thereby limiting the need for server bandwidth adjustment. The evaluation in Paper A shows that the proposed approach works well for many tasksets, i.e., the utilization adjustment within the application can keep the tasks schedulable within the reservation, but for certain tasksets, the bandwidth readjustments may be unavoidable and the bandwidth will have to be readjusted for every period change request.

C 2: A reservation based scheduling framework for executing elastic real-time applications on a multiprocessor system.

This contribution addresses the research goal *RG 1* and is provided in Paper B. Here, we consider a system model where an application is modeled according to the elastic task model and is assumed to execute within a reservation server. The reservation server provides the resource supply to the application tasks according to the multiprocessor minimum-parallelism resource supply form[2, 60]. This reservation form provides the application with m fully dedicated processors and at most one partial processor. The partial processor is assumed to be made available according to the periodic resource supply form. Each reservation includes a local scheduling policy to schedule the application tasks. In paper B, the partitioned EDF scheduling policy is used for scheduling the application's tasks within the reservation. Similar to the uniprocessor scheduling framework discussed previously, the reservation parameters are decided based on the initial desired desired utilization and period of the application's tasks. Once the reservation parameters have been decided, the objective is to minimize the frequency of modifications of these parameters. To do so, whenever a task requests for a change in its period, the utilization modification algorithm considers only those tasks that share the processor with the task requesting the change and tries to generate new periods while satisfying the constraints of each of those tasks. If no solution is found, the algorithm re-partitions the tasks among different processors including the partial processor. The re-partitioning can be done according to any reasonable allocation scheme [78]. In paper B, the evaluation is done using the first-fit approach. only if the re-partitioning approach fails, the bandwidth of the reservation is considered for modification. The modification can either be in the form of modifying the partial processor bandwidth or the provisioning of a new fully dedicated processor if possible. The evaluation in paper B shows that the proposed approach can satisfy up to ninety four percent of the requests for period adaptations using the per-core utilization modification scheme and hundred percent of requests when combined with the re-partitioning step.

C 3: A software engineering methodology to migrate legacy real-time systems to multi-processor platforms

This contribution addresses research goal *RG 2* and constitutes the content of paper C. Many of the existing industrial real-time systems have their software developed for execution on single-core platforms and for realizing complex functionality they tend to rely on a network of multiple single-core devices. With the availability of modern multiprocessor platforms, the network based system architecture can

be replaced by an integrated approach where the functionality can be executed on the same hardware platform with reduced communication latencies. While some systems may benefit from a complete redevelopment process targeting the multiprocessor platforms, some others can benefit from reuse of already developed code rather than a complete redevelopment. Doing so successfully however requires a well defined methodology. In paper C, we propose systematic approach that defines a series of processes to be adopted to be able to reuse existing code on a multiprocessor platforms and review some of the available tools that can be utilized during this transition. The proposed approach is tailored for systems with real-time requirements and provides a three-stage workflow starting from architecture migration, followed by implementation level processes, and finally the validation of the transformed architecture. The validity of this approach is evaluated using a survey. The survey was designed according to the guidelines in [79] for survey-based research. The evaluation was defined to address the objectives of feasibility, usability and usefulness of the proposed methodology. The results of the survey indicated that although the proposed approach satisfied the evaluation objectives, it may not be generally applicable to all real-time systems and that the methodology should be adapted for individual cases.

C 4: A fog-based architecture for industrial robotic systems

This contribution addresses the research goal *RG 3* and is the main content of paper D. Fog computing has been proposed as an extension of the cloud computing paradigm that introduces an intermediate layer of computational resources between the edge of a network and the cloud resources to reduce the communication latencies between applications at the edge and the cloud, thus making it suitable for time sensitive real-time applications. While the benefits of fog computing architectures were evaluated in the context of non real-time systems, this contribution highlights some of the benefits that can be achieved by adopting the fog-based architecture for industrial robotic system. The benefits were illustrated through a comparative study that analysed an existing robotic system and identified some of its limitations. Based on the analysis, a fog-based architecture was proposed that addressed these limitations. The proposed fog architecture moved many of the existing system functionality from a dedicated hardware platform to computational resources in the fog layer. However, as the fog computing paradigm is relatively new, there remain considerable issues that need to be investigated for implementing such an architecture in practice. Therefore, a set of research challenges were identified relating to resource isolation and virtualization for integrating independent applications in the fog layer and orchestration of the real-time workload among the fog computing resources for better efficiency and performance.

3.4 Included papers

Paper A

Title: Scheduling Elastic Applications in Compositional Real-Time Systems [80]
Authors: S. M. Salman, S. Mubeen, F. Markovic, A. V. Papadopoulos, and T. Nolte
Status: published at ETFA 2021
Abstract: Many real-time applications have functional behaviour that requires variability in timing properties at run- time. The elastic task model provides a convenient mechanism to specify and encapsulate such variability and enables the modification of an application's periods during run-time to keep the application schedulable. Additionally, reservation-based scheduling techniques were proposed for the same purpose of taming unpredictability of timing variations, but with a different solution, i.e., by providing the spatial and temporal isolation for executing independent applications on the same hardware. In this paper, we combine the two approaches by proposing a two-level adaptive scheduling framework which is based on the elastic task model and the compositional framework based on the periodic resource model. The proposed framework minimises the number of requests for bandwidth adaption at the reservation (system) level and primarily enables schedulability by accounting for the application's elasticity by adjusting the periods. The motivation for this design choice is to rather localise the effect of the modifications within the application, without necessarily affecting all the applications at the system level compared to the changes made at the reservation level. The evaluation results show that the local application changes may often be enough to solve the problem of variability, significantly reducing the number of bandwidth adjustments, and therefore reducing the potential negative impact on all the applications of a system.

Paper contributions: An adaptive scheduling framework for elastic real-time applications in a compositional system for a single core processor.

My role: I was the main driver of the work under supervision of the co-authors.

Paper B

Title: Multi-processor scheduling of elastic applications in compositional real-time systems
Authors: S. M. Salman, A. V. Papadopoulos, S. Mubeen, and T. Nolte
Status: Published in Journal of Systems Architecture, (December 2021)

Abstract: Scheduling of real-time applications modelled according to the periodic and the sporadic task model under hierarchical and compositional real-time systems has been widely studied to provide temporal isolation among independent applications running on shared resources. However, for some real-time applications which are amenable to variation in their timing behaviour, usage of these tasks models can result in pessimistic solutions. The elastic task model addresses this pessimism by allowing the timing requirements of an application's tasks to be specified as a range of values instead of a single value. Although the scheduling of elastic applications on dedicated resources has received considerable attention, there is limited work on scheduling of such applications in hierarchical and compositional settings.

In this paper, we evaluate different earliest deadline first scheduling algorithms to schedule elastic applications in a minimum parallelism supply form reservation on a multiprocessor system. Our evaluation indicates that the proposed approach provides performance comparable to the current state-of-art algorithms for scheduling elastic applications on dedicated processors in terms of schedulability.

Paper contributions: A scheduling framework for elastic applications that require a multi-processor reservation.

My role: I am the main driver of the work under supervision of the co-authors.

Paper C

Title: A systematic methodology to migrate complex real-time software systems to multi-core platforms[81]
Authors: Salman, S. M., Papadopoulos, A. V., Mubeen, S., and Nolte, T.
Status: Published in Journal of Systems Architecture, (August 2021)
Abstract: This paper proposes a systematic three-stage methodology for migrating complex real-time industrial software systems from single-core to multi-core computing platforms. Single-core platforms have limited computational capabilities that prevent integration of computationally demanding applications such as image processing within the existing system. Modern multi-core processors offer a promising solution to address these limitations by providing increased computational power and allowing parallel execution of different applications within the system. However, the transition from traditional single-core to contemporary multi-core computing platforms is non-trivial and requires a systematic and well-defined migration process. This paper reviews some of the existing migration methods and provides a systematic multi-phase migration process with emphasis on software architecture recovery and transformation to explicitly address the timing and dependability attributes expected of industrial software systems. The

methodology was evaluated using a survey-based approach and the results indicate that the presented methodology is feasible, usable and useful for real-time industrial software systems.

Paper contributions: The main contribution of this work is a systematic migration methodology and its evaluation by industry experts.

My role: I was the main driver of the work under supervision of the co-authors.

Paper D

Title: Fogification of industrial robotic systems: Research challenges.[82]
Authors: S. M. Salman, V. Struhar, A. V Papadopoulos, M. Behnam, and T. Nolte
Status: Published at IoT-Fog 2019 - Proceedings of the 2019 Workshop on Fog Computing and the IoT.
Abstract: To meet the demands of future automation systems, the architec-ture of traditional control systems such as the industrial robotic systems needs to evolve and new architectural paradigms need to be investigated. While cloud-based platforms provide services such as computational resources on demand, they do not address the requirements of real-time performance expected by control applications. Fog computing is a promising new architectural paradigm that complements the cloud-based platform by addressing its limitations. In this paper, we analyse the existing robot system architecture and propose a fog-based solution for industrial robotic systems that addresses the needs of future automation systems. We also propose the use of Time-Sensitive Networking (TSN) services for real-time communication and OPC-UA for information modelling within this architecture. Additionally, we discuss the main research challenges associated with the proposed architecture.

Paper contributions: Proposed a Fog-based architecture for industrial robotic systems.

Chapter 4

Conclusion

In this thesis, we considered the problem of scheduling real-time applications with variable timing requirements on virtualized hardware platforms and proposed a scheduling framework that minimizes the frequency of bandwidth modifications at the system level on uniprocessor and multiprocessor platforms. Simultaneously, we considered the issue of reusing existing application code developed for single-core platforms in multi-core platforms and proposed a systematic methodology to guide the migration with emphasis on architecture recovery and its transformation. The methodology was validated using a survey-based approach for its feasibility, usefulness and usability. Furthermore, we considered the limitations of an existing system architecture of an industrial robot controller and proposed a fog-based architecture that addresses these limitations and identified some of the research challenges that should be addressed to implement such an architecture in practice.

For future work, we will investigate the worst case performance of the proposed scheduling framework and consider alternative reservation policies such the GMPR interface [3] and the applicability of the utilization modification algorithms not only at the application level but also at the system level to improve the resource utilization and quality-of-service of the applications.

Bibliography

[1] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *2008 Euromicro Conference on Real-Time Systems*, pages 181–190, 2008.

[2] Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Proceedings - Euromicro Conference on Real-Time Systems*, pages 191–200, 2008.

[3] Artem Burmyakov, Enrico Bini, and Eduardo Tovar. Compositional multiprocessor scheduling: the GMPR interface. *REAL-TIME SYSTEMS*, 50(3, SI):342–376, MAY 2014.

[4] Sebastian Altmeyer, Roeland Douma, Will Lunniss, and Robert I. Davis. On the effectiveness of cache partitioning in hard real-time systems. *REAL-TIME SYSTEMS*, 52(5, SI):598–643, SEP 2016. 26th Euromicro Conference on Real-Time Systems (ECRTS), Madrid, SPAIN, JUL 08-11, 2014.

[5] Sparsh Mittal. A survey of techniques for cache partitioning in multicore processors. *ACM Comput. Surv.*, 50(2), may 2017.

[6] Benny Akesson and Kees Goossens. *Memory Controllers for Real-Time Embedded Systems: Predictable and Composable Real-Time Systems*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[7] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems*. Springer, 2005.

[8] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings Real-Time Systems Symposium*, pages 308–319, 1997.

[9] Sanjoy Baruah, Marko Bertogna, and Giorgio C. Buttazzo. *Multiprocessor scheduling for real-time systems*. Embedded systems. Springer, Cham, 2015.

[10] Luca Abeni and Giorgio Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Syst.*, 27(2):123–167, jul 2004.

[11] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3):30:1–30:39, 2008.

[12] Miguel Alcon, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J Cazorla. Timing of autonomous driving software: Problem analysis and prospects for future solutions. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 267–280. IEEE, 2020.

[13] Martin Stigge and Wang Yi. Graph-based models for real-time workload: a survey. *Real-Time Systems*, 51(5):602–636, 2015.

[14] Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. In *Proceedings - Real-Time Systems Symposium*, pages 286–295. IEEE, 1998.

[15] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[16] Marion Sudvarg, Chris Gill, and Sanjoy Baruah. Linear-time admission control for elastic scheduling. *Real-Time Systems*, 57(4):485–490, 2021.

[17] L. Abeni and G. Buttazzo. Hierarchical qos management for time sensitive applications. In *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium*, pages 63–72, 2001.

[18] Luca Abeni, Tommaso Cucinotta, Giuseppe Lipari, Luca Marzario, and Luigi Palopoli. Qos management through adaptive reservations. *Real-Time Systems*, 29(2-3):131–155, 2005.

[19] Nima Moghaddami Khalilzad, Thomas Nolte, Moris Behnam, and Mikael Asberg. Towards adaptive hierarchical scheduling of real-time systems. In *ETFA*, pages 1–8. IEEE, 2011.

[20] Nima Moghaddami Khalilzad, Moris Behnam, Giacomo Spampinato, and Thomas Nolte. Bandwidth adaptation in hierarchical scheduling using fuzzy controllers. In *SIES*, pages 148–157. IEEE, 2012.

[21] Nima Moghaddami Khalilzad, Moris Behnam, and Thomas Nolte. Adaptive hierarchical scheduling framework: Configuration and evaluation. In *ETFA*, pages 1–10. IEEE, 2013.

[22] Nima Moghaddami Khalilzad, Moris Behnam, and Thomas Nolte. Multi-level adaptive hierarchical scheduling framework for composing real-time systems. In *RTCSA*, pages 320–329. IEEE Computer Society, 2013.

[23] Ricardo Marau, Karthik Lakshmanan, Paulo Pedreiras, Luís Almeida, and Raj Rajkumar. Efficient elastic resource management for dynamic embedded systems. *Proc. 10th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications, TrustCom 2011, 8th IEEE Int. Conf. on Embedded Software and Systems, ICESS 2011, 6th Int. Conf. on FCST 2011*, pages 981–990, 2011.

[24] Stefan Groesbrink, Luis Almeida, Mario De Sousa, and Stefan M. Petters. Towards certifiable adaptive reservations for hypervisor-based virtualization. *Real-Time Technology and Applications - Proceedings*, 2014-October(October):13–24, 2014.

[25] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. *Fog Computing: A Taxonomy, Survey and Future Directions*, pages 103–130. Springer Singapore, Singapore, 2018.

[26] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[27] Paul Pop, Michael Lander Raagaard, Marina Gutierrez, and Wilfried Steiner. Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking (TSN). *IEEE Communications Standards Magazine*, 2(2):55–61, 2018.

[28] Paul Pop, Bahram Zarrin, Mohammadreza Barzegaran, Stefan Schulte, Sasikumar Punnekkat, Jan Ruh, and Wilfried Steiner. The FORA fog computing platform for industrial iot. *Inf. Syst.*, 98:101727, 2021.

[29] D. Simon, D. Robert, and O. Sename. Robust control/scheduling co-design: application to robot control. In *11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 118–127, 2005.

[30] Ala' Qadi, Steve Goddard, Jiangyang Huang, and Shane Farritor. Modelling computational requirements of mobile robotic systems using zones and processing windows. *Real-Time Systems*, 42(1):1–33, 2009.

[31] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 4–13, 1998.

[32] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. *Proceedings - Real-Time Systems Symposium*, 2005.

[33] Fengxiang Zhang and Alan Burns. Analysis of hierarchical edf pre-emptive scheduling. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 423–434, 2007.

[34] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No.98EX168)*, pages 172–179, 1998.

[35] Jorge Real and Alfons Crespo. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[36] Luca Santinelli, Giorgio C. Buttazzo, and Enrico Bini. Multi-moded resource reservations. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 37–46. IEEE Computer Society, 2011.

[37] James Orr, Chris Gill, Kunal Agrawal, Jing Li, and Sanjoy Baruah. Elastic Scheduling for Parallel Real-Time Systems. *ACM Subject Classification*, 6(1):5:1–5:14, 2019.

[38] James Orr and Sanjoy Baruah. Algorithms for implementing elastic tasks on multiprocessor platforms: a comparative evaluation. *Real-Time Systems*, 57(1):227–264, 2021.

[39] James Orr, Christopher D. Gill, Kunal Agrawal, Sanjoy K. Baruah, Christian Cianfarani, Phyllis Ang, and Christopher Wong. Elasticity of workloads and periods of parallel real-time tasks. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10-12, 2018*, pages 61–71. ACM, 2018.

[40] James Orr, Johnny Condori Uribe, Christopher D. Gill, Sanjoy K. Baruah, Kunal Agrawal, Shirley Dyke, Arun Prakash, Iain Bate, Christopher Wong,

and Sabina Adhikari. Elastic scheduling of parallel real-time tasks with discrete utilizations. In Liliana Cucu-Grosjean, Roberto Medina, Sebastian Altmeyer, and Jean-Luc Scharbarg, editors, *28th International Conference on Real Time Networks and Systems, RTNS 2020, Paris, France, June 10, 2020*, pages 117–127. ACM, 2020.

[41] Nima Khalilzad, Fanxin Kong, Xue Liu, Moris Behnam, and Thomas Nolte. A feedback scheduling framework for component-based soft real-time systems. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 182–193, 2015.

[42] Wei-Ju Chen, Peng Wu, Pei-Chi Huang, Aloysius K Mok, and Song Han. Online reconfiguration of regularity-based resource partitions in cyber-physical systems. *Real-Time Systems*, pages 1–44, 2021.

[43] Hongyu Pei Breivold, Ivica Crnkovic, and Magnus Larsson. Software architecture evolution through evolvability analysis. *Journal of Systems and Software*, 85(11):2574–2592, 2012.

[44] Leszek Wlodarski, Boris Pereira, Ivan Povazan, Johan Fabry, and Vadim Zaytsev. Qualify First! A Large Scale Modernisation Report. In *SANER*, pages 569–573. IEEE, 2019.

[45] Philip Church, Harald Mueller, Caspar Ryan, Spyridon V. Gogouvitis, Andrzej Goscinski, and Zahir Tari. Migration of a SCADA system to IaaS clouds – a case study. *Journal of Cloud Computing*, 6(1):256, 2017.

[46] Konstantinos Plakidas, Daniel Schall, and Uwe Zdun. Software Migration and Architecture Evolution with Industrial Platforms: A Multi-case Study. In Carlos E. Cuesta, David Garlan, and Jennifer Pérez, editors, *Software Architecture*, volume 11048 of *Lecture Notes in Computer Science*, pages 336–343. Springer International Publishing, Cham, 2018.

[47] Hongyu Pei-Breivold. *Software Architecture Evolution through Evolvability Analysis*. PhD thesis, Mälardalen University, November 2011.

[48] H. M. Sneed. Planning the reengineering of legacy systems. *IEEE Software*, 12(1):24–34, 1995.

[49] Mohammadreza Barzegaran, Anton Cervin, and Paul Pop. Performance optimization of control applications on fog computing platforms using scheduling and isolation. *IEEE Access*, 8:104085–104098, 2020.

[50] Zeinab Bakhshi, Guillermo Rodriguez-Navas, and Hans Hansson. Fault-tolerant permanent storage for container-based fog architectures. In *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, volume 1, pages 722–729, 2021.

[51] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726–740, 2014.

[52] Aloysius K Mok and Xiang Alex. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*, pages 129–138. IEEE, 2001.

[53] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2–13. IEEE Computer Society, 2003.

[54] Arvind Easwaran, Insup Lee, Insik Shin, and Oleg Sokolsky. Compositional schedulability analysis of hierarchical real-time systems. In *ISORC*, pages 274–281. IEEE Computer Society, 2007.

[55] Nathan Fisher and Farhana Dewan. A bandwidth allocation scheme for compositional real-time systems with periodic resources. *Real Time Syst.*, 48(3):223–263, 2012.

[56] Farhana Dewan and Nathan Fisher. Bandwidth allocation for fixed-priority-scheduled compositional real-time systems. *ACM Trans. Embed. Comput. Syst.*, 13(4):91:1–91:29, 2014.

[57] Jin Hyun Kim, Kyong Hoon Kim, Arvind Easwaran, and Insup Lee. Towards overhead-free interface theory for compositional hierarchical real-time systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 37(11):2869–2880, 2018.

[58] Kecheng Yang and Zheng Dong. Mixed-criticality scheduling in compositional real-time systems with multiple budget estimates. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 25–37. IEEE, 2020.

[59] Arvind Easwaran, Insup Lee, Oleg Sokolsky, and Steve Vestal. A compositional scheduling framework for digital avionics systems. In *RTCSA*, pages 371–380. IEEE Computer Society, 2009.

[60] Kecheng Yang and James H. Anderson. On the Dominance of Minimum-Parallelism Multiprocessor Supply. *Proceedings - Real-Time Systems Symposium*, 0:215–226, 2016.

[61] Risat Mahmud Pathan, Per Stenström, Lars Göran Green, Torbjörn Hult, and Patrik Sandin. Overhead-aware temporal partitioning on multicore processors. *Real-Time Technology and Applications - Proceedings*, 2014-Octob(October):251–262, 2014.

[62] Andreas Menychtas, Kleopatra Konstanteli, Juncal Alonso, Leire Orue-Echevarria, Jesus Gorronogoitia, George Kousiouris, Christina Santzaridou, Hugo Bruneliere, Bram Pellens, Peter Stuer, Oliver Strauss, Tatiana Senkova, and Theodora Varvarigou. Software modernization and cloudification using the ARTIST migration methodology and framework. *Scalable Computing: Practice and Experience*, 15(2), 2014.

[63] Ravi Erraguntla and Doris L. Carver. Migration of sequential systems to parallel environments by reverse engineering. *Information & Software Technology*, 40(7):369–380, 1998.

[64] M. Battaglia, G. Savoia, and J. Favaro. Renaissance: a method to migrate from legacy to immortal software systems. In *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, pages 197–200, 1998.

[65] Louis Forite and Charlotte Hug. FASMM: Fast and Accessible Software Migration Method. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science*, pages 1–12. IEEE, 2014.

[66] Ralf Reussner, Michael Goedicke Wilhelm Hasselbring, Birgit Vogel-Heuser, Jan Keim, Lukas Märtin, editor. *Managed Software Evolution*. Springer Nature Switzerland AG, 2019.

[67] Christian Wagner. *Model-Driven Software Migration: A Methodology*. Springer Fachmedien Wiesbaden, Wiesbaden, 2014.

[68] Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Fog-Enabled Multi-Robot Systems. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2018.

[69] Siva Leela Krishna Chand Gudi, Suman Ojha, Benjamin Johnston, Jesse Clark, and Mary-Anne Williams. Fog Robotics for Efficient, Fluent and Robust Human-Robot Interaction. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–5. IEEE, 2018.

[70] Zijiang Hao, Ed Novak, Shanhe Yi, and Qun Li. Challenges and Software Architecture for Fog Computing. *IEEE Internet Computing*, 21(2):44–53, 2017.

[71] Hamid Reza Faragardi, Saeid Dehnavi, Mehdi Kargahi, Alessandro V. Papadopoulos, and Thomas Nolte. A time-predictable fog-integrated cloud framework: One step forward in the deployment of a smart factory. In *2018 Real-Time and Embedded Systems and Technologies (RTEST)*, pages 54–62. IEEE, 2018.

[72] Per Skarin, William Tarneberg, Karl-Erik Å rzén, and Maria Kihl. Towards Mission-Critical Control at the Edge and Over 5G. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 50–57. IEEE, 2018.

[73] Christoph Pallasch, Stephan Wein, Nicolai Hoffmann, Markus Obdenbusch, Tilman Buchner, Josef Waltl, and Christian Brecher. Edge Powered Industrial Control: Concept for Combining Cloud and Automation Technologies. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 130–134. IEEE, 2018.

[74] S. Mubeen, P. Nikolaidis, A. Didic, H. Pei-Breivold, K. Sandström, and M. Behnam. Delay mitigation in offloaded cloud controllers in industrial iot. *IEEE Access*, pages 4418–4430, 2017.

[75] Zhaolong Ning, Jun Huang, and Xiaojie Wang. Vehicular fog computing: Enabling real-time traffic management for smart cities. *IEEE Wireless Communications*, 26(1):87–93, 2019.

[76] Mohammadreza Barzegaran, Nitin Desai, Jia Qian, Koen Tange, Bahram Zarrin, Paul Pop, and Juha Kuusela. Fogification of electric drives: An industrial use case. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 77–84, 2020.

[77] Gordana Dodig-crnkovic. Scientific methods in computer science. In *In Proc. PROMOTE IT 2002, 2nd Conference for the Promotion of Research in IT at New Universities and at University Colleges in*, pages 22–24, 2002.

[78] José María López, José Luis Díaz, and Daniel F. García. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real Time Syst.*, 28(1):39–68, 2004.

[79] Barbara A Kitchenham and Shari L Pfleeger. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London, London, 2008.

[80] Shaik Mohammed Salman, Saad Mubeen, Filip Marković, Alessandro V Papadopoulos, and Thomas Nolte. Scheduling elastic applications in compositional real-time systems. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2021.

[81] Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte. A systematic methodology to migrate complex real-time software systems to multi-core platforms. *Journal of Systems Architecture*, 117(March):102087, 2021.

[82] Shaik Mohammed Salman, Vaclav Struhar, Alessandro V. Papadopoulos, Moris Behnam, and Thomas Nolte. Fogification of industrial robotic systems: Research challenges. *IoT-Fog 2019 - Proceedings of the 2019 Workshop on Fog Computing and the IoT*, pages 41–45, 2019.

II

Included Papers

Chapter 5

Paper A: Scheduling Elastic Applications in Compositional Real-Time Systems

Shaik Mohammed Salman, Filip Markovic, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte.
In the proceedings of the IEEE 26th International Conference on Emerging Technologies and Factory Automation (ETFA 2021)

Abstract

Many real-time applications have functional behaviour that requires variability in timing properties at runtime. The elastic task model provides a convenient mechanism to specify and encapsulate such variability and enables the modification of an application's periods during run-time to keep the application schedulable. Additionally, reservation-based scheduling techniques were proposed for the same purpose of taming unpredictability of timing variations, but with a different solution, i.e., by providing the spatial and temporal isolation for executing independent applications on the same hardware.

In this paper, we combine the two approaches by proposing a two-level adaptive scheduling framework which is based on the elastic task model and the compositional framework based on the periodic resource model. The proposed framework minimises the number of requests for bandwidth adaption at the reservation (system) level and primarily enables schedulability by accounting for the application's elasticity by adjusting the periods. The motivation for this design choice is to rather localise the effect of the modifications within the application, without necessarily affecting all the applications at the system level compared to the changes made at the application level. The evaluation results show that the local application changes may often be enough to solve the problem of variability, significantly reducing the number of bandwidth adjustments, and therefore reducing the potential negative impact on all the applications of a system.

5.1 Introduction

Many industrial real-time systems such as robot controllers have real-time requirements that are flexible to variability in execution times of the tasks, and the frequency of the task invocations[1]. For instance, Simon et al. [2] provided a feedback-based scheduling algorithm for computed torque control of an industrial arm, where the frequency of the dynamic compensation tasks, such as that of gravity and Coriolis compensation, was regularly adapted to meet both the control objectives as well as the schedulability of the system tasks. Buttazzo et al. [3] proposed the elastic task model to capture such dynamic behaviour of the tasks where the schedulability of the system is managed by adapting the frequencies of the tasks. Recently, modern system architectures based on fog and cloud computing concepts have been proposed to improve the performance of robots [4, 5]. A key idea behind such architectures is to exploit the improved computation capabilities offered by the processors by executing independent applications on the same hardware, e.g., running multiple instances of the robot controller software to control different robots on the same processor. Since the execution of independent applications requires temporal and spatial isolation, the concepts of virtualization and hierarchical scheduling, based on reservations, provide the necessary infrastructure to enable such a requirement. While there exist many solutions to schedule adaptive tasks of independent applications in a hierarchical scheduling approach [6, 7, 8, 9], most of them focus on modifying the reservation parameters according to the application demands, rather than adapting the application behaviour to a fixed reservation bandwidth. A disadvantage of modifying the reservation parameters according to the application demands is that the performance of another independent application co-executing on the same processor may be unnecessarily affected. By making the applications adapt to a fixed reservation bandwidth, we can limit the impact on other applications running on the same processor. However, there may be instances where the local adaptation of the application can fail, e.g., due to insufficient bandwidth, compelling a bandwidth modification. Therefore, to meet the aforementioned requirements, we propose a two-layered adaptive approach to schedule applications specified according to the elastic task model within the compositional real-time framework based on the periodic resource model [10]. Concretely, we address the following questions:

Q1 Given an application with elastic tasks, what is a feasible bandwidth reservation according to the periodic resource model?

Q2 Given a fixed bandwidth reservation according to the periodic resource model, how can the elastic application adapt its frequencies to remain schedu-

lable?

Q3 Given an elastic application, can a schedulable reservation be found if the application requests for a modified bandwidth reservation?

We address Q1 by assuming that an application specifies initial desired frequencies for each of its tasks and then uses those values to identify a feasible bandwidth. We address Q2 by modifying the application task frequencies whenever there is an overload or an application's task requests a different frequency such that the application satisfies the schedulability conditions under the periodic resource model. We address Q3 by checking if the system-level schedulability is satisfied for the modified bandwidth reservation.

We provide the system model and discuss the necessary background on elastic tasks and the periodic resource model in Section 5.2, followed by the proposed solution in Section 5.3. We present the evaluation of our approach in Section 5.4 and the related work in Section 5.5. Finally, Section 5.6 concludes the paper.

5.2 Proposed System Model

This section presents the system model of the two-level compositional scheduling framework for uniprocessor systems. At the application level, we consider a real-time application specified according to the elastic task model with implicit deadlines(See Section. 5.2.1). We assume that each application provides a local scheduler, based on either fixed-priority preemptive scheduling implementing Rate Monotonic (RM) priority assignment or dynamic-priority preemptive scheduling implementing the Earliest Deadline First (EDF) policy. At the system level, we assume that the CPU resource is made available to each application according to the Periodic Resource Model (PRM) [10](See Section. 5.2.2).

5.2.1 The Basic Elastic Task Model

Buttazzo et al. [3, 11] proposed the elastic task model for applications whose tasks can have an adaptive temporal behaviour to address overload situations as well as requests for starting new tasks or modifying the task periods. Under this model, whenever there is an overload or a task requests a new period, the utilization of the remaining tasks is adjusted to keep the overall application's utilization under an upper-bound value for a given scheduling algorithm. For example, if the application tasks are scheduled according to EDF, then the application utilization bound is set to 1 and the utilization values of the individual tasks are adjusted accordingly. While the elastic task model can be applied to applications that have

computation time variability as well as period variability, in this paper, we will only consider applications with period variability.

Formally, we define an elastic application **A** as a set of n elastic tasks $\tau_i = \{C_i, T_i^{min}, T_i^{max}, T_i^d, e_i\}$, where C_i is the Worst-Case Execution Time (WCET) of the task τ_i. T_i^{min} and T_i^{max} specify the minimum and the maximum inter-arrival time between consecutive jobs of τ_i. T_i^d represents the desired period of τ_i. The elastic co-efficient e_i represents the flexibility of τ_i to change. For instance, e_i can be defined to be in the range $[0, 1]$, where $e_i = 0$ indicates that the $T_i^{min} = T_i^d = T_i^{max}$ and that this task's period cannot be modified, and $e_i = 1$ indicates that the task's period can be modified to take up values upto its maximum period. We use T_i (without any postscript) to indicate the current inter-arrival time of τ_i. An example of an elastic taskset is shown in the Table 5.1. while the task τ_1 can execute at its maximum period, the task τ_5 can only execute at its desired period.

Table 5.1. An elastic Task set

Task ID	**WCET**	T_i^{min}	T_i^d	T_i^{max}	e_i
τ_1	4	40	120	240	1
τ_2	7	40	80	360	0.75
τ_3	10	240	240	480	0.5
τ_4	9	200	240	600	0.25
τ_5	8	40	40	40	0

The utilization of a task τ_i is given by $U_i = \frac{C_i}{T_i}$. Further, the minimum and maximum utilization of each task is represented by $U_i^{min} = \frac{C_i}{T_i^{max}}$ and $U_i^{max} = \frac{C_i}{T_i^{min}}$ respectively. At run-time, the utilization of a task is kept as close as possible to a desired utilization $U_i^d = \frac{C_i}{T_i^d}$. The desired application utilization is given by $U^d = \sum_{i=1}^{n} U_i^d$. Similarly, the minimum and maximum application utilization is given by $U^{min} = \sum_{i=1}^{n} U_i^{min}$ and $U^{max} = \sum_{i=1}^{n} U_i^{max}$. If a task requests for a change in its current period, its desired utilization U_i^d is updated. An elastic application is said to be schedulable if $U^d \leq U^{ub}$, where U^{ub} is the utilization upper-bound for a given scheduling algorithm. It is assumed that the deadline is elastic-implicit. i.e., the relative deadline of each job of an elastic task is equal to its current period at runtime.

Elastic Compression Algorithm At runtime, if a task exceeds its execution time or requests for a change in its period, the application is made schedulable by modifying the periods of the application's tasks to accommodate the new

values and ensuring that the total utilization of the application's tasks is below the schedulable utilization bound. This is done according to the original task compression algorithm proposed by Buttazzo et al. [3] and is reproduced here as Algorithm 1. It takes as input the elastic application and the maximum schedulable utilization bound. It computes the minimum utilization of the taskset and compares it to the schedulable utilization bound. If the minimum utilization of the elastic application exceeds the schedulable utilization bound, it immediately exits and returns a failure. Otherwise, it iterates through each task of the application and depending on the elastic coefficients and the current period T_i of each task, it separates the tasks into two disjoint sets $\mathbf{A_f}$ and $\mathbf{A_v}$. The set $\mathbf{A_f}$ contains all the tasks whose utilization values are fixed, i.e., the tasks with elastic coefficients set to 0 and tasks executing with their maximum period values. The set $\mathbf{A_v}$ contains the remaining tasks whose utilization can be varied. Further, U_f represents the sum of the utilization of the tasks in $\mathbf{A_f}$, while E_v represents the sum of the elastic coefficients of tasks in $\mathbf{A_v}$. For each task in $\mathbf{A_v}$, its utilization value is scaled according to the ratio of the elastic coefficient and the sum of all the coefficients in $\mathbf{A_v}$ (Line 21). A new task period T_i is then assigned to the task. If the new task period exceeds the T_i^{max} value, it is set equal to T_i^{max}. If this happens, the task τ_i is added to the set $\mathbf{A_f}$ and the process is repeated. The algorithm returns a feasible taskset if either all the tasks have reached their maximum period or if all the tasks' periods have been updated such that they are schedulable. We use this algorithm as a part of our proposed solution (Section 5.3).

5.2.2 Periodic Resource Model

Lee et al. [10] proposed the compositional scheduling framework based on the periodic resource model to support the development of component-based hierarchical software systems. In this framework, the computational resource is described as a periodic resource model $\Gamma(\Theta, \Pi)$, where Θ is the periodic resource allocation time and Π is the resource period. Essentially, the periodic resource Γ provides an application $\mathbf{A}$ with Θ time units of CPU time every Π time units. The worst case resource supply of the periodic resource model is shown in Fig. 1. The utilization of the resource supply is defined as $U_\Gamma = \frac{\Theta}{\Pi}$.

Generating the Resource Supply Parameters we use the method described in Section. 6 of [10] to generate the resource supply parameters Θ and Π, such that an application $\mathbf{A}$, modeled as a set of n periodic tasks with implicit deadlines, where each task modeled as $\tau_i = \{C_i, T_i\}$ is schedulable (under EDF or RM scheduling policy). According to the compositional framework, Given the smallest period of

Algorithm 1 Task_Compress()

1: **function** TASK_COMPRESS($\mathbf{A}, U^{ub}$)
2: $U^d = \sum_{i=1}^{n} \frac{C_i}{T_i^d}$
3: $U^{min} = \sum_{i=1}^{n} \frac{C_i}{T_i^{max}}$
4: **if** $U^{ub} < U^{min}$ **then**
5: **return** Infeasible
6: **end if**
7: OK = 0
8: **while** OK= 0 **do**
9: $U_f = 0$
10: $E_v = 0$
11: **for** each τ_i in **A do**
12: **if** $e_i == 0$ **or** $T_i == T_i^{max}$ **then**
13: $U_f = U_f + U_i$
14: **else**
15: $E_v = E_v + e_i$
16: **end if**
17: **end for**
18: OK = 1
19: **for** each τ_i in $\mathbf{A_v}$ **do**
20: **if** $E_i > 0$ **and** $T_i < T_i^{max}$ **then**
21: $U_i = U_i^d - (U^d - U^{ub} + U_f) * \frac{e_i}{E_v}$
22: $T_i = \frac{C_i}{U_i}$
23: **if** $T_i > T_i^{max}$ **then**
24: $T_i == T_i^{max}$
25: OK = 0
26: **end if**
27: **end if**
28: **end for**
29: **end while**
30: **return** Feasible
31: **end function**

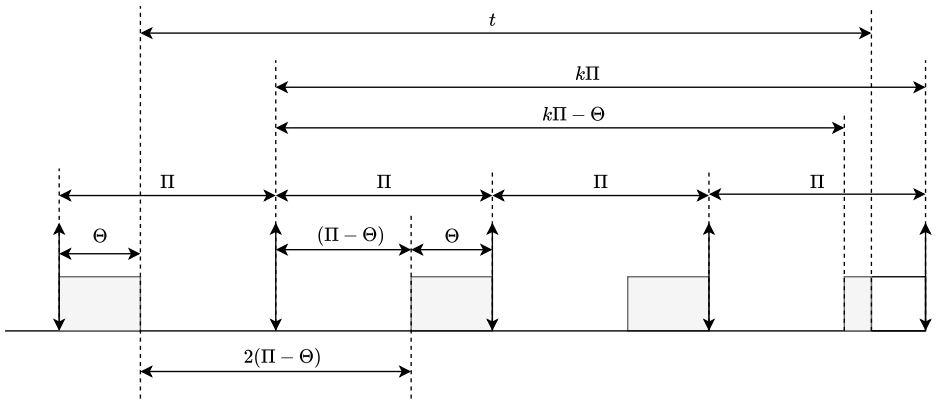

Figure 5.1. Resource supply of a Periodic Resource Model.

the application, T^{min}, the application is schedulable under RM scheduling policy if the resource supply utilization satisfies Eq. (5.1) (Eq. 23 in [10]).

$$U_{\Gamma,RM}(k) = \frac{U_{\mathbf{A}}}{\log\left(\frac{2k+2(1-U_{\mathbf{A}})}{k+2(1-U_{\mathbf{A}}}\right)}, \tag{5.1}$$

where,

$$k = max\{k \in \mathbb{Z} | (k+1)\Pi - \Theta < T^{min}\} \tag{5.2}$$

Similarly, the application is schedulable under EDF scheduling policy if the resource supply utilization satisfies Eq. (5.3) (Eq. 21 in [10]).

$$U_{\Gamma,EDF}(k) = \frac{(k+2).U_{\mathbf{A}}}{k+2(U_{\mathbf{A}})}, \tag{5.3}$$

where,

$$k = max\{k \in \mathbb{Z} | (k+1)\Pi - \Theta - \frac{k\Theta}{k+2} < T^{min}\} \tag{5.4}$$

Schedulable Utilization Bounds Since the main goal of the our solution is to minimize the modifications of the resource supply parameters once they have been defined, we rely on the schedulable utilization bounds defined in the Section 5 of [10] to keep the application schedulable by changing the application utilization rather than changing the resource supply utilization. Accordingly, an application is schedulable under RM policy, if the application utilization $U_{\mathbf{A}}$ is less than or equal to the utilization bound $\mathbf{UB}_{RM}(n, T^{min})$ as defined in Eq. (5.5) (Eq. 16 in [10]).

$$\mathbf{UB}_{RM}(n, T^{min}) = U_\Gamma.n\left[\left(\frac{2k + 2(1 - U_\Gamma)}{k + 2(1 - U_\Gamma)}\right)^{\frac{1}{n}} - 1\right], \tag{5.5}$$

where

$$k = max\{k \in \mathbb{Z} | (k + 1)\Pi - \Theta < T^{min}\}$$

Similarly, an application is schedulable under EDF policy, if the application utilization $U_\mathbf{A}$ is less than or equal to the utilization bound $\mathbf{UB}_{EDF}(T^{min})$ as defined in Eq. (5.6)(Eq. 13 in [10]).

$$\mathbf{UB}_{EDF}(T^{min}) = \frac{kU_\Gamma}{k + 2(1 - U_\Gamma)}, \tag{5.6}$$

where

$$k = max\{k \in \mathbb{Z} | (k + 1)\Pi - \Theta - \frac{k\Theta}{k + 2} < T^{min}\}$$

5.3 Proposed Solution

To schedule an elastic application in a hierarchical scheduling framework based on the periodic resource model, we propose a two-layered adaptive scheduling mechanism that first attempts to adapt the utilization of the tasks at the application level and if this adaptation fails, it attempts to reallocate available spare resource capacity at the system level. The different components of the proposed framework along with the data flow between them are shown in Fig. 5.2. At the application level, it consists of an independent application defined according to the elastic task model, an elastic manager that implements the task compression algorithm of Buttazzo et al. [3]. and a local scheduler implementing either the RM scheduling policy or the EDF scheduling policy. At the system level, the Global Compositional Scheduling Resource (GCSR) manager provides the necessary interface for communicating with the different applications and the functional support for serving requests of new bandwidth resource allocations from the individual applications. The functional behaviour of the GCSR manager is supported by the OS or the hypervisor kernel. At the application level, whenever there is an overload situation or an elastic task requests for a new period, the elastic manager will try to modify and update the periods of the rest of the tasks to keep the application schedulable using the Algorithm 1. If the resource supply is insufficient for the current demand, the elastic manager generates a new sufficient resource supply interface and requests the GCSR manager for updating the resource supply parameters. The GCSR manager will accept the request and responds successfully(i.e., assign new

resource supply parameters) if the global system schedulability is preserved with the updated parameters. The elastic manager will then re-adjust the periods based on the updated resource supply parameters.

5.3.1 Initial Desired Resource Supply

In the proposed framework, we first find the suitable resource supply $\Gamma(\Theta, \Pi)$ for the application $\mathbf{A}$ considering the parameters $\tau_i(C_i, T_i)$. We choose as T_i, the desired periods for each task. For example, in Table 3.1, The values under the column T_i^d represent the initial desired periods of the application tasks. We assume that such a taskset is feasible. Next, depending on the scheduling algorithm, we find the resource supply utilization bound necessary to keep the application tasks schedulable according to Eq. (5.1) and Eq. (5.3). While there exist fully polynomial time solutions to find approximate bandwidth allocations for the periodic resource model, e.g., [12], we use the approach proposed by Lee et al. [10] in this paper. We assume that $\Gamma(\Theta, \Pi)$ is schedulable at the system level. Note that if $\Gamma(\Theta, \Pi)$ is not schedulable at the system level, then the application will have to modify its initial desired periods or the resource supply of the other co-running applications will have to be modified. In our approach, we reject an application if the initial resource supply is not schedulable.

5.3.2 Runtime Adaptation

Under worst-case conditions, the resource supply provided according to PRM can result in a no supply interval of duration $2(\Pi - \Theta)$ (see Fig. 1). Therefore, once the application is executing, whenever a task requests for a new period T_i^{new}, we need to consider two different scenarios depending on the value of T_i^{new}. If T_i^{new} is greater than the no supply duration, we can adapt the tasks utilization at the application level without changing the resource supply parameters. If T_i^{new} is less than or equal to the no supply duration, we need to adapt the resource supply at the system level.

Application level Adaptation From Eq. (5.5) and Eq. (5.6), it is easy to see that the utilization bound to keep the application tasks schedulable remains constant as long as T^{min} remains unchanged. When the requested period T_i^{new} is greater than or equal to T^{min}, it implies that the resource supply parameters do not have to be changed since the current T^{min} remains unchanged. As a consequence, and based on the sustainability property of the utilization tests [13], the elastic manager can find a schedulable period reassignment by ensuring that the modified application utilization $U_{\mathbf{A}}^{new}$ remains below the schedulable utilization bound as

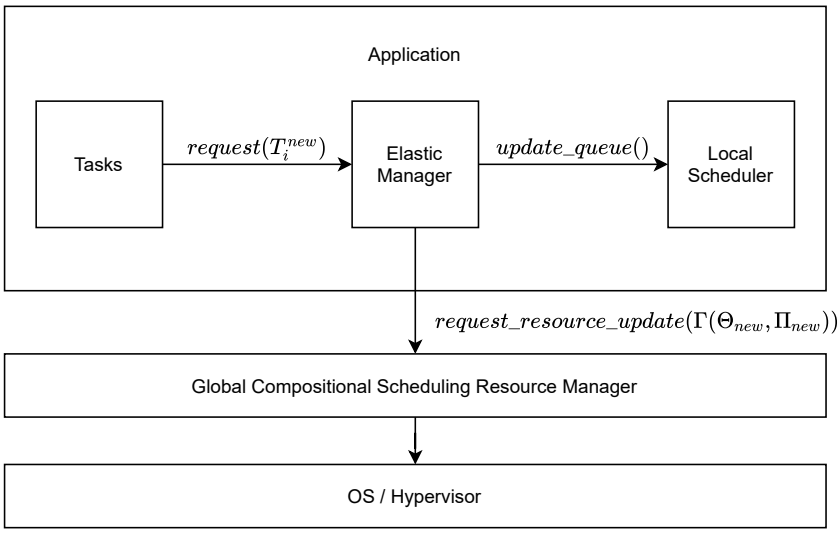

Figure 5.2. Data Flow Between The Adaptive Scheduling Framework Components.

shown in Eq. (5.7) or Eq. (5.8). Note that the periodic supply resource utilization U_Γ remains constant under application level adaptation.

$$U_{\mathbf{A}}^{new} \leq \mathbf{UB}_{EDF}(T^{min}) \tag{5.7}$$

$$U_{\mathbf{A}}^{new} \leq \mathbf{UB}_{RM}(n, T^{min}) \tag{5.8}$$

System Level Adaptation At runtime, if a task requests for a new period T_i^{new} that is less than T^{min}, then it is not guaranteed that the existing resource supply Γ can provide sufficient CPU time for the application tasks to remain schedulable. This is because the schedulable utilization bound (for both EDF and RM scheduling policy) is a function of the minimum period of the taskset and since we are now reducing the minimum period, it may so happen that the T_i^{new} will have its arrival and deadline in the no supply interval of duration $2(\Pi - \Theta)$ in the worst case (see Fig. 1). Therefore, whenever a task makes a request of T_i^{new} less than T^{min}, the elastic manager will generate new resource supply parameters and request the GCSR manager to update the resource supply according to the new parameters so that the application remains schedulable. If the GCSR manager rejects the request, then the elastic manager will not be able to satisfy the application request and it is then up to the application to decide how it needs to proceed. If the resource

supply parameters are updated, the next request for a period change will be handled based on the updated resource supply parameters. While it is possible to apply the elastic task compression algorithm at the system level to modify the resource supply utilization to accommodate the requests from the different applications, it requires modifications of the resource supply of the co-running applications which in turn can trigger application level modifications. To avoid this, we require some spare capacity to be made available at the system level so that it can be distributed among the different applications whenever required. Although we do not propose any particular method in this paper for the distribution of the spare capacity, the methods in [14, 15] are particularly well suited for the spare capacity distribution.

PRM Elastic Scheduler We now discuss how the elastic manager and the GCSR manager work together to adapt the application as well as system resources to maintain schedulability. The pseudo-code is presented in Algorithm 2. The functional behaviour is split between the elastic manager and the GCSR manager. The Elastic manager takes as input the request for T_i^{new} and if T_i^{new} is greater than or equal to the T^{min} of the application, it uses the elastic compression algorithm to find a feasible period reassignment. Before it calls the task compression algorithm, it modifies the period parameter of the task from T_i^d to T_i^{new}. The task compression algorithm then takes as input the updated taskset parameters and the current resource supply utilization U_Γ to adapt the periods of the tasks to keep the application schedulable. If T_i^{new} is less than T^{min} of the application, it sets the T^{min} value equal to T_i^{new}. It then generates the new resource supply parameters via the `GET_RESOURCE_INTERFACE` function. This function takes as input the updated taskset parameters and the value k satisfying Eq. (5.2) or Eq. (5.4). It then finds the resource supply utilization bound according to Eq. (5.5) or Eq. (5.6). It uses this value to find a solution according to the approach given in [10]. The Elastic manager requests the GCSR manager to modify its resource supply parameters via the `REQUEST_RESOURCE_UPDATE` function. The GCSR manager tries to allocate resources from the spare capacity while maintaining system schedulability. It returns success if the requested resource supply parameters can be accommodated or returns failure along with the maximum resource supply utilization that it can provide. In case of failure, it is up to the application to decide on how to handle this failure.

5.4 Evaluation

We evaluate the performance of the proposed framework in the context of EDF scheduling. To demonstrate the advantages of the proposed method, we generated

Algorithm 2 PRM Elastic Scheduler

function GET_PERIOD_INTERFACE($\mathbf{A}, T^{min}$,k)
  $U_\Gamma^{new} \leftarrow$ FIND_UTILIZATION_BOUND(U_A)
  $\Gamma(\Theta_{new}, \Pi_{new}) \leftarrow$ FIND_SOLUTION($k, \mathbf{A}, U_\Gamma^{new}$)
  $success \leftarrow$ REQUEST_RESOURCE_UPDATE($\Gamma(\Theta_{new}, \Pi_{new})$)
  if $success == true$ **then**
    return $\Gamma(\Theta_{new}, \Pi_{new}), U_\Gamma^{new}$
  else
    return Failure
  end if
end function
function ELASTIC_MANAGER(T_i^{new})
  $T_i^d \leftarrow T_i^{new}$
  if $T_i^{new} \geq T^{min}$ **then**
    $success \leftarrow$ TASK_COMPRESS($\mathbf{A}, U_\Gamma$)
    if $success != true$ **then**
      $handleFailure()$
    end if
  else
    $T^{min} \leftarrow T_i^{new}$
    $Interface \leftarrow$ GET_PERIOD_INTERFACE($\mathbf{A}, T^{min}, k$)
    if $Interface!$ = Failure **then**
      $success \leftarrow$ TASK_COMPRESS($\mathbf{A}, U_\Gamma$)
      if $success != true$ **then**
        $handleFailure()$
      end if
    else
      $handleFailure()$
    end if
  end if
  return
end function

900 random tasksets with each taskset consisting of 10, 20 or 30 tasks. For each taskset, we set the initial desired utilization equal to 0.25, 0.5, and 0.75. The utilization for each task was then was derived using the algorithm proposed by Griffin et al. [16]. The initial desired periods were chosen at random from a normal distribution in the range [10,100]. The WCET values were set as $C_i = U_i * T_i$. The minimum and maximum periods for each of the tasks were assigned as a function of the initial desired period, i.e., to fix the minimum period, we subtracted a random percentage in the range [10-50] from the desired period. Similarly, for the maximum period, we added a random percentage in the range [10-50] to the desired period. We assigned random integer values from a normal distribution n the range [0-10] as the elastic coefficients of the tasks. For each taskset, we then derived a periodic resource interface for the initial desired periods according to the algorithm in [10].

We set up the experiments according to the different configurations of the number of tasks N and the total desired utilization U, i.e., each configuration was defined as a pair(N,U). For each configuration, 100 random tasksets were generated. For each configuration and a random taskset, we requested a change in the desired period 100 times. For each new period request, we assigned the new period values chosen from a uniform distribution within their defined period ranges. For each configuration, we counted the number of times the elastic manager was able to modify the utilization such that the application remains schedulable. If the elastic manager failed to find a feasible solution, it would adapt the interface bandwidth[1]. The task requesting for a new period was chosen at random for each of the new period request.

Fig. 5.3 shows the distribution of requests between the elastic manager and the GCSR manager for 300 different configurations with N equal to 10 and the total utilization set to 0.25, 0.5, and 0.75. We can observe that the elastic manager was able to successfully handle a significantly large percentage of the new period requests locally. For lower utilization values, the percentage of requests handled locally was less than the percentage for the higher utilization. Fig. 5.4 shows the distribution of the requests between the elastic manager and the GCSR manager for 300 configurations with N equal to 20 and the total utilization set to 0.25, 0.5, and 0.75. Similar to the previous observations, the elastic manager was able to successfully handle a large percentage of the requests locally, while for lower utilization values, the percentage was less than the percentage for higher utilization. For the remaining 300 configurations, we set N equal to 30 and the total utilization was set to 0.25, 0.5, and 0.75. In this case, the elastic manager was able

[1]Note that for this evaluation we did not check for system schedulability since a failure of system schedulability test could only mean that the application's resource demands were not feasible.

to successfully handle a higher percentage of requests at lower utilization values when compared to higher utilization tasksets. This is shown in Fig. 5.5. Here, even when compared to the lower number of tasks with the same total utilization, there are more requests for system-level bandwidth adaptation.

In another experiment, we modified the range of the minimum and maximum periods of the taskset from the initial [10,50] percent values to [10,100] percent. When the difference between the initial desired period and the minimum and maximum periods is increased, fewer requests were handled at the application level compared to the system level. As shown in Fig. 5.6, for the configuration of 10 tasks and utilization set to 0.25, more than 70% of the requests are for bandwidth adaptation when the difference between the minimum and maximum periods was changed to [10,100] percent from [10-50] percent. Another significant observation is that for certain configurations and tasksets, all of the 100 new period requests could either be handled locally by the elastic manager or handled only at the system level by the GCSR manager. This can be observed in Fig. 5.7. This indicates that the approach based on setting the initial resource supply according to the initial desired periods can have a considerable impact on the number of requests that need bandwidth adaptations.

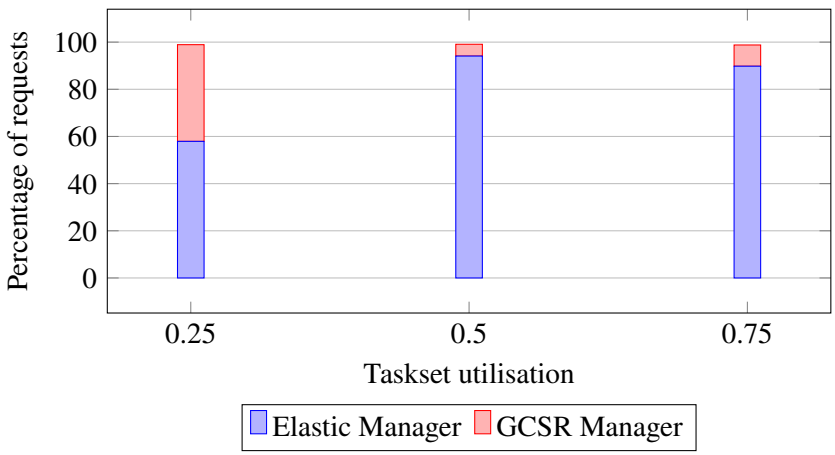

Figure 5.3. Percentage of requests handled by Elastic and GCSR manager for taskset size 10.

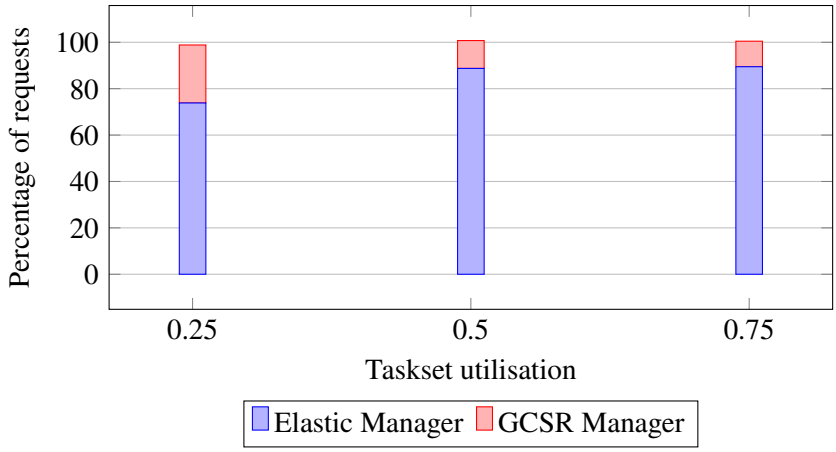

Figure 5.4. Percentage of requests handled by Elastic and GCSR manager for taskset size 20.

5.5 Related Work

The concept of elastic tasks was introduced by Buttazzo et al. [3] to model applications whose computational demands can occasionally exceed the available capacity by allowing the application to modify the demand by changing the frequency of its jobs through an elastic coefficient. This was extended to address resource sharing within the elastic task model in [11]. Chantem et al. [17, 18] reformulated the problem as a quadratic optimization problem and showed that the original elastic tasks compression algorithm was indeed a solution to solving a quadratic problem. Tian et al. [19] extended the modified problem to include a "Quality-of-Control" metric as a part of the objective function of the quadratic optimization problem. More recently, Orr et al. [20, 21] provided algorithms to schedule sequential elastic tasks on multiprocessor systems and further extended the concept of the elastic task to federated DAG-based parallel task systems in [22, 23]. Beccari et al. [24, 25] provided alternative algorithms to schedule similar applications by expressing the task period ranges in a linear programming formulation.

Hierarchical scheduling of applications was encapsulated in a compositional real-time scheduling framework by Lee et al. [10]. In this framework, the computational demand of an application was abstracted with a single demand interface as a pair of capacity and period and the resource supply server was abstracted as a periodic resource model where each server was guaranteed a reserved capacity Θ every Π time units. Easwaran et al. [26] extended the periodic resource model to include the deadline parameter, where each server was guaranteed a reserved

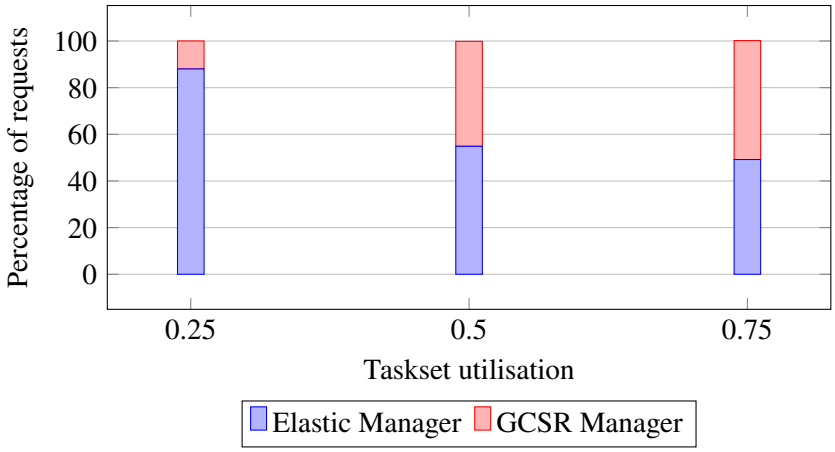

Figure 5.5. Percentage of requests handled by Elastic and GCSR manager for taskset size 30.

capacity Θ within D time units, in every time interval Π. Dewan et al. [27, 12] provided algorithms to find an approximate allocation of bandwidth for a set of periodic and sporadic tasks under the periodic resource model. Khalilzad et al. [6] proposed an adaptive hierarchical scheduling model to accommodate the adaptive behaviour of the periodic and sporadic tasks by changing the bandwidth allocation. In contrast, this paper assumes that a bandwidth allocated for a server under the periodic resource model remains constant and that the workload within the server can be adapted according to the elastic task model. However, if the elastic assignment fails, a request for a new bandwidth allocation will be made. We note that the proposed solution does not take into account possible bandwidth reclamation or mixed-criticality-based approaches to assign new bandwidths if no schedulable allocation can be made. Instead, we leave it to the individual application to handle such failures.

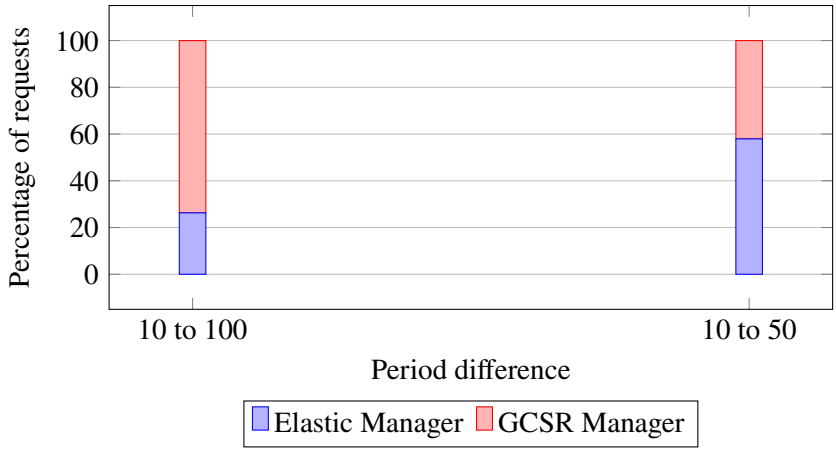

Figure 5.6. Percentage of requests handled by Elastic and GCSR manager for different intervals between the minimum and the maximum periods.

5.6 Conclusion

Many real-time applications designed to accommodate their behaviour at run-time depend upon user configuration or the physical environment in which they operate. Further, for open real-time systems, the applications can be developed independently and can be run on the same hardware. To accommodate such adaptive behaviour and minimize the impact of an individual application's variability in its timing and resource demands, we proposed a two-level scheduling framework based on the periodic resource model and the elastic task model. We provided a mechanism based on the utilization tests to enable the execution of the elastic applications in a compositional real-time system. Further, by combining the application-level adaptation along with the system-level bandwidth modifications, we have shown that a large percentage of task modifications can be handled by the framework at the application level. If the local adaptation fails, the system-level reallocation provides an additional mechanism to support the scheduling of elastic applications. However, for some cases, if both the levels fail to find a schedulable modification, it is up to the application to handle such failures. Overall, our combined approach improves the number of application-level variations that can be handled without affecting the property of independence of other co-running applications of the same processor. In future work, we intend to investigate techniques related to mixed-criticality and compositional real-time systems to reallocate resources at the system level.

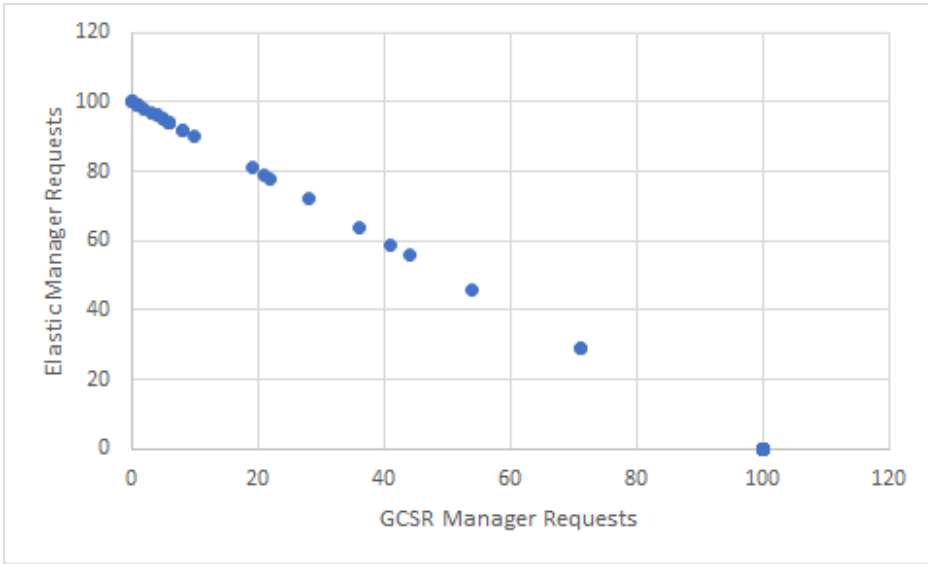

Figure 5.7. Requests handled by Elastic and GCSR Manager for 100 tasksets of a single configuration (20,0.25).

Acknowledgement

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation, and by the Swedish Knowledge Foundation (KKS) under the FIESTA project.

Bibliography

[1] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*, pages 21–28, 1999.

[2] D. Simon, D. Robert, and O. Sename. Robust control/scheduling co-design: application to robot control. In *11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 118–127, 2005.

[3] Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. In *Proceedings - Real-Time Systems Symposium*, pages 286–295. IEEE, 1998.

[4] Shaik Mohammed Salman, Vaclav Struhar, Alessandro V Papadopoulos, Moris Behnam, and Thomas Nolte. Fogification of industrial robotic systems: Research challenges. In *Proceedings of the Workshop on Fog Computing and the IoT*, pages 41–45, 2019.

[5] Mohammed Salman Shaik, Václav Struhár, Zeinab Bakhshi, Van-Lan Dao, Nitin Desai, Alessandro V Papadopoulos, Thomas Nolte, Vasileios Karagiannis, Stefan Schulte, Alexandre Venito, et al. Enabling fog-based industrial robotics systems. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 61–68. IEEE, 2020.

[6] Nima Moghaddami Khalilzad, Thomas Nolte, Moris Behnam, and Mikael Åsberg. Towards adaptive hierarchical scheduling of real-time systems. In *ETFA2011*, pages 1–8, 2011.

[7] Nima Moghaddami Khalilzad, Moris Behnam, and Thomas Nolte. Multi-level adaptive hierarchical scheduling framework for composing real-time

systems. In *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 320–329, 2013.

[8] Nima Khalilzad, Mohammad Ashjaei, Luis Almeida, Moris Behnam, and Thomas Nolte. Towards adaptive resource reservations for component-based distributed real-time systems. *SIGBED Rev.*, 12(3):24–27, August 2015.

[9] Nima Khalilzad, Fanxin Kong, Xue Liu, Moris Behnam, and Thomas Nolte. A feedback scheduling framework for component-based soft real-time systems. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 182–193, 2015.

[10] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):30:1–30:39, 2008.

[11] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[12] Farhana Dewan and Nathan Fisher. Bandwidth allocation for fixed-priority-scheduled compositional real-time systems. *ACM Trans. Embed. Comput. Syst.*, 13(4):91:1–91:29, 2014.

[13] Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 159–168, 2006.

[14] Ricardo Marau, Karthik Lakshmanan, Paulo Pedreiras, Luís Almeida, and Raj Rajkumar. Efficient elastic resource management for dynamic embedded systems. *Proc. 10th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications, TrustCom 2011, 8th IEEE Int. Conf. on Embedded Software and Systems, ICESS 2011, 6th Int. Conf. on FCST 2011*, pages 981–990, 2011.

[15] Stefan Groesbrink, Luis Almeida, Mario De Sousa, and Stefan M. Petters. Towards certifiable adaptive reservations for hypervisor-based virtualization. *Real-Time Technology and Applications - Proceedings*, 2014-October(October):13–24, 2014.

[16] David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 76–88, 2020.

[17] Thidapat Chantem, Xiaobo Sharon Hu, and M. D. Lemmon. Generalized elastic scheduling. *Proceedings - Real-Time Systems Symposium*, pages 236–245, 2006.

[18] Thidapat Chantem, Xiaobo Sharon Hu, and Michael D. Lemmon. Generalized elastic scheduling for real-time tasks. *IEEE Transactions on Computers*, 58(4):480–495, 2009.

[19] Yu Chu Tian and Li Gui. QoC elastic scheduling for real-time control systems. *Real-Time Systems*, 47(6):534–561, 2011.

[20] James Orr, Chris Gill, Kunal Agrawal, Jing Li, and Sanjoy Baruah. Elastic Scheduling for Parallel Real-Time Systems. *ACM Subject Classification*, 6(1):5:1–5:14, 2019.

[21] James Orr and Sanjoy Baruah. Algorithms for implementing elastic tasks on multiprocessor platforms: a comparative evaluation. *Real-Time Systems*, 57(1):227–264, 2021.

[22] James Orr, Christopher D. Gill, Kunal Agrawal, Sanjoy K. Baruah, Christian Cianfarani, Phyllis Ang, and Christopher Wong. Elasticity of workloads and periods of parallel real-time tasks. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10-12, 2018*, pages 61–71. ACM, 2018.

[23] James Orr, Johnny Condori Uribe, Christopher D. Gill, Sanjoy K. Baruah, Kunal Agrawal, Shirley Dyke, Arun Prakash, Iain Bate, Christopher Wong, and Sabina Adhikari. Elastic scheduling of parallel real-time tasks with discrete utilizations. In Liliana Cucu-Grosjean, Roberto Medina, Sebastian Altmeyer, and Jean-Luc Scharbarg, editors, *28th International Conference on Real Time Networks and Systems, RTNS 2020, Paris, France, June 10, 2020*, pages 117–127. ACM, 2020.

[24] Giuseppe Beccari, Stefano Caselli, Monica Reggiani, and Francesco Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*, pages 21–28. IEEE Computer Society, 1999.

[25] Giuseppe Beccari, Stefano Caselli, and Francesco Zanichelli. A technique for adaptive scheduling of soft real-time tasks. *Real Time Syst.*, 30(3):187–215, 2005.

[26] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 129–138. IEEE Computer Society, 2007.

[27] Nathan Fisher and Farhana Dewan. A bandwidth allocation scheme for compositional real-time systems with periodic resources. *Real Time Syst.*, 48(3):223–263, 2012.

Chapter 6

Paper B: Multi-Processor Scheduling of Elastic Applications in Compositional Real-Time Systems

Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, Thomas Nolte.
In the Journal of Systems Architecture, Volume 122, 2021.

Abstract

Scheduling of real-time applications modelled according to the periodic and the sporadic task model under hierarchical and compositional real-time systems has been widely studied to provide temporal isolation among independent applications running on shared resources. However, for some real-time applications which are amenable to variation in their timing behaviour, using these tasks models results in pessimistic solutions. The elastic task model addresses this pessimism by allowing the timing requirements of the application's tasks to be specified as a range of values instead of a single value. While the scheduling of elastic applications on dedicated resources has received considerable attention, there is limited work on scheduling of such applications in hierarchical and compositional settings.

In this paper, we evaluate different earliest deadline first scheduling algorithms to schedule elastic applications in a minimum parallelism supply form reservation on a multiprocessor system. Our evaluation indicates that the proposed approach provides performance comparable to the current state-of-art algorithms for scheduling elastic applications on dedicated processors in terms of schedulability.

6.1 Introduction

The elastic task model enables convenient modelling of real-time applications that can tolerate some amount of variation in their timing behaviour (changes in interarrival times or changes in execution times) while still maintaining their functional correctness. In uniprocessor systems and multi-processor systems where the resources are fully dedicated to an elastic application, the variation can be managed online by modifying the utilization of the application's tasks to ensure schedulability. Scheduling multiple elastic applications on a shared resource, however, requires the use of reservation-based mechanisms to minimize the impact of such variations on the co-scheduled applications. While different solutions have been proposed for scheduling applications with tasks specified according to the periodic or the sporadic task model under reservation-based mechanisms, there is limited work related to scheduling of elastic applications under reservation schemes. In this context, we propose a scheduling framework for executing elastic applications under the minimum-parallelism periodic resource supply model (MPS) on multi-processor systems. While the elastic task model allows defining the task parameters to capture the variable timing requirements of the application, the minimum-parallelism periodic resource supply model [1, 2] provides a relatively simpler mechanism to reserve the resource supply on a multi-processor system. Moreover, under certain period assignment constraints, the MPS model dominates other comparable state-of-art techniques [2].

Concretely, in this paper, we address the following questions:

Q1 Given an application with elastic tasks, what is a feasible bandwidth required to schedule the applications in an MPS reservation on a multiprocessor system?

Q2 Given a fixed bandwidth MPS reservation, what mechanism can the elastic application tasks adopt to remain schedulable?

Q3 Given an elastic application, can a schedulable reservation be found if the application requests for a modified bandwidth reservation?

Q4 What is the trade-off between the proposed approach and the current state-of-art approaches?

6.2 Proposed System Model

We consider the scheduling of a real-time application specified according to the elastic task model with each task having an implicit deadline(See Section. 6.2.1). At the system level, we assume that the CPU resources are made available to each application according to the multi-processor Minimum Parallelism Supply Model (MPS) [2, 1] (see Section 6.2.2). We assume that each application provides its own local scheduling mechanism, based on the partitioned dynamic-priority scheduling implementing the Earliest Deadline First (EDF) policy. The dedicated full processors do not need a system-level scheduler but rather use the application provided scheduler. The partial processor can be managed by any scheduler which can ensure that the partial processor provides supply according to the periodic resource model [3].

6.2.1 The Basic Elastic Task Model

We define an elastic application **A** as a set of n tasks where each task is specified as $\tau_i = \{C_i, T_i^{min}, T_i^d, T_i^{max}, E_i\}$. Here, C_i is the Worst-Case Execution Time (WCET) of the task while T_i^{min} and T_i^{max} specify the minimum and the maximum interarrival time between consecutive jobs of τ_i. T_i^d represents the desired period of the task τ_i. The elastic co-efficient E_i determines the flexibility of the task τ_i to change in its period [4]. For instance, if E_i is defined to be in the range $[0, 1]$, $E_i = 0$ ensures that the task's desired period cannot be modified, implying that $T_i^{min} = T_i^d = T_i^{max}$. Similarly, $E_i = 1$ indicates that the task's period can be modified to take up values up to its maximum period. Further, the elastic coefficient acts as a weighting factor, determining the extent to which its utilization

Task ID	WCET	T_i^{min}	T_i^d	T_i^{max}	E_i
τ_1	4	40	120	240	1
τ_2	8	40	80	360	0.75
τ_3	12	240	240	480	0.5
τ_4	12	200	240	600	0.25
τ_5	4	40	40	40	0

Table 6.1. An Elastic Task Set

can be reduced in relation to other tasks. We use T_i (without any postscript) to indicate the current inter-arrival time of τ_i. An example of an elastic taskset is shown in Table 6.1. Here, the period of the task τ_1 can be extended up to its maximum period, while the task τ_5 can only execute with its desired period.

The utilization of a task τ_i is defined as $U_i = \frac{C_i}{T_i}$. The minimum and maximum utilization of each task is represented by $U_i^{min} = \frac{C_i}{T_i^{max}}$ and $U_i^{max} = \frac{C_i}{T_i^{min}}$, respectively. The desired utilization is represented by $U_i^d = \frac{C_i}{T_i^d}$. The desired application utilization is given by the sum of the desired utilization of the individual tasks. i.e., $U^d = \sum_{i=1}^{n} U_i^d$. Similarly, the minimum and maximum application utilization is given by $U^{min} = \sum_{i=1}^{n} U_i^{min}$ and $U^{max} = \sum_{i=1}^{n} U_i^{max}$. The relative deadline of each job of an elastic task is equal to its current period at runtime. If a task requests for a change in its current period, its desired utilization U_i^d is updated and a schedulable utilization for rest of the tasks is calculated. An elastic application is said to be schedulable if $U^d \leq U^{ub}$, where U^{ub} is the utilization upper-bound for a given scheduling algorithm. It is assumed that the deadline is elastic-implicit. Note that when the utilization of a task is changed to accommodate increased utilization of another task, the reduced utilization is used to check for schedulability instead of the initial desired period.

6.2.2 Minimum-Parallelism Resource Supply Model

The resource supply provided to the application is based on the minimum parallelism resource supply model [1, 2]. Here the resource supply is provided by reserving *m* dedicated processors for an application and at most one periodic resource model based partial processor [3] for the exclusive use of the application. Here, the partial processor is defined as $\Gamma(\Theta, \Pi)$, where Θ is the periodic resource allocation time and Π is the resource period. Essentially, the periodic resource Γ provides an application with Θ time units of CPU time every Π time units. The utilization of the partial resource supply is defined as $U_\Gamma = \frac{\Theta}{\Pi}$.

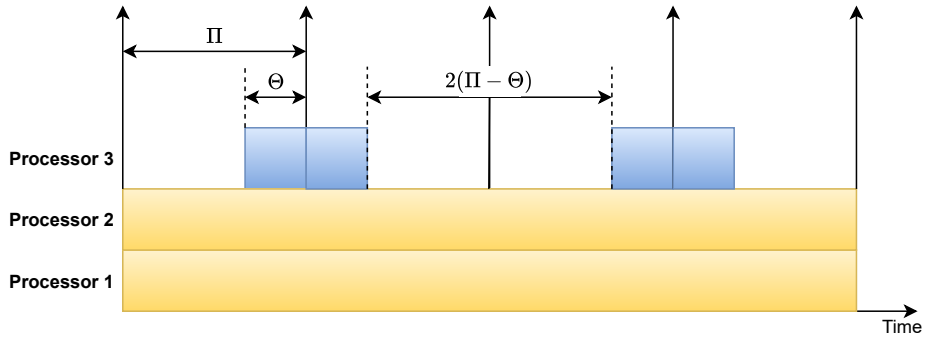

Figure 6.1. Minimum Parallelism Resource Supply Model.

Task ID	U_i^d
τ_1	0.55
τ_2	0.50
τ_3	0.50
τ_4	0.45
τ_5	0.25
τ_6	0.15

Table 6.2. taskset with desired utilization.

Fig. 6.1 shows the resource supply provided according to the MPS model with 3 processors. Here, processor 1 and processor 2 are fully dedicated to the application, while processor 3 is available partially and modelled according to the periodic resource model. In the worst case, i.e., when the resource budget is exhausted at the beginning of its period and the next instant it is available close to the end of the next period, the no supply interval of the partial processor is equal to $2(\Pi - \Theta)$ [3].

Generating the resource supply reservation parameters As the MPS model requires reserving m full processors and one partial processor, we need to identify the number of dedicated processors along with the resource supply parameters of the partial processor necessary to schedule the application. We identify these values based on the initial desired utilization. As we consider partitioned EDF scheduling, we apply the reasonable allocation decreasing (RAD) partitioning heuristic specified in [5]. For example, consider the taskset with its desired utilization in Table 6.2. The tasks τ_1 and τ_4 are assigned to the same processor since their utilization sums up to one. Similarly τ_2 and τ_3 are assigned to another processor.

As τ_5 and τ_6 have their utilization much less than the utilization limit for a fully dedicated processor under EDF, they are assigned a partial processor. We apply the method described in [3] to identify the capacity and the period of the partial processor to schedule τ_5 and τ_6.

Schedulability Bounds For the dedicated processors, we assume the uniprocessor utilization bound of one for partitioned EDF. For the partial processor, we use the utilization bound defined according to the periodic resource model. Particularly, the application tasks assigned to the partial processor with utilization $U_{\mathbf{A}}$ are schedulable under EDF scheduling policy if the partial processor resource supply utilization satisfies Eq. (6.1) (Eq. 21 in [3]).

$$U_{\Gamma,EDF}(k) = \frac{(k+2).U_{\mathbf{A}}}{k+2(U_{\mathbf{A}})}, \tag{6.1}$$

where,

$$k = \max\left\{ k \in \mathbb{Z} \middle| (k+1)\Pi - \Theta - \frac{k\Theta}{k+2} < T^{min} \right\} \tag{6.2}$$

and T^{min} is the smallest period among the tasks assigned to the partial processor.

6.3 Proposed Solution

As the minimum parallelism resource supply model allows at most one partial processor, the remaining resources are provided by dedicated processors. Therefore, we first consider scheduling of elastic applications on dedicated multiprocessors. In this section, we summarize the current state-of-art solution to the scheduling problem based on the partitioned heuristics proposed by Orr et al. [6]. We then extend this solution to include the partial processor.

6.3.1 Scheduling Elastic Applications on Dedicated Multi-Processors

The schedulability of elastic applications on dedicated multi-processors was evaluated by Orr et al. [6] using both global as well as partitioned scheduling approaches. Their work highlighted the relatively better schedulability performance of the partitioned approach when compared to global scheduling for elastic applications. We summarize their approach below and refer to it as the global re-partitioning approach in the rest of the paper.

The Global Re-Partitioning Approach This approach involves iteratively compressing the utilization and re-partitioning the taskset until a schedulable partition is found. The algorithm iterates over the values in the range $[0, \Phi]$, where Φ is calculated according to (6.4), to find the smallest value of λ such that the utilization assigned to each task according to (6.3) can result in a schedulable partition. During each iteration, it assigns a value to λ and calculates the utilization to be assigned to each task. For this set of utilization values, it then applies a predetermined partitioning heuristic such as the `reasonable allocation decreasing(RAD)` [5], where the tasks are ordered according to monotonically decreasing utilization values and each task is then assigned to the processor on which it fits. The task to processor assignment can be based on

1. First Fit: Assign the task to the first processor on which it fits. i.e., the task meets its schedulability conditions.

2. Best Fit: Assign the task to the processor with minimum remaining capacity after its allocation.

3. Worst Fit: Assign the task to the processor with maximum remaining capacity after its allocation.

If a task remains unassigned, the value of λ is incremented by a granularity constant ϵ and the process is repeated until either all tasks are assigned to processors, resulting in a successful allocation, or the value of λ exceeds Φ, resulting in failure.

$$U_i = \max(U_i^d - \lambda E_i, U_i^{min}) \tag{6.3}$$

$$\Phi = \max\left(\frac{U_i^d - U_i^{min}}{E_i}\right) \tag{6.4}$$

An Example Consider an elastic taskset with values given in Table.6.3 that needs to be scheduled on a 2 core processor. According to the first fit heuristic and considering the desired utilization values, the tasks τ_1 and τ_2 are assigned to core 1 while τ_3 and τ_4 are assigned to core 2. At runtime, suppose that the task τ_1 requests for a new utilization equal to 0.65. The algorithm tries to find a suitable value for λ such that a schedulable partition is found. Observing the utilization values in this example, one can notice that the new utilization request of the task τ_1 could have been easily accommodated by changing the utilization of task τ_2 from its desired value to its minimum utilization value without the re-partitioning step. Based on this observation, we propose a new algorithm that applies the utilization modification algorithm on a per-core basis and only re-partitions the taskset if the

Task ID	U_i^{min}	U_i^d	U_i^{max}	E_i
τ_1	0.25	0.55	0.75	1
τ_2	0.25	0.45	0.65	1
τ_3	0.25	0.50	0.75	1
τ_4	0.25	0.50	0.75	1

Table 6.3. An Elastic Task Set

Task ID	U_i^{min}	U_i^d	U_i^{max}	E_i
τ_1	0.25	0.55	0.75	1
τ_2	0.35	0.45	0.65	1
τ_3	0.15	0.50	0.75	1
τ_4	0.15	0.50	0.75	1

Table 6.4. An Elastic Task Set with Unschedulable Demand

per-core approach fails to successfully accommodate the requests for increased utilization values.

Utilization Modification on a Single Core While the re-partitioning approach is relatively straightforward to implement, one disadvantage of this approach is that if a taskset is not schedulable, the global re-partitioning can cause multiple task migrations resulting in additional book-keeping overhead. Further, if the temporal isolation strategies such as cache partitioning are part of an overall solution, especially those based on task-based cache allocation, then task migrations may require re-partitioning of the caches adding to the overhead. One way to minimize such overheads is to limit the utilization modification to tasks running on the same processor or the partial processor as the task requesting the increased utilization.

An Example While the request for increased utilization of the task τ_1 in the previous example could be successfully managed following the per-core utilization approach, this approach can fail to find a schedulable solution for other requests and tasksets. To illustrate this, consider the elastic taskset as given in Table. 6.4, which is similar to the previous example but with modified minimum utilization values. Based on the desired utilization values, the tasks τ_1 and τ_2 are assigned to core 1 while τ_3 and τ_4 are assigned to core 2. At runtime, suppose that the task τ_1 requests for a new utilization equal to 0.75. Applying the per-core approach would require the utilization of the task τ_2 to be reduced to 0.25. Since the minimum utilization of task τ_2 is greater than the required value, this results in an unschedulable condition.

If re-partitioning was applied, however, the request could be successfully handled by assigning τ_1 and τ_3 to the same core and reducing the utilization τ_3 to 0.25. Similarly, τ_2 and τ_4 could be assigned to the same core to ensure schedulability.

Based on these observations, we describe next a combined approach that exploits the advantages offered by the per-core approach while improving the range of schedulable elastic tasksets through limited re-partitioning.

The Combined Approach Although the original re-partitioning approach finds schedulable solutions for most of the evaluated scenarios(See Section. 6.4), the associated overheads can be minimised by first applying the utilization modification algorithm to the tasks running on the same processor as the task requesting for increased utilization to find a schedulable utilization on that processor. Since the complexity of the utilization modification algorithm is a function of the number of tasks and the number of processors, by reducing the number of tasks whose utilization should be adjusted, the efficiency of the algorithm can be improved. Indeed, if a schedulable utilization is not found, then re-partitioning considering all the cores and the complete taskset cannot be avoided and as such, the overheads related to task migration (and if applicable, cache partitioning) remain the same as in the original approach. The general steps for the combined approach are as follows:

1. Order the taskset either in an (i) arbitrary manner, or (ii) monotonically increasing utilization, or (iii) monotonically decreasing utilization.
2. Order the processors in some arbitrary manner.
3. Allocate tasks to processors according to a `Reasonable Allocation` scheme.
4. When a task makes a request for increased utilization, apply the iterative utilization modification algorithm only to tasks on the same core.
5. If such a request fails, apply the iterative utilization modification algorithm including re-partitioning by considering all the cores.

The combined approach is able to find schedulable solutions similar to the original re-partitioning approach but with improved efficiency since the re-partitioning step is applied only if the per-core utilization modifications fail. In some of the evaluated scenarios, the results indicate per-core utilization modifications are sufficient to keep the application schedulable while completely avoiding the re-partitioning step. In limited cases where the per-core utilization fails, the re-partitioning approach successfully finds a schedulable partition.

6.3.2 Extension to Minimum Parallelism Resource Supply Model

The approaches discussed in the previous section considering dedicated processors can be extended to the minimum parallelism resource supply reservation model with some minor modifications. One of the key distinguishing characteristics between the MPS reservation model and the fully dedicated processors is the presence of the partial processor. Since the partial processor is only available periodically, under worst-case conditions, there can be a no supply interval equal to $2(\Pi - \Theta)$ (See Fig. 6.1). Any task with a period less than this no supply interval cannot be allocated to the partial processor. As a consequence, the re-partitioning step should not only consider the utilization but also the period of the tasks. Moreover, in the case where both per-core utilization modification, as well as re-partitioning, fail due to the period constraints, depending on the availability of resources on the core executing the partial processor, a re-allocation of the bandwidth on the partial processor can be considered. The general steps of the proposed approach are as follows:

1. Order the taskset (i) arbitrary manner (ii) monotonically increasing desired utilization or (iii) monotonically decreasing desired utilization.
2. Order the processors in some arbitrary manner.
3. Allocate tasks to processors according to a `Reasonable Allocation` scheme.
4. if any task remains unallocated, generate resource supply parameters for the partial processor according to the initial desired utilization values of the unallocated tasks.
5. When a task makes a request for increased utilization, apply the iterative utilization modification algorithm only to tasks on the same core (or partial processor as applicable).
6. if such a request fails, apply the iterative utilization modification algorithm by including re-partitioning considering all the dedicated cores. Include partial processor only if the failing request belongs to the partial processor.
7. During re-partitioning, only allocate tasks whose periods are greater than the worst-case no-supply duration on the partial processor while ensuring that the total utilization of the task assigned to the partial processor remains below the partial processor utilization according to Eq.(6.1).
8. If this request fails, modify the bandwidth of the partial processor if resources are available and repeat steps 3 and 4.

6.4 Evaluation

We evaluated the proposed approach for the case of fully dedicated processors by considering randomly generated tasksets with a varying number of tasks and utilization values along with a different number of processors and partitioning heuristics. Here we present the results of the 16 such configurations as detailed in Table 6.5. For each configuration, we measured the number of successful requests along with the computation time (based on the Query Performance Counter provided by the Windows OS) for a prototype implementation of the proposed algorithms.

Taskset Generation The desired utilization of each task of a taskset was generated using the algorithm proposed by Griffin et al. [7]. The minimum utilization was generated by subtracting a randomly generated percentage (taken from a uniform distribution) from the desired utilization. Similarly, the maximum utilization was generated by adding a randomly generated percentage to the desired utilization while ensuring that the maximum utilization for each task was under one. Each request for increased utilization was generated by selecting a random task and a uniformly distributed random utilization value from within the task's desired and maximum utilization. The elastic co-efficient was assigned to each task by randomly generating real values taken from a uniform distribution between [1-10].

Configurations To compare the performance of the proposed approaches with a state-of-art method, we evaluated different configurations by varying the number of processors, and tasks along with different percentage limits on the utilization values to define the maximum and minimum utilization of each task. For the desired utilization, the maximum utilization of each task was set to be not greater than 0.5. Further, the tasks were ordered according to monotonically decreasing utilization values. The first fit approach was used as the task to processor assignment strategy. Table 6.5 shows the different configurations where the column `maxmin` indicates the maximum and minimum values used to define the utilization values. For example, the values $(1, 0.5)$ indicate that the maximum utilization that a task can have is in the interval $[U_d, U_d + (1 \cdot U_d)]$, while the minimum utilization a task can have is in the interval $[U_d - (0.5 \cdot U_d), U_d]$. The values under the column "configuration" refer to the respective (row) configurations settings and are used as identifiers on the x-axis in the associated graphs.

Configuration	Maxmin	No.of tasks	No.of Processors
1	1, 0.5	100	2
2	1, 0.5	100	4
3	1, 0.5	100	8
4	1, 0.5	100	16
5	1, 0.5	200	2
6	1, 0.5	200	4
7	1, 0.5	200	8
8	1, 0.5	200	16
9	0.5, 0.5	100	2
10	0.5, 0.5	100	4
11	0.5, 0.5	100	8
12	0.5, 0.5	100	16
13	0.5, 0.5	200	2
14	0.5, 0.5	200	4
15	0.5, 0.5	200	8
16	0.5, 0.5	200	16

Table 6.5. Different Configuration Settings.

6.4.1 Results

We evaluated the different approaches for schedulability along with the computation times under different configuration requirements. For each configuration, we generated ten different tasksets and for each taskset, we simulated 1000 requests for utilization modifications. For each utilization request, we assigned the new utilization values chosen from a uniform distribution between the desired utilization and the maximum utilization. Moreover, each task requesting increased utilization was chosen randomly.

Schedulability The schedulability of the different configurations for the three methods is shown in Fig. 6.2. Here, GP refers to the global re- partitioning approach of ORR et al., while PCM refers to the per-core utilization modification approach and CA refers to the combined approach of per-core modifications and re-partitioning. The results show that the combined approach has a 100 percent success value while the per-core modifications have a success value of 95 percent in the worst-case indicating that in most cases, the per-core modification is sufficient. In case of failure, the limited re-partitioning associated with the combined approach can find a schedulable solution. The variation in successful requests for certain

configurations is mostly due to the randomness in the generated tasksets and the tasks requesting for the modified utilization.

For configurations with a reduced number of tasks (Table 6.6), the schedulability performance of the per-core approach was around 93 percent in the worst-case scenario, which was relatively worse than compared to tasksets with a larger number of tasks while the other two approaches had 100 percent success.

Configuration	Maxmin	No.of tasks	No.of Processors
1	1, 0.5	20	2
2	1, 0.5	20	4
3	1, 0.5	20	8
4	0.5, 0.5	20	2
5	0.5, 0.5	20	4
6	0.5, 0.5	20	8

Table 6.6. Configuration with a smaller number of tasks.

Computation time For each configuration, we measured the computation time required to find a schedulable solution and to report a failure if no solution is found through a prototype implementation of the proposed approaches on a Windows system. Fig. 6.3 shows the average computation time per request for each of the different configurations. The average was calculated by measuring 10000 requests. The results indicate that the per-core and the combined approach perform relatively better compared to the global re-partitioning approach. In certain scenarios, the average computation time of the combined approach is 6 times better than the global re-partitioning approach. This improvement is mostly due to the fact that most requests are managed within the core and that re-partitioning is done only if necessary, unlike the global re-partitioning approach where every new request results in re-partitioning.

Fig. 6.4 shows the worst-case computation times taken by the different approaches. The results indicate that for most configurations, the proposed approaches perform better than the global partitioning approach. In certain cases, however, the worst-case performance of the combined approach is similar (or worse in limited scenarios) to the global re-partitioning approach.

For configurations with a reduced number of tasks (Table 6.6), the computation performance of all the approaches was better compared to the configurations with a larger number of tasksets. Comparing the performance of different approaches for the same number of tasks (See Fig. 6.6), the average value of the combined approach was up to two times better than the global partitioning approach. The

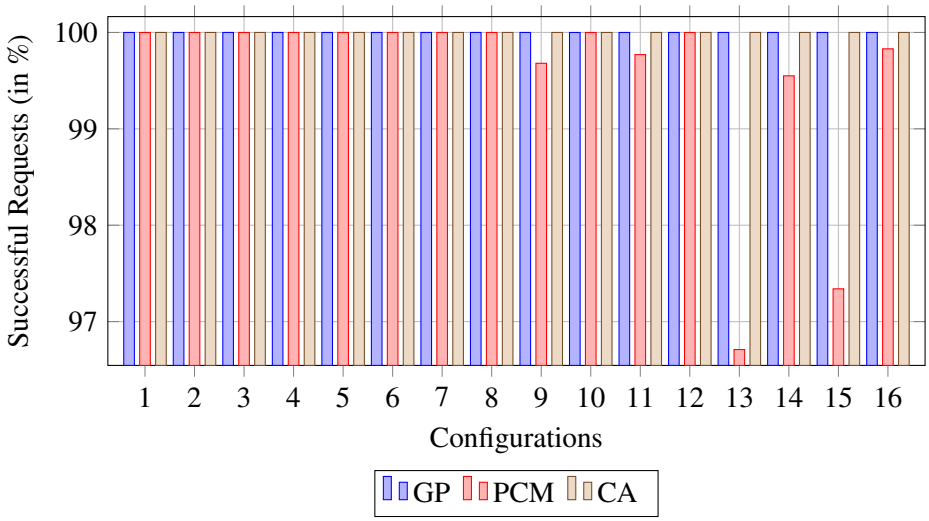

Figure 6.2. Schedulability Performance of the Different Approaches.

worst-case performance of the combined approach (See Fig. 6.7) was worse than the global approach for most of the evaluated configurations.

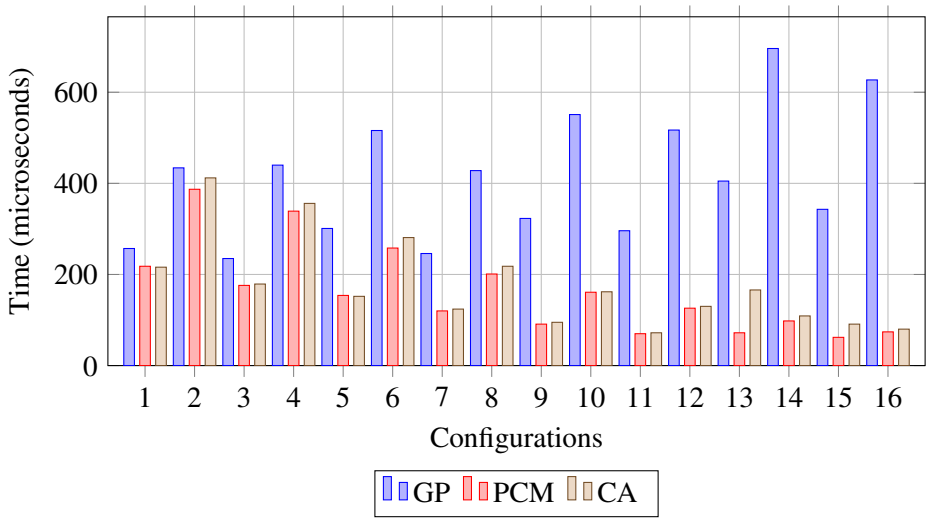

Figure 6.3. Average Computation Times of the Different Approaches.

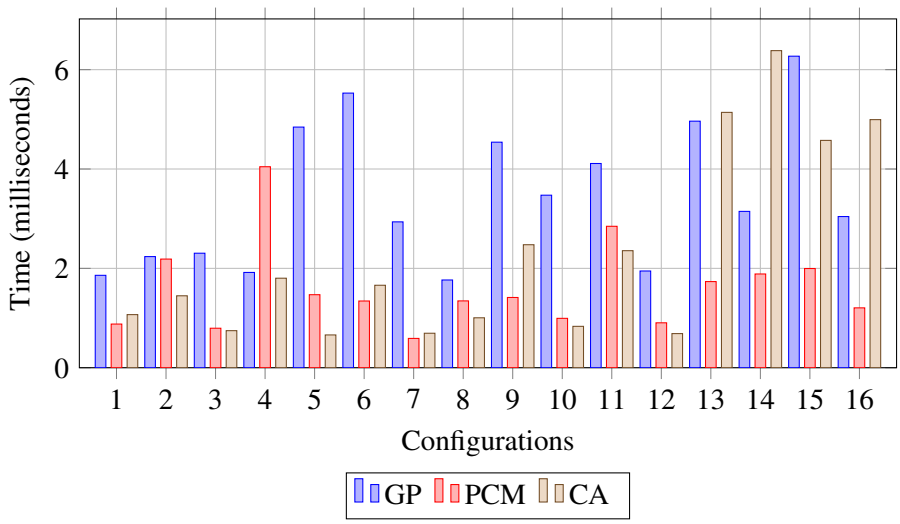

Figure 6.4. Worst Case Computation Times of the Different Approaches.

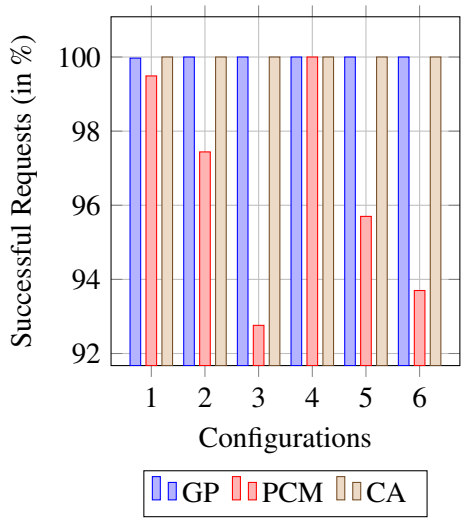

Figure 6.5. Schedulability Performance of Smaller Tasksets.

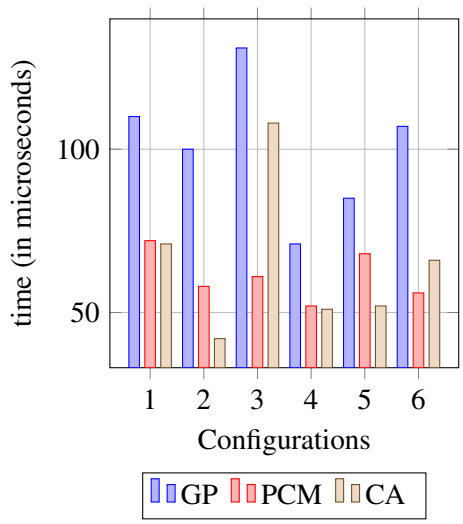

Figure 6.6. Average Computation Times for Smaller Tasksets.

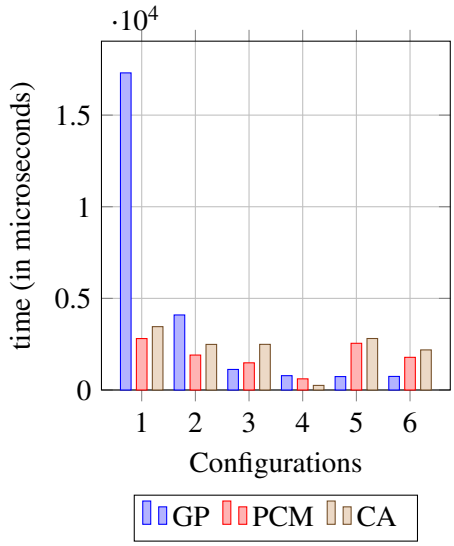

Figure 6.7. Worst Case Computation Times for Smaller Tasksets.

6.5 Related Work

Hierarchical scheduling of periodic and sporadic real-time applications was encapsulated in a compositional real-time scheduling framework by Lee et al. [3]. In this framework, the computational demand of an application was abstracted with a single demand interface as a pair of capacity and period and the resource supply server was abstracted as a periodic resource model where each server was guaranteed a reserved capacity Θ every Π time units. Easwaran et al. [8] extended the periodic resource model to include the deadline parameter, where each server was guaranteed a reserved capacity Θ within D time units, in every time interval Π. Dewan et al. [9, 10] provided algorithms to find an approximate allocation of bandwidth for a set of periodic and sporadic tasks under the periodic resource model. The periodic resource model was further extended for the case of multiprocessors by Shin et al. [11] where the periodic resource supply model was augmented with a parameter indicating the maximum concurrency with which the resource supply is provided. The tasks assigned to this resource supply were then scheduled using a global scheduler. Bini et al. [12, 13] provide a more expressive and general model for specifying multiprocessor resource supply in the form of the parallel supply function. Lee et al.[14] build upon the multiprocessor resource supply model and provide a cache-aware compositional analysis for the minimum parallelism resource supply form for the global EDF scheduling policy.

The concept of elastic tasks was introduced by Buttazzo et al. [15] to model applications whose computational demands can occasionally exceed the available capacity by allowing the application to modify the demand by changing the frequency of its jobs through an elastic coefficient. This was extended to address resource sharing within the elastic task model in [4]. Guangming [16] provided an earlier time for accelerating and adding tasks for the elastic scheduling approach. Chantem et al. [17, 18] reformulated the problem as a quadratic optimization problem and showed that the original elastic tasks compression algorithm was indeed a solution to solving a quadratic problem. Tian et al. [19] extended the modified problem to include a "Quality-of-Control" metric as a part of the objective function of the quadratic optimization problem. More recently, Orr et al. [20, 6] provided algorithms to schedule sequential elastic tasks on multiprocessor systems and further extended the concept of the elastic task to federated DAG-based parallel task systems in [21, 22]. Beccari et al. [23, 24] provided alternative algorithms to schedule similar applications by expressing the task period ranges in a linear programming formulation.

Another commonly used approach to schedule application tasks with timing variability is to modify the resource reservations. Thiele et al. [25] provide an online reconfiguration algorithm for the constant bandwidth server. Khalilzad et al. [26] proposed an adaptive hierarchical scheduling model to accommodate the adaptive behaviour of the periodic and sporadic tasks by changing the bandwidth allocation. In contrast, this paper assumes that a bandwidth allocated for a server under the periodic resource model remains constant and that the workload within the server can be adapted according to the elastic task model. However, if the elastic assignment fails, a request for a new bandwidth allocation will be made. We note that the proposed solution does not take into account possible bandwidth reclamation or mixed-criticality-based approaches to assign new bandwidths if no schedulable allocation can be made. Instead, we leave it to the individual application to handle such failures. For uniprocessor systems, we provide a method to schedule elastic applications within a periodic resource supply model reservation in [27].

6.6 Conclusion

Scheduling of elastic applications on reservation-based multiprocessors systems has not been extensively studied. To address this, we proposed a scheduling framework based on the minimum-parallelism resource reservation model and compared different partition-based EDF scheduling mechanisms. The proposed methods introduce a relatively intuitive approach to scheduling elastic tasks under

reservation schemes on multiprocessors combining the advantages of the simplicity offered by the MPS reservation and flexibility of the elastic tasks. The evaluation results indicate the proposed methods outperform the current state-of-art approaches on average, while in the worst case, none of the approaches outperforms the other. In future work, we intend to investigate methods to improve the worst-case performance and evaluate the results on a real system.

6.7 Acknowledgement

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation, and by the Swedish Knowledge Foundation (KKS) under the projects FIESTA, HERO and DPAC.

Bibliography

[1] Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Proceedings - Euromicro Conference on Real-Time Systems*, pages 191–200, 2008.

[2] Kecheng Yang and James H. Anderson. On the Dominance of Minimum-Parallelism Multiprocessor Supply. *Proceedings - Real-Time Systems Symposium*, 0:215–226, 2016.

[3] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):30:1–30:39, 2008.

[4] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[5] José María López, José Luis Díaz, and Daniel F. García. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real Time Syst.*, 28(1):39–68, 2004.

[6] James Orr and Sanjoy Baruah. Algorithms for implementing elastic tasks on multiprocessor platforms: a comparative evaluation. *Real-Time Systems*, 57(1):227–264, 2021.

[7] David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 76–88, 2020.

[8] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 129–138. IEEE Computer Society, 2007.

[9] Nathan Fisher and Farhana Dewan. A bandwidth allocation scheme for compositional real-time systems with periodic resources. *Real Time Syst.*, 48(3):223–263, 2012.

[10] Farhana Dewan and Nathan Fisher. Bandwidth allocation for fixed-priority-scheduled compositional real-time systems. *ACM Trans. Embed. Comput. Syst.*, 13(4):91:1–91:29, 2014.

[11] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 181–190, 2008.

[12] Enrico Bini, Marko Bertogna, and Sanjoy Baruah. Virtual multiprocessor platforms: Specification and use. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 437–446, 2009.

[13] Artem Burmyakov, Enrico Bini, and Eduardo Tovar. Compositional multiprocessor scheduling: the GMPR interface. *Real-Time Syst.*, 50(3):342–376, 2014.

[14] Meng Xu, Linh T.X. Phan, Insup Lee, Oleg Sokolsky, Sisu Xi, Chenyang Lu, and Christopher Gill. Cache-aware compositional analysis of real-time multicore virtualization platforms. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 1–10, 2013.

[15] Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. In *Proceedings - Real-Time Systems Symposium*, pages 286–295. IEEE, 1998.

[16] Qian Guangming. An earlier time for inserting and/or accelerating tasks. *Real-Time Syst.*, 41(3):181–194, 2009.

[17] Thidapat Chantem, Xiaobo Sharon Hu, and M. D. Lemmon. Generalized elastic scheduling. *Proceedings - Real-Time Systems Symposium*, pages 236–245, 2006.

[18] Thidapat Chantem, Xiaobo Sharon Hu, and Michael D. Lemmon. Generalized elastic scheduling for real-time tasks. *IEEE Transactions on Computers*, 58(4):480–495, 2009.

[19] Yu Chu Tian and Li Gui. QoC elastic scheduling for real-time control systems. *Real-Time Systems*, 47(6):534–561, 2011.

[20] James Orr, Chris Gill, Kunal Agrawal, Jing Li, and Sanjoy Baruah. Elastic Scheduling for Parallel Real-Time Systems. *ACM Subject Classification*, 6(1):5:1–5:14, 2019.

[21] James Orr, Christopher D. Gill, Kunal Agrawal, Sanjoy K. Baruah, Christian Cianfarani, Phyllis Ang, and Christopher Wong. Elasticity of workloads and periods of parallel real-time tasks. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10-12, 2018*, pages 61–71. ACM, 2018.

[22] James Orr, Johnny Condori Uribe, Christopher D. Gill, Sanjoy K. Baruah, Kunal Agrawal, Shirley Dyke, Arun Prakash, Iain Bate, Christopher Wong, and Sabina Adhikari. Elastic scheduling of parallel real-time tasks with discrete utilizations. In Liliana Cucu-Grosjean, Roberto Medina, Sebastian Altmeyer, and Jean-Luc Scharbarg, editors, *28th International Conference on Real Time Networks and Systems, RTNS 2020, Paris, France, June 10, 2020*, pages 117–127. ACM, 2020.

[23] Giuseppe Beccari, Stefano Caselli, Monica Reggiani, and Francesco Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *11th Euromicro Conference on Real-Time Systems (ECRTS 1999), 9-11 June 1999, York, England, UK, Proceedings*, pages 21–28. IEEE Computer Society, 1999.

[24] Giuseppe Beccari, Stefano Caselli, and Francesco Zanichelli. A technique for adaptive scheduling of soft real-time tasks. *Real Time Syst.*, 30(3):187–215, 2005.

[25] Pratyush Kumar, Nikolay Stoimenov, and Lothar Thiele. An algorithm for online reconfiguration of resource reservations for hard real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 245–254. IEEE, 2012.

[26] Nima Moghaddami Khalilzad, Thomas Nolte, Moris Behnam, and Mikael Åsberg. Towards adaptive hierarchical scheduling of real-time systems. In *ETFA2011*, pages 1–8, 2011.

[27] Shaik Mohammed Salman, Saad Mubeen, Filip Marković, Alessandro V Papadopoulos, and Thomas Nolte. Scheduling elastic applications in compositional real-time systems. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2021.

Chapter 7

Paper C: A Systematic Methodology to Migrate Complex Real-time Software Systems to Multi-Core Platforms

Shaik Mohammed Salman, Alessandro V. Papadopoulos, Saad Mubeen, and Thomas Nolte.
In the Journal of Systems Architecture, Volume 117, Aug 2021.

Abstract

This paper proposes a systematic three-stage methodology for migrating complex real-time industrial software systems from single-core to multi-core computing platforms. Single-core platforms have limited computational capabilities that prevent integration of computationally demanding applications such as image processing within the existing system. Modern multi-core processors offer a promising solution to address these limitations by providing increased computational power and allowing parallel execution of different applications within the system. However, the transition from traditional single-core to contemporary multi-core computing platforms is non-trivial and requires a systematic and well-defined migration process. This paper reviews some of the existing migration methods and provides a systematic multi-phase migration process with emphasis on software architecture recovery and transformation to explicitly address the timing and dependability attributes expected of industrial software systems. The methodology was evaluated using a survey-based approach and the results indicate that the presented methodology is feasible, usable and useful for real-time industrial software systems.

7.1 Introduction

Software evolution has been a continuous process in industrial real-time embedded software systems with new functionality, performance improvements and bug fixes introduced with each new version, revision or release [1, 2]. Many of the industrial systems have been developed over the decades [3], undergoing major revisions due to technology shifts, changing customer requirements, improved development processes, among others. One constant factor associated with the evolution of such systems is that the software architectures and the implementations have focused on single-core computing platforms. Integrating new data-intensive and computationally demanding applications withing the system, however, requires additional computational capacity. Moreover, with the decreasing availability of the single-core processors, migrating the existing software to multi-core computing platforms is becoming a necessity. By *migration*, we refer to the modification of the existing software to execute on the multi-core platforms, while ensuring that the performance and quality attributes, such as dependability [4, 5], match the current system quality and more optimistically, improved much further. Such migration is essential since the long life-cycle of existing software systems has resulted in the creation of assets that have become critical for a business [6] and that a complete redevelopment may not be feasible.

Migrating existing real-time software systems towards multi-core systems requires (i) Identifying the timing requirements of the existing software systems and (ii) Identifying the technical solutions that can improve the performance, resource usage and the timing predictability of the software systems [7, 8, 9]. Invariably, any migration approach should also address the extra-functional attributes such as scalability, maintainability and portability of the software. Furthermore, the migration should consider maximum reuse of the existing software while minimizing the re-engineering efforts.

To address these aspects for the migration of a complex real-time software system with strict timing and dependability requirements, we used a focus group discussion to formulate an open-ended Research Question (RQ),

RQ: How to migrate a complex real-time software from a single-core to a multi-core architecture with maximum software reuse and minimal re-engineering effort?

We further refined this question into the following sub-questions:

RQ1: Which migration methodology addresses the concerns of software reuse, dependability and timing requirements?

RQ2: How to evaluate and analyse the applicability of different multi-core solutions for embedded control software?

RQ3: What are the tools that facilitate the migration process?

These questions were motivated by the need for migrating a configurable robot controller software [4] developed at ABB Robotics[1], with functionality ranging from motion control to cloud connectivity. The controller software has close to 140 tasks and 71,128 methods, integrating real-time and non real-time functionalities with varying Quality of Service (QoS) requirements on a single-core platform.

To address the discussed questions, we used a mixed research methodology utilising discussions within a focus group and subject experts, complemented with a review of the state-of-the-art literature, to identify key concerns and provide a systematic methodology to migrate industrial software with real-time requirements from single-core to multi-core platforms. Concretely, the paper provides the following contributions:

- A systematic methodology for migrating complex embedded software from single-core to multi-core platforms;
- A review of tools that facilitate the migration process; and
- A survey-based evaluation of the proposed methodology.

This paper reinforces the validity of the methodology presented in our previous work [10] by including a survey-based evaluation of the methodology.

The rest of the paper is organised as follows. Section 7.2 provides an overview of a robotic system and its controller software. Section 7.3 reviews the existing software migration methods. Section 7.4 provides an overview of the overall methodology. Section 7.5 includes a systematic approach focusing on architecture migration, followed by implementation and verification of the migration in Section 7.6 and Section 7.7 respectively. A review of the tools facilitating the migration process is discussed in Section 7.8. Section 7.9 presents the evaluation of the proposed methodology. Finally, Section 7.10 concludes the paper.

7.2 System Overview

The system corresponds to a robotic system consisting of a manipulator arm, a controller, and a graphical controller interface. The paper focuses on the software functionality of the controller, which can be divided into functions concerning (i)

[1] `https://new.abb.com/products/robotics/controllers`

configuration, (ii) communication, and (iii) control. The configuration functions provide the robot programming interface that allows a user to configure and specify the runtime behaviour of the manipulator. The user is also able to define the robot environment such as additional sensors and actuators. The communication functions provide a real-time networking capability to enable the controller to interact with devices such as Programmable Logic Controllers (PLCs). It also includes a non-real-time communication capability that allows the controller to interact with enterprise network including PCs and the cloud. The control functions generate the path the manipulator has to follow based on the user-defined configuration. The output of the control functions is used to drive controllers that manage the low-level motor actuation.

The controller software has different runtime modes and the available functions vary between the modes. The main modes include the "Initialisation mode", "Safe-init mode", "System update and configuration mode", "Normal operation mode", and "Fail-safe mode" [11]. The different modes and the transition between the modes is shown in Fig. 7.1. At startup, the controller transitions into the initialisation mode. Here all the tasks are initialised with values based on the previously saved configuration settings. It enters the safe-init mode if there are errors during the startup. The behaviour of the controller software can be configured in the system update and configuration mode. Once the required configuration has been set, the controller enters the normal operation mode. This is the operational mode of the controller, where the physical movement of the robot arm is enabled. It is in this mode that the controller executes the motion planning algorithms with real-time communication enabled for data exchange with external sensors and actuators. It transitions into a fail-safe mode from the normal operation mode if an unexpected error such as an unresponsive sensor, or detection of possible collision with unexpected objects occurs. During normal operation, the user-defined instructions from the robot programming interface provide input to the motion generation components of the software, which in turn generate the path to be followed by the manipulator. Simultaneously, the sensor information and actuator commands are read and written by the communication components based on the user configuration.

Timing related properties of a subset of the tasks that make up the robot controller is provided in the Table. 7.1. There are two tasks, namely TS_Ethercat and TS_RT, that are responsible for real-time communication between the controller and the sensors and actuators. The TS_Ethercat task comprises the network driver, whereas the TS_RT task encapsulates the runtime middleware that provides the necessary interface for data exchange with other components. The two tasks are activated by periodic timers of 10 ms period each and their worst-case execution

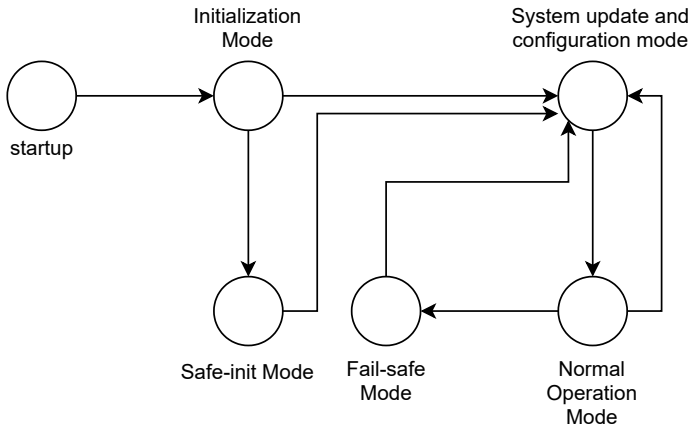

Figure 7.1. Main Modes in the System

times (WCETs) are 120 μs and 80 μs respectively. The priorities of TS_Ethercat and TS_RT are 12 (highest) and 11 (second highest) respectively. Furthermore, the *utilization* of these two tasks are 0.012 and 0.008 respectively. The utilization of a task represents the portion of CPU time required by the task and is calculated by dividing the WCET of the task by its period. The TS_Ethernet, TS_NRT and TS_Web tasks are responsible for non real-time communication such as web-based connectivity for communication with enterprise network and for uploading robot programs and managing and updating the controller configurations. These tasks encapsulate the network drivers, non real-time middleware and web server providing an interface for data exchange between the controller and external devices respectively. The robot program interpretation is performed by the TS_RPI and TS_RPI_Transform tasks. These tasks are responsible for converting the robot program into controller data structures that act as inputs for the trajectory generation functionality of the controller. The TS_RPI task parses the robot program and validates its syntactical correctness. The TS_RPI_Transform task then converts the robot program into a data structure that can be used as input for the trajectory generation functionality, which allows planning of the robot motion and generating the required set-points for the controller task (TS_Control). The trajectory generation functionality is realised with the tasks TS_IPL_Path and TS_IPL_JointPath. Further, the controller software includes the system state manager tasks, namely TS_Sys_Events and TS_Sys_Backup, that are responsible for managing different system level signals and generating events that define the behaviour of other tasks. For example, the system state manager task can observe a change in the state of the safety switch signal and generate an event that will trigger a mode change from

System Functions	Task functionality	Task	Task Trigger Type	Task Priority	Task Period (ms)	WCET (us)	Utilization
RT Comm.	Network driver	TS_Ethercat	timer	12	10	120	0.012
RT Comm.	Network middleware	TS_RT	timer	11	10	80	0.008
Non RT Comm.	Network Driver	TS_Ethernet	timer	5	10	75	0.0075
Non RT Comm.	Network Middleware	TS_NRT	timer	4	50	800	0.016
Non RT Comm.	Application	TS_Web	timer	2	100	200	0.002
Robot Program Interpreter	Parse robot program	TS_RPI	event from TS_NRT	3	50	4000	0.08
Robot Program Interpreter	Format data for trajectory generation	TS_RPI_Transform	event from TS_Sys_Events	6	20	200	0.01
System State Manager	Monitor and handle system state events	TS_Sys_Events	periodic	10	10	60	0.006
System State Manager	Create system backup	TS_Sys_Backup	event from TS_WEB	1	100	200	0.002
Trajectory Generation	Interpolate Cartesian Path	TS_IPL_Path	timer	7	20	2000	0.1
Trajectory Generation	Interpolate Joint Space Path	TS_IPL_JointPath	timer	8	20	200	0.02
Controller	Create setpoints and receive feedback for motor drivers	TS_Control	timer	9	2	100	0.05

Table 7.1. Subset of the tasks in the Robot Controller.

normal operation mode to a fail-safe mode.

7.3 Related Work

Software migration is usually carried out when adopting a different architectural paradigm than the existing one, such as changing the programming language [12] or when moving from native server deployments to cloud-based deployments [13, 14]. Sneed [15] proposed a five-step re-engineering planning process for legacy systems, covering *Project Justification, Portfolio Analysis, Cost estimation, Cost-benefit analysis* and *Contracting*. The author highlights the need for creating measurable metrics to justify the effort and the improvements achievable with the migration. Erraguntla et al. [16] discussed a three phase migration method consisting of analysis, synthesis and transformation phases to migrate single-core to multi-core parallel environments. During the analysis and synthesis phase, the design of the existing software is recovered while recommendations for the multi-core environment are made during the transformation phase of the migration method. They also provided a reverse engineering toolkit called *RETK* for the analysis and synthesis phases. Battaglia [17] presented the *RENAISSANCE* method for re-engineering a legacy system. The method focuses on planning and management of the evolution process.

Menychtas et al. [18] presented a framework called *ARTIST*, a three-phase approach for software modernization focusing on migration towards the cloud. They

Figure 7.2. Proposed migration workflow.

categorised the migration into three main phases, *Pre-migration, Migration and Modernisation* and *Post-migration*. During the pre-migration phase, they proposed a feasibility study to address the technical and economic points of view. During the migration and modernisation phase, the actual migration is carried out and finally during the Post-migration phase, the system is deployed and validated. Forite et al. [19] proposed the *FASMM* approach to better manage the migration and to record and reuse the knowledge gained during the migration in other projects. More recently, Reussner et al. [2] and Wagner [20] proposed model-driven approaches to software migration. The focus in these approaches is to reverse engineer the system using automated tools and capture the information in modelling languages and then use the model-driven approach for further maintenance of the system.

Most of the works discussed so far focused on reverse engineering the existing system to get an understanding of the system, and then to use this information to model and transform the system based on the technical requirements. However, an important aspect we found lacking was emphasis on verification and validation of the reverse engineering processes. Additionally, while many of these works focused on architecture transformation and implementation changes, emphasis on migration of the testing methods was negligible. During our discussions in the focus group, testing was identified as an important domain which required investigation as multi-core architectures are more prone to concurrency issues, e.g., livelock, deadlock, race-conditions and data corruption along with the interference due to the contention for shared resources such as the caches affecting the timing predictability of the overall software system.

7.4 Migration Methodology

Based on the reviewed methods and the extra-functional requirements, we create a migration workflow as depicted in Fig. 7.2 and apply the *A*nalyze, *V*erify, *T*ransform and *Va*lidate approach to this workflow. Essentially, during analysis, the requirements for the migration process are established and the existing system behaviour is recovered. Then the results of the analysis are verified by the subject experts. New solutions are identified and evaluated during the transforma-

tion phase. Finally, the applicability of these solutions, along with the migration process, is validated during the validation phase. Additionally, we consider the migration process to be iterative in the sense that each stage can be revisited and decisions can be roll-backed or modified to address issues that may have been missed or if they do not meet the objective of the migration. A brief overview of the different stages of the proposed workflow is as follows:

1. During the first stage, we focus on the migration of software architecture. In this stage, the goal is to synthesize an abstract system model, validate its accuracy and transform the model for the multi-core environment.

2. In the second stage, the implementation and verification migration, the goal is to analyse the system source code to identify potential concurrency issues within the code and transform the code according to the new multi-core architecture model. Additionally. the existing verification techniques are augmented with methods relevant for a multi-core architecture.

3. In the third stage, we validate the migration process by identifying the validation parameters and measuring these parameters and then comparing them with the values obtained before migration.

7.5 Software Architecture Migration

Many of the real-time systems including the robot controller software have a strong focus on timing, safety and dependability requirements. Therefore, we need a well-defined software architecture to support such requirements. As there are significant differences in the single-core and multi-core platforms, the existing software architecture should be modified to address the constraints of multi-core platforms and make the best use of the available resources. To approach this modification systematically, the software architecture migration stage is divided into five well-defined phases as shown in the Fig. 7.3. The five phases are :

1. Architecture requirements specification;

2. Architecture abstraction and representation;

3. Architecture recovery;

4. Architecture transformation; and

5. Architecture verification.

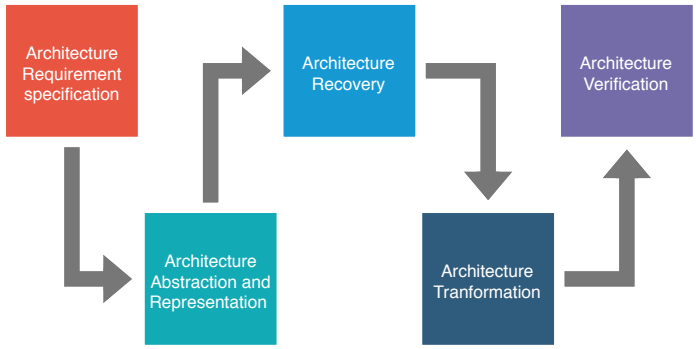

Figure 7.3. Various phases in the software architecture migration.

7.5.1 Architecture Requirements Specification

The architecture requirements specification is the first phase of the architecture migration process. The requirements are essentially high-level and the extra-functional requirements of scalability, performance and timing guarantees are the guiding principles for the complete migration process. The more concrete requirements are defined during the architecture recovery phase of the migration process. We also include the identification of a requirements specification and management process in this phase to better manage the requirements for the rest the migration process.

7.5.2 Architecture Abstraction and Representation

In this phase, we seek to identify an abstraction level that can accurately represent the system behaviour. An abstraction level close to the implementation may be too detailed, while a higher abstraction level can miss critical information that may be necessary for assuring correct system behaviour. Therefore, to identify the right abstraction, we need to identify the system properties that can be affected when moving to the multi-core architectures. Further, a representation model that can sufficiently capture the system properties should be identified. The representation model should be easy to comprehend, and should act as a communication tool between different stakeholders such as the system architects and developers. To address these issues, we rely on expert interviews and the review of state-of-the-art literature related to multi-core in the real-time systems domain and the model-driven engineering domain to guide the selection of the abstraction level and for the identification of the representation tools.

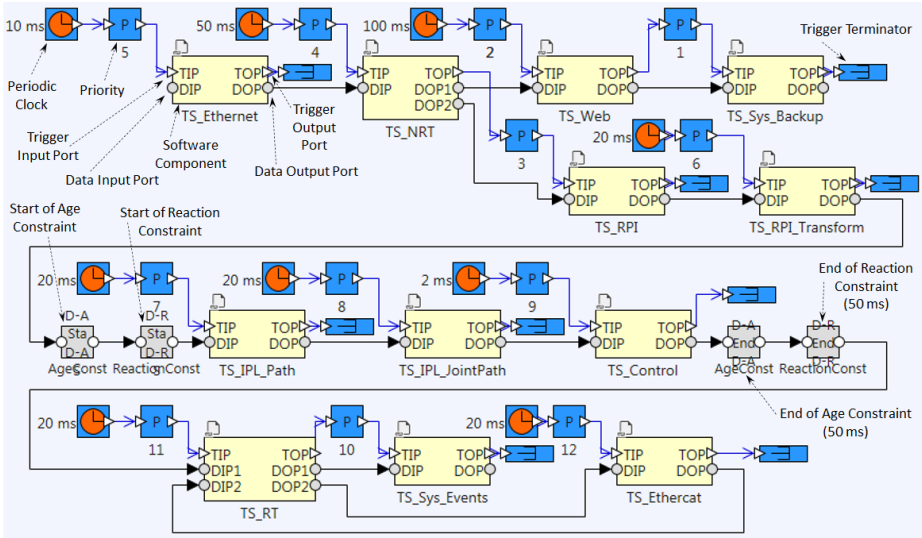

Figure 7.4. Software Architecture Representation.

Software Architecure, Real-time Task Models and Representation Tools The system we considered provides multiple functionalities ranging from embedded control to cloud connectivity. Therefore, we relied on informal and open-ended interviews with the system software architects and domain architects to identify possible abstraction levels. From these discussions, we were able to identify that the task-level abstraction provides the necessary semantics to capture the system properties and therefore, can be used during the later stages of the migration process. Moreover, most of the literature in real-time systems uses the task-level abstraction for the system representation [21, 8].

There are several modelling languages that allow modelling of software architectures and task-level abstraction models of real-time systems. The UML MARTE[2] profile [22], Rubus Component Model [23, 24], UPPAAL [25], MechatronicUML[3] [26], AUTOSAR [27], ART-ML Framework [28], are some of the possible modelling languages and frameworks that can be used to represent the system under discussion.

To demonstrate the software architecture abstraction in the proposed methodology, we model the software architecture of the robot controller using the Rubus Component Model as shown in Fig. 7.4. Note that the Rubus Component Model and its runtime environment consider a one-to-one mapping between a *software*

[2] `https://www.omg.org/omgmarte/`
[3] `http://www.mechatronicuml.org/en/index.html`

component and a task. A software component is the lowest-level hierarchical element in a component model that is used to model the software architecture of a system. The software component is a design-time entity that may correspond to one or more tasks at runtime. For example, the model of a software component that conforms to the Rubus Component Model (RCM) [23, 24] is shown in Fig. 7.5. A software component communicates with other components by means of input and output data and trigger ports. The trigger ports indicate when the task (corresponding to the software component) is activated for execution. A software component can be triggered by an independent source (e.g., a periodic clock) or by another software component. The properties of the software component such as their execution times, activation periods and priorities are specified using the values from Table 7.1. Note that there are two timing constraints, namely Age (50ms) and Reaction (50 ms), that are specified on a chain of software components within the software architecture in Fig. 7.4. These timing constraints conform to the AUTOSAR standard and are supported by several other modelling languages and methodologies for real-time systems [29].

7.5.3 Architecture Recovery

We need to have a better understanding of existing architecture to be able to modify and adapt it to new platforms. However, in many cases,the documented architecture or the intended architecture does not represent the actual implementation. Such deviations can be attributed to multiple reasons. For example, many of the software systems are developed using a top-down development approach. As a result, implementation level changes are not propagated back to the architectural documents resulting in inconsistencies. Recovering the architecture, therefore, is an essential step for the migration. While many useful architecture visualisation tools such as CodeSonar[4] and Imagix[5] analyse the source code to provide architecture visualisation, they only provide information on the logical structure of the software and additionally, they may not be able to detect faulty architectural patterns within the recovered architecture.

Since the transition to multi-core platforms in general affects the timing behaviour of the system, we focus primarily on extracting the temporal properties of the system. For example, a timing requirement can be derived based on the communication between TS_IPL _Path and the TS_IPL _JointPath. Here, one job of the TS_IPL _Path generates data for n jobs of the TS_IPL _JointPath. The next instance of the TS_IPL _Path task should complete its execution before the

[4] `https://www.grammatech.com/products/code-visualization`
[5] `https://www.imagix.com/index.html`

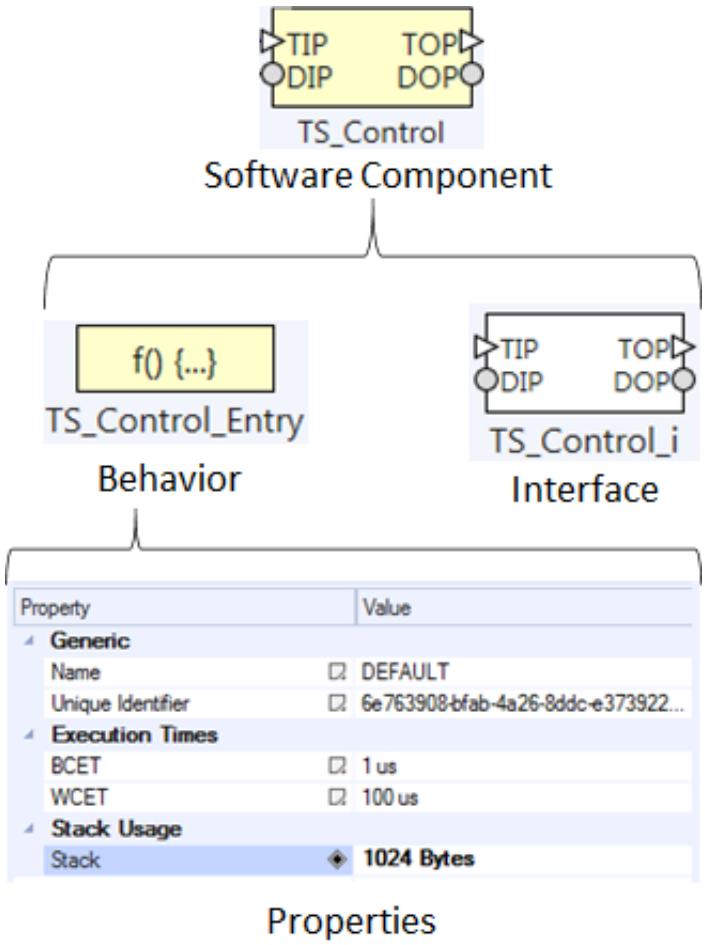

Figure 7.5. Properties of a Software Component.

nth job of the TS_IPL _JointPath is executed. Further, we consider the system to be modelled with cause-effect task chains [30, 31], which implicitly consider maintaining the causality in the underlying communication. These chains are constrained by the timing constraints similar to that of the AUTOSAR standard.

At the task-level abstraction, each task can be represented in terms of its period, worst-case execution time and various types of timing requirements such as deadline, data age, and data reaction constraints [32]. Note that the tasks and their corresponding software components at the software architecture abstraction have the read-execute-write semantics, which allow them to be adapted to comply

with the Logical Execution Time (LET) model [33]. In addition to these, there can be indirect temporal requirements such as the number of messages in a message queue should not be less than a specific value during a certain operating mode, which then requires that the task producing the messages for the queue can be blocked only for a duration that does not violate this requirement. Therefore, we need a comprehensive multi-dimensional software comprehension and reverse engineering approach to extract such information from the existing software architecture, specifically, the timing properties and constraints, which are crucial in verifying timing predictability of the system [32].

To extract the necessary timing requirements, such as the periodicity, execution times and deadlines, we require analysis of multiple data sources. We identified that the architecture documentation, the run-time execution logs and expert validation of the analysis are essential resources for the architecture recovery phase of the migration process, also shown in Fig. 7.6.

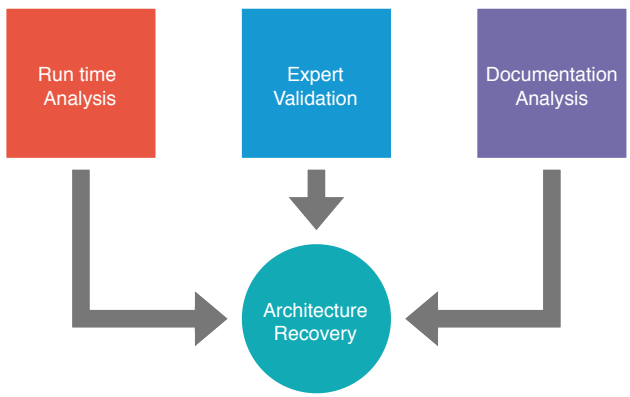

Figure 7.6. Architecture analysis.

Documentation Analysis The architecture of large software intensive systems is normally documented according to the "4+1" architectural view model [34] or an enhanced variant. The format for architecture documentation can vary depending on the internal process and industry-relevant certification requirements. SysML [35] and UML models are some of the formal description formats for documentation used in the industry. Complementing such formal description formats are the textual documents explaining the architecture in natural language as a part of the documentation. These high-level architectural models and documents identify the different components of the system and the interaction between components, summarise the design patterns and technologies employed in the

implementation and provide a concise overview of the functions of these components. By analysing the documentation, it should be possible to identify chains of dependent components, the tasks associated with these components and the expected timing behaviours. The system we considered was documented both in UML models, as well as textual documents.However, during our analysis, we found that there was limited information on expected timing behaviours available in the architecture documents, necessitating other analysis approaches such as run-time analysis and expert validation.

Run-time Analysis While the high-level documents are good sources of information, the information provided by such documentation may either be incomplete or may not reflect the actual implementation. One reason for such an inconsistency is due to the structure of the development process, where the information flow is usually top-down, and the changes made at the implementation level are not propagated back to the architecture documents [36]. Additionally, these industrial software systems have been incrementally developed over many years with the addition of new functionality, bug fixing, and other optimisations in each increment. Therefore, due to the accumulation of undocumented changes made during implementation over the years, relying solely on high-level documentation as the only source of information for modelling the system can result in an inaccurate representation of the expected system behaviour. This makes it necessary to consider the run-time logs as complementary sources of the system information. One approach to understanding the run-time behaviour of the system is the tracing and measurement-based approach [37]. Using this approach, information such as number of context switches, response times, execution times, number of task instances, periodicity of the tasks, among others can be collected. By using dynamic analysis and visualisation tools such as Tracealyzer [37], additional information such as the communication flow between different tasks, identification of shared resources, task chains and precedence constraints between the tasks can be obtained. The information gained from the run-time analysis can be used to refine and enhance the model.

The run-time analysis comes with its own set of conundrums. As the system under consideration is configurable, i.e., the user can configure and specify the runtime behaviour, it is difficult to identify a configuration that can be a single representative of possible configurations for run-time analysis.

One possible approach to address this issue is to use the “maximum load” approach. We consider the system to be in “maximum load” state, if under normal operation mode, all system tasks are active and that each task is executing its most computationally heavy or memory intensive jobs. Relying on a single configuration,

however, is not sufficient to make any statistically reliable conclusions about the measurements. Therefore, another argument would be to gather run-time behaviour from as many possible configurations as feasible. Again, identifying this "feasible" number is not straight forward. This is made even more complicated by the continuous development process, where code is modified and new builds generated daily. Identifying a fixed version of the software for analysis becomes non-trivial for such cases. Further, since the controller software operates under different modes, the "maximum load" approach could be pessimistic. Depending on the system under migration, we will need to identify an appropriate configuration and analyse the run time behaviour of each mode independently. For the controller software considered, the "normal operation mode" had the highest resource demand and since all the other modes run only a subset of the "normal operation mode" tasks, we use the maximum load configuration of the "normal operation mode"and ensure that all the required system software components are active during the trace period. Note that we rely on the latest released version of the software.

During the run-time analysis of our system, we found that there were inconsistencies between the expected and observed behaviours. A few of the inconsistencies were a result of incorrect configuration of the instrumented code, while others were actual deviations from the expected behaviour. For example, the incorrect configuration resulted in the trace logs showing multiple instances of the jobs of a task as a single job of the same task. This observation highlights the fact that relying on a single source for information is not only ineffective but also error-prone. This necessitates the need for expert validation of the collected information to create a sufficiently accurate system model.

Expert Validation Architectural design decisions are made by analys-ing multiple factors such as domain requirements, dependencies on services provided by the operating systems and the underlying hardware platform, among others. However, the high-level architectural models and documents do not describe the rationale behind the design decisions and even if they do, such information is limited. Moreover, in legacy systems, such documents do not completely reflect the implementation [36]. Furthermore, as the information from the run-time analysis is quantitative and statistical in nature, it is possible to misinterpret any deviation from a commonly occurring pattern as an inconsistency whereas this could have been a design decision. To avoid such misinterpretations and improve system model accuracy, discussions with domain experts are mandatory during the architecture analysis. These discussions will be used to understand the rationale behind the design decisions, and to validate the observations of the documentation and the run-time analysis phases. In our work, we were able to validate the inconsistencies

such as the deviation from a commonly occurring pattern as a design decision and also mark some of the observed results as an outcome of incorrect code instrumentation configuration. For example, due to incorrect configuration of the code instrumentation library, the periodicity of the TS_RPI observed during run-time analysis phase did not match the values expected by the experts. the functional behaviour however, was accurate, prompting a separate analysis. This analysis identified incorrect configuration of the code instrumentation as the root cause for observed deviation in the periodicity.

7.5.4 Architecture Transformation

As discussed earlier, the architecture transformation phase focuses primarily on evaluating potential solutions and identifying the most appropriate ones for the final implementation. Before we evaluate any solution, we need to identify the system requirements that need to be considered to identify, evaluate and qualitatively rank possible solutions. Since in our case, the migration to multi-core will primarily affect the runtime behaviour, we focus on the explicit temporal requirements, implicit requirements such as the number of messages in a queue and assigned QoS levels to different functional domains. An important requirement here is to ensure that this transformation results in improved system predictability, performance and that the architecture is scalable in terms of the number of cores and new functionality that needs to be integrated into future versions of the software. Since the terms predictability, performance, and scalability are generic in nature, we need to ensure that we have measurable definitions for these terms. For example, we use scalability to refer to the capability of the controller software to control more than one manipulator on the same hardware platform. Once we define the evaluation criteria, we then move towards the evaluation process itself. The evaluation can be carried out in various ways depending on the evaluation metric and the solution being considered, such as simulation, model-checking and analytical calculations. Once the evaluation of possible solutions is complete, we rank these solutions based on an agreed evaluation metric and based on these rankings, we select the solutions for the final implementation phase. To ensure that this transformation is systematic, we divide the transformation phase into the following steps:

1. identification of potential solutions;
2. evaluation of the solutions;
3. ranking of the solutions;
4. selection of the solutions.

Identification of potential solutions Identification of potential solutions can be done in many different ways. Although we don't make any specific recommendations, we would like to point out that the number of potential solutions could be infinitely many and we hypothesize that evaluating each solution will be impossible. Especially in the case of real-time systems, where the search space in terms of near-optimal solutions is large [8, 9, 38, 39]. Therefore, a good starting point in this stage are the domain experts. Also, the information from the architecture abstraction and recovery phases can be a useful guide in reducing the search space. In our case, we use expert interviews and review the state-of-art in the real-time systems domain to identify potential solutions. Another important consideration is that since application developers are focused primarily on the application functionality, they rely on the operating systems to provide support for real-time properties. This implies that in many cases, only those mechanisms supported by an operating system can be considered as part of the potential solution set.

As highlighted earlier, the purpose of an abstract system model is to capture all the relevant properties of the system but without the functional complexity. This enables creation of synthetic tasks for simulation and verification of new design solutions. These abstract task sets can be modified and verified in short time spans when compared to modification of the actual implementation of the system. Many of the real-time workload models such as those reviewed in [21] have been successfully used to represent practical systems such as in the avionics domain as well as in the automotive domain. While many of these workload models consider the tasks to be independent, we found that the system under study violates this assumption and that new jobs of tasks are triggered by jobs of other tasks. Also, the presence of event triggered components within the system along with multi-rate task chains implementing a single functionality, requires that the precedence constraints as well as task chains be considered when considering potential solutions [30].

Some of the relevant issues that should be addressed by the potential solutions for transitioning from single core to multi-core platforms were highlighted by Macher et al. [40], and Nemati et al. [41]. For example, use of single-core hardware implies that the system tasks execute in sequential manner. If run on multi-core, the task precedence constraints may not be maintained affecting system dependability. Additionally, systems designed for single-core do not require any mapping of software and multiple compute resources. However, predictable execution on multi-core is provided by partitioned scheduling approaches [39]. Ad hoc partitioning can affect system performance and scalability. Multi-level caching can cause data inconsistencies when tasks sharing a variable are executing on different cores [42]. In the case of fixed-priority scheduling, priority assignment can impact response

times [38].

Evaluation of the solutions Once the potential solutions have been identified, the next step is to evaluate these solutions. By evaluation, we refer to the application of the potential solutions from the previous step to the abstract model from the architecture recovery stage and measurement of the identified metrics. The evaluation can be done in different ways as already highlighted earlier such as simulation in the case of ART-ML framework [28] or the Cheddar tool [43], analytical calculations if using techniques such as those identified in [39], or model-checking if using the timed automata approach specified in [44]. For the system described in Section 7.2, one strategy could be to allocate the parts of the system that are constrained by the timing constraints to one core and rest of the software components to other cores (e.g., TS_IPL_Path, TS_IPL_JointPath, and TS_Control to one core and the rest of the components to the other core(s)). Another strategy could be to allocate the software components to the cores such that the specified age and reaction delays are minimized. Another strategy could be based on precedence constraints between the software components, which should be on the same core (e.g., TS_Web and TS_Sys_Backup have an implicit precedence constraint as the latter is triggered by the former, hence both should be on the same core). Similarly, another allocation strategy could be based on the criticality levels associated to the software components so that non safety-critical software cannot interfere with the safety-critical software as proposed in [45]. We would like to point out that given the safety-critical nature and complexity of the system, we hypothesise that the potential solution identification and evaluation steps are rather time consuming and are critical in the migration process. The time spent during these phases can potentially result in practical solutions that ensure that the migration process is successful in meeting the extra-functional requirements.

Moving forward, we return to the question of identifying the best solution among the many evaluated solutions. To guide in this direction, we use the ranking approach as follows.

Ranking of the solutions The ranking step of the transformation phase orders the evaluated solutions in terms of certain criteria. For example, the evaluated solution may be required to adhere to safety and security requirements of the domain. Further it may be possible that the extra-functional properties such as portability between different hardware platforms may be prioritised over performance improvement on a single hardware device. To address such requirements in a systematic manner, we propose to use the following multi-step approach:

- identify parameters to rank potential solutions;

- provide measurable definitions to the identified parameters;
- arrive at a consensus on measurement methods for the parameters;
- prioritize or assign weights to the parameters for trade-off analysis;
- rank the evaluated solutions.

We believe that this approach provides a systematic way to measure effectiveness of the evaluated solutions and guide in selection of the final solution. By identifying measurable parameters, the methods to measure them, and prioritize them if a trade-off is necessary, we can remove any ambiguity associated with the perceived effectiveness. To identify these parameters, we propose focus group discussions involving the different domain experts.

Selection of the solutions Once the potential solutions have been evaluated and ranked, the selection of final solutions should be rather straight forward. However we would like to point out the fact that there could be solutions that may optimize one requirement while negatively affecting another requiring a trade-off analysis to select a final solution.

Architecture Verification

The last step in the architecture transformation phase is the verification of the transformed architecture. Here we essentially verify if the transformed architecture complies with requirements from the architecture requirements specification phase and the recovery phase. The verification stage is rather simple and straight forward since the different steps in the transformation phase involve verification in the evaluation stage with the systematic ranking and selection approach.

7.6 Implementation Migration

So far, we discussed the transformation at the architecture level of the system in our migration process. We now discuss the processes necessary to implement the transformed architecture at the source code level. Although not directly related to the migration process itself, we consider that some form of refactoring at the source-code level may be necessary prior to the migration process. Depending on the existing logical architecture and the quality of the software, the refactoring may address different concerns. For example, removal of duplicate and dead code, creating components based on functionality, adoption of a layered architecture

among others. For further discussion, we assume that the system has a layered architecture with well-defined components, that the logical architecture is capable of handling new components and modifications in the abstraction layers, and that the source code is separated according to the components.

Further, we classify the architecture solutions as abstract component level or functional component level solutions. For example, if the solution is a new priority order for the tasks, then it is functional component level solution if the tasks are associated with the component and that the priorities can only be changed in the component files. If it is a new synchronisation protocol, then it is an abstract level solution, which is used by all components and may need a new implementation. Therefore, before we make the changes, we identify components that need to be modified, map solutions that need new components and then implement the changes.

7.6.1 Component Identification and Creation

The solutions selected during the transformation phase may require that changes be made to the existing components in the system. For example, if the components use nested semaphores and if the identified solution does not support nested semaphores, then such nested semaphores need to be removed. To do this in a systematic manner, we index and categorise the transformed solutions, review the solutions with the domain experts and component owners and associate each component with the solution that requires that component to be modified. For example, the trajectory generation component may require that its source code be modified to accommodate the changes necessary to migrate to multi-core platform. We then review the solution with the owners of the trajectory generation component. Further, if there are solutions that are classified as abstract-level solutions or which could not be mapped to existing components, we create new components for such changes. For example, if a new real-time middleware, that will provide a common inter-task communication mechanism is to be implemented, then a new component will be created.

7.6.2 Implementation

Once all components have been identified for modification and new components created, the necessary changes are implemented in the source code. Although the concurrency related issues are addressed during the architecture transformation phase, it is possible that they could manifest during the implementation stage. Therefore, coding guidelines that address these issues are provided to the developers to minimise the manifestation of these issues during the implementation.

7.7 Verification Migration

The system verification and validation stage is the final stage of the migration process. Typically, for the system such as the one being considered, a reliable verification process is already in place. This includes the usual verification approaches such as unit testing, functional testing, and system integration tests. Since the architectural transformation is primarily related to the runtime behaviour and performance, we expect that most, if not all existing tests related to functional behaviour to be valid. Therefore, we hypothesise that any failures here could be related to the concurrent execution of the system tasks. To maintain the quality of the system software, we focus on augmenting the existing tests with concurrency related testing approaches along with performance verification. Again, to approach this enhancement in a systematic way, we divide the verification migration process into concurrency testing and the migration validation phase.

7.7.1 Concurrency Testing

The goal during this phase is to augment the existing verification process to identify concurrency related issues. These include race conditions, atomicity violations and deadlocks. A comprehensive review can be found in the work by Bianchi et al. [46]. We propose the analysis of solutions during the architecture transformation phase to identify scenarios that could lead to potential concurrency issues. This way, it will be possible to create tests for those specific scenarios. Additionally, static code analysis that identifies concurrency bugs is added to enhance the verification process.

7.7.2 Migration Validation

During this phase, we focus on validation of the migration process itself. We begin by identifying the parameters to qualitatively validate the outcome of the process. We use two metrics for this purpose: (i) results of the functional and system integration tests, and (ii) performance related parameters such as response times. In the first case, no new failures should be introduced after the migration. In the second, the values of the performance parameters should not be less than those measured with the pre-migration version. We point out here that although the validation is the last step, depending on the development process, this validation can be applied to each build prior to release. By using the results of the validation with each build, the pace of the migration process can be measured.

7.8 Tools for Migration

Software migration from single-core to multi-core architectures is a complex process and requires the use of different tools at different stages of the migration process. Here, we review some of the tools that can be used during the different phases of the migration process.

7.8.1 Architecture Representation

Software requirements and the architecture can be described in natural language and as models using different modelling languages such as the UML. For embedded systems with timing requirements, there exist many tools that allow modelling and specification of different views of the system. The APP4MC tool[6], allows modelling and specification of the hardware as well as software components and provides support for scheduling algorithms. Another tool is the MARTE [47] profile for UML. The MARTE profile extends the UML models to include description of timing requirements. The MAST tool-suite[7] allows for modelling as well as performing automatic schedulabilty analysis and supports many of the common scheduling algorithms for single-core as well as multi-core architectures. UPPAAL [25] is another tool for modelling the software as timed-automata and it supports model checking for formal analysis and verification. A few concerns with many of these tools are that some have steep learning curves, while others such as UPPAAL are not scalable to large systems and almost all lack support for automatic conversion of existing source code to abstract models.

7.8.2 Architecture Recovery

For architecture recovery, static code visualization tools such as CodeSonar and Imagix could be used. For dynamic analysis, tools which provide visualization of the run-time behaviour along with statistical information on timing properties can be effective. For example, Tracelyzer allows visualization of the run-time behaviour and provides different views to analyse this information.

7.9 Evaluation

We chose a survey-based approach to evaluate the proposed methodology. We followed the guidelines provided by Kitchenham et al. [48] for survey-based

[6] `https://www.eclipse.org/app4mc/`
[7] `https://mast.unican.es/`

research and the discussion of the results. We begin by describing the design of the survey and then discuss the results of the survey.

7.9.1 Survey Design

As a first step in the survey-based evaluation, we identified (i) feasibility, (ii) usability and, (iii) usefulness as the evaluation objectives for the migration methodology. Next, we identified the target population for the evaluation to be those organisations that develop complex real-time software systems such as industrial automation systems and construction vehicles. We identified a sample from the target population in a non-probabilistic manner through convenience and judgement based sampling. We created the survey instrument in the form of online questionnaire that included both close and open ended questions. The close ended questions were designed to verify the generalisation of the observations and the applicability of the different steps in the methodology. The open ended questions required the respondents to provide their opinion in a textual format on feasibility and usefulness of the methodology. The complete questionnaire was piloted by requesting colleagues not involved in the study to ensure clarity of language before it was shared with the respondents. The questionnaire was made available digitally and included a brief overview of the purpose of the questionnaire. The respondents were requested to read about the presented methodology before they answered the survey. The received responses were then analysed to evaluate the methodology.

Evaluation Objectives

As previously mentioned, we identified three key objectives for the evaluation, namely feasibility, usability and usefulness of the methodology. For each of these objectives, we adopt the definitions used by Adesola et al. [49] to evaluate their business improvement process methodology. Briefly, we use *feasibility* to imply that all the steps in the methodology can be followed in practice. We use the term *usability* to refer to the ease of applicability of the methodology steps and the tools mentioned therein. We use *usefulness* to refer to the outcome of applying the methodology to relevant systems by an organisation. Furthermore, we also included the objective of validating the possibility of generalising key observations in the methodology.

Target Population and Sampling Strategy

To address the evaluation objectives, the target population was identified as organisations developing complex real-time systems. As for the sample, we identified

2 different departments within the same organisation working on independent and unrelated products and also two other organisations. We then identified 9 expert practitioners from the sample group as the most relevant for the evaluation. The participants were chosen based on their experience in managing and developing software(10+ years) for industrial systems and for background in multi-core technologies and their knowledge of the application domains.

Table 7.2. Mapping among the different steps of the methodology, the evaluation type for each of the step and the associated question IDs.

Methodology Stage (Step)	**Evaluation Type/ No. Of Questions**	**Question ID**
Architecture Abstraction and Representation (General)	Explicit : 1 Implicit : 6	11 27-32
Architecture Abstraction and Representation (Expert Interviews)	Implicit : 6	27-32
Architecture Abstraction and Representation (State-of-art in Real-time Systems)	Implicit : 6	27-32
Architecture Abstraction and Representation (State-of-art in Model-Driven Engineering)	Implicit : 6	27-32
Architecture recovery (Documentation Analysis)	Explicit : 3 Implicit : 6	13-15 27-32
Architecture recovery (Runtime Analysis)	Explicit : 7 Implicit : 6	12, 16- 21 27-32
Architecture Recovery (Expert Validation)	Implicit : 6	27-32
Architecture Transformation (Identification of Potential Solutions)	Implicit : 6	27-32
Architecture Transformation (Evaluation of the Solutions)	Implicit : 6	27-32
Architecture Transformation (Ranking of the Solutions)	Explicit : 3 Implicit : 6	22- 24 27-32
Architecture Transformation (Selection of the solutions)	Implicit : 6	27-32
Architecture Verification	Implicit : 6	27-32
Implementation Migration (Component Identification and Creation)	Implicit : 6	27-32
Implementation Migration (Implementation)	Implicit : 6	27-32
Verification Migration (Concurrency Testing)	Implicit : 6	27-32
Verification Migration (Migration Validation)	Explicit : 2 Implicit : 6	25-26 27-32
Tools for Migration (Architecture Representation)	Implicit : 6	27-32
Tools for Migration (Architecture Recovery)	Implicit : 6	27-32

Instrument Design

The survey was designed in the form of a questionnaire, combining nominal, close-ended questions, and the open-ended questions requiring textual input from the respondents. The questionnaire was designed to address two different aspects, (i) problem relevance and (ii) methodology evaluation. For the problem relevance, we developed six questions to verify if the respondents were considering multi-core platforms for their products. The rest of the questionnaire was focused on methodology evaluation. We classified the evaluation related questions as either implicit or explicit. The implicit questions required the respondents to reflect on

the overall feasibility, usability and usefulness of the methodology. The explicit questions were designed to validate the generalisation of some of the observations made in the methodology. Table 7.2 shows the mapping among the different steps of the methodology, the evaluation type for each of the step and the associated question IDs. Appendix 8.1 shows the questionnaire.

7.9.2 Survey Results and Discussion

As mentioned previously, the questionnaire was shared with nine carefully identified participants from the sample population. Of the nine participants invited, five respondents participated in the survey. We use the labels A,B,C,D and E to refer to each of the respondent individually. We discuss the results for the objectives of problem relevance, generalisation, overall feasibility, overall usability and the overall usefulness.

Problem Relevance From the problem relevance perspective, 4 of the 5 the respondents, (A,B,C and E) said that their applications were not designed for multi-core. Respondent *D* said that their applications were designed for multi-core but they have been developed from the scratch with only limited reuse of existing code. Respondents *C* and *E* confirmed that they are planning to migrate to a multi-core platform while the rest of the respondents did not provide any information. Additionally, the same four respondents chose the option of redesigning the application while reusing the existing code over developing the application from scratch. The responses indicate that migration to multi-core platforms is being considered in the industry and at the same time, the respondents prefer reusing the existing code over the development of the applications from scratch.

Generalisation and Feasibility Since the methodology was developed based on observations of one system, we created the questionnaire to verify if the observations made in different steps can be generalised for other complex real-time software systems as well. This was done by asking directed nominal questions focused on architecture representation, architecture recovery (runtime analysis and documentation), architecture transformation (ranking of solutions), and verification migration. For the architecture representation, the results indicate that only parts of the application can be described by timing properties such as worst-case execution times, periods and deadlines.

Similar to the observations about lack of information in the documentation, 4 of the 5 the respondents, (A,B,C and E) said that the application design was not fully documented. Further, only one respondent said that the timing properties

were discussed in the design documentation while the rest of the respondents said that the timing properties of only a few critical parts of the application were discussed in the documentation.

The methodology relies on the presence of diagnostic information such as execution times and periodicity for architecture recovery. All the respondents said that their systems provide such diagnostic information. Furthermore, all the respondents mentioned that their applications had multiple configurations and that the runtime behaviour depended on the configuration. None of the respondents said that they tested all possible configurations but only a few. Four out of five respondents (A, B,C and D) said they tested average-case configurations. Furthermore, respondents A and E said that they test the worst-case configurations while respondent D said that they test the best-case, average-case as well as the worst-case configurations. This indicates that identifying a representative configuration for architecture is not straight forward and can depend on individual application requirements.

Evaluation and ranking of solutions is an important step in the methodology. Here we assumed that it will be possible to identify and provide measurable metrics for ranking possible multi-core solutions. To verify if the assumptions are valid, the respondents were explicitly asked if they can provide measurable parameters and also prioritise them. Four out of five respondents (A, B D and E) agreed that they can define as well as prioritise, while respondent C answered negatively.

For the verification migration stage of the methodology, a key assumption is that the complex real-time systems such as the one discussed in this paper have a robust testing mechanism in place for verifying functional correctness. All the respondents agreed that they do have such a mechanism in place. Further, all respondents agreed that they will reuse the existing tests to verify the behaviour of the systems after migration, which is consistent with the assumptions made in the proposed methodology.

The results of the questionnaire so far indicate that much of the observations can be generalised to other complex real-time systems. One key observation however, is that describing all of the application components with timing properties may not be possible. For the steps not discussed in generalisation, we address them from the overall feasibility perspective discussed next.

Overall Feasibility In order to validate the feasibility of the methodology, i.e., to verify if all the steps of the methodology can be followed, the respondents were asked to answer if they found the methodology feasible and to describe the rationale behind their choice. Four out of five respondents (A B D and E) considered the methodology to be feasible while respondent C considered otherwise. When

describing the rationale, respondent C said that they needed more information and the correct answer would actually be that they are not sure. Respondents B and E did not explain the rationale. Respondent A and D agreed that it is possible to represent the architecture at a feasible abstraction level and that the methodology covered all the critical steps. One concern however was that the industrial applications are rather big, and therefore we need to address the migration in parts and avoid a "big bang" approach.

Overall Usability The survey also included questions to evaluate the overall usability of the methodology, i.e., to verify if the steps in the methodology are workable and are easy to apply in practice. Similar to the question of feasibility, four out of five respondents (A B D and E) answered positively while respondent C said no. When describing the rationale, respondent C said that their correct answer would actually be that they are not sure. Respondent A and B said that the transformation phase was uncertain but the steps are general enough to be followed and that the difficulty in following the steps may depend on the "architecture, requirements and availability of tools". Similar response was provided by respondent D who said that the level of modelling may vary depending on the company. Based on the responses it can be observed that the steps in the proposed methodology can be followed in general but the overall usability is dependent on individual applications.

Overall Usefulness Another objective of the evaluation is to assess overall usefulness of the methodology for the target population. To address this, the respondents were asked to evaluate "Usefulness: if the methodology can produce results that the organisation will find useful?". Two out of five respondents (A and B) consider the methodology to be useful for the industry, whereas the remaining three respondents consider the methodology to be "partially" useful. Respondent B justified their choice by highlighting the general applicability of the steps and respondent A said that having such a methodology will create a "common understanding" between the different stakeholders and the developers, thus *increasing the possibility of success and decreasing risks*. Respondent C said the it may not be possible to follow the steps completely, but the ideas can be "useful". A similar observation was made by respondent D who said it will be necessary to consider the product to see if the methodology fits the product being considered for migration. Although it is not possible to draw a straight forward conclusion about the usefulness of the methodology, we can observe from the responses that having a methodology can reduce the risks of migration projects but the methodology will have to be adapted to suit individual application needs in the industry.

Discussion The proposed methodology was evaluated for feasibility, usability and usefulness by expert practitioners via a questionnaire. From the feasibility perspective, the analysis of the questionnaire responses indicate that the methodology covers the critical steps necessary for a software migration. From the usability perspective, the analysis of the responses shows that the different steps can be applied in practice but depending on the application, the abstraction level and the modelling requirements will depend on individual applications. From the usefulness perspective, the responses show that following the methodology steps can decrease the risks associated with the migration. From the Generalisation perspective, the response show that the observations made in the methodology can be extended to systems other than the robotic system considered, while highlighting the fact that it may not always be possible to describe the timing properties for all of the application components.

Threats to Validity Since the evaluation of the methodology has been carried out using a survey, we include a discussion on the validity of the results. Kitchenham et al. [48] advocates that a survey is reliable if it has been administered multiple times and if we get similar results each time. In our case, the survey was administered only once. This implies that the results may vary if the respondents were to answer questionnaire at different times. However, much of the questionnaire had nominal questions and the number of options provided were binary but with an additional option to provide textual information thereby limiting the possibility of variability in the responses. Furthermore, although the sample group was carefully chosen in a non-probabilistic manner, it is possible that a different sample of respondents may have provided different responses, affecting the validity of the conclusions drawn from the survey results. While the survey included questions relating to generalisation of the observations, not all of the methodology steps were explicitly considered but were included under the general questions of overall feasibility, usability and usefulness. Explicit questions may have lead to a different conclusion from the one discussed in the paper.

7.10 Conclusion

Migration of complex embedded software from single-core to multi-core computing platforms is non-trivial. To ensure a successful migration of these software systems, a systematic approach is needed that takes multiple software engineering perspectives into account such as software processes, software architectures, requirements engineering, reverse engineering, model-based development, real-time scheduling and schedulability analysis. In this paper, we presented a systematic

multi-stage methodology for migrating real-time industrial software systems from single-core to multi-core computing platforms. In this regard, we studied a complex real-time software system from the automation industrial domain that requires such a migration. We used focus group discussions, expert interviews and reviewed the literature to guide the development of the migration strategy. We identified the software architecture transformation as the main phase in the migration process and presented a systematic approach to perform the transformation with emphasis on the architecture recovery and an evaluation mechanism for possible multi-core solutions. We used task-level abstraction of the system to drive the transformation and associated timing properties to task-level models and proposed their use as input for the evaluation of multi-core solutions. To select suitable solutions from the set of evaluated approaches we proposed ranking of these solutions based on measurable parameters for the final implementation and we reviewed some of the tools that can be used during the migration process. We evaluated feasibility, usability and usefulness of the methodology using a survey-based approach. Majority of the respondents agreed that the methodology is feasible, usable and useful in general for the industrial applications. The evaluation also revealed that the methodology will have to be individually adapted to each system under migration.

Acknowledgements

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation, the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the projects DESTINE and PROVIDENT, and the Swedish Knowledge Foundation (KKS) through the projects HERO, FIESTA and DPAC. We would also like to thank our industrial partners for providing valuable feedback via the survey.

Bibliography

[1] Ned Chapin, Joanne E. Hale, Khaled Md. Kham, Juan F. Ramil, and Wui-Gee Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance*, 13(1):3–30, January 2001.

[2] Ralf Reussner, Michael Goedicke Wilhelm Hasselbring, Birgit Vogel-Heuser, Jan Keim, Lukas Märtin, editor. *Managed Software Evolution*. Springer Nature Switzerland AG, 2019.

[3] Johan Kraft, Yue Lu, Christer Norström, and Anders Wall. A Metaheuristic Approach for Best Effort Timing Analysis Targeting Complex Legacy Real-Time Systems. In *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 258–269.

[4] Goran Mustapić, Johan Andersson, Christer Norström, and Anders Wall. A Dependable Open Platform for Industrial Robotics – A Case Study. In Rogério de Lemos, Cristina Gacek, and Alexander Romanovsky, editors, *Architecting Dependable Systems II*, pages 307–329. Springer Berlin Heidelberg, 2004.

[5] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1), Jan 2004.

[6] G. Mustapic, A. Wall, C. Norstrom, I. Crnkovic, K. Sandstrom, J. Froberg, and J. Andersson. Real world influences on software architecture - interviews with industrial system experts. In *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture* , 2004.

[7] Saad Mubeen, Elena Lisova, and Aneta Vulgarakis Feljan. Timing predictability and security in safety-critical industrial cyber-physical systems: A position paper. *Applied Sciences*, 10(9):3125, 2020.

[8] Claire Maiza, Hamza Rihani, Juan M. Rivas, Joël Goossens, Sebastian Altmeyer, and Robert I. Davis. A survey of timing verification techniques for multi-core real-time systems. *ACM Comput. Surv.*, 52(3):56:1–56:38, 2019.

[9] Robert I. Davis and Liliana Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for real-time systems. *LITES*, 6(1):04:1–04:53, 2019.

[10] Shaik Mohammed Salman, Alessandro Vittorio Papadopoulos, Saad Mubeen, and Thomas Nolte. A systematic migration methodology for complex real-time software systems. In *2020 IEEE 23rd IEEE International Symposium on Real-Time Distributed Computing*, pages 192–200.

[11] Goran Mustapić, Johan Andersson, Christer Norström, and Anders Wall. A dependable open platform for industrial robotics – a case study. In Rogério de Lemos, Cristina Gacek, and Alexander Romanovsky, editors, *Architecting Dependable Systems II*, pages 307–329, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[12] Leszek Wlodarski, Boris Pereira, Ivan Povazan, Johan Fabry, and Vadim Zaytsev. Qualify First! A Large Scale Modernisation Report. In *SANER*, pages 569–573. IEEE, 2019.

[13] Philip Church, Harald Mueller, Caspar Ryan, Spyridon V. Gogouvitis, Andrzej Goscinski, and Zahir Tari. Migration of a SCADA system to IaaS clouds – a case study. *Journal of Cloud Computing*, 6(1):256, 2017.

[14] Konstantinos Plakidas, Daniel Schall, and Uwe Zdun. Software Migration and Architecture Evolution with Industrial Platforms: A Multi-case Study. In Carlos E. Cuesta, David Garlan, and Jennifer Pérez, editors, *Software Architecture*, volume 11048 of *Lecture Notes in Computer Science*, pages 336–343. Springer International Publishing, Cham, 2018.

[15] H. M. Sneed. Planning the reengineering of legacy systems. *IEEE Software*, 12(1):24–34, 1995.

[16] Ravi Erraguntla and Doris L. Carver. Migration of sequential systems to parallel environments by reverse engineering. *Information & Software Technology*, 40(7):369–380, 1998.

[17] M. Battaglia, G. Savoia, and J. Favaro. Renaissance: a method to migrate from legacy to immortal software systems. In *Proceedings of the Second*

Euromicro Conference on Software Maintenance and Reengineering, pages 197–200, 1998.

[18] Andreas Menychtas, Kleopatra Konstanteli, Juncal Alonso, Leire Orue-Echevarria, Jesus Gorronogoitia, George Kousiouris, Christina Santzaridou, Hugo Bruneliere, Bram Pellens, Peter Stuer, Oliver Strauss, Tatiana Senkova, and Theodora Varvarigou. Software modernization and cloudification using the ARTIST migration methodology and framework. *Scalable Computing: Practice and Experience*, 15(2), 2014.

[19] Louis Forite and Charlotte Hug. FASMM: Fast and Accessible Software Migration Method. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science*, pages 1–12. IEEE, 2014.

[20] Christian Wagner. *Model-Driven Software Migration: A Methodology*. Springer Fachmedien Wiesbaden, Wiesbaden, 2014.

[21] Martin Stigge and Wang Yi. Graph-based models for real-time workload: a survey. *Real-Time Systems*, 51(5):602–636, 2015.

[22] F.Herrera, H. Posadas, P.Peñil, E.Villar, F.Ferrero, R.Valencia, and G.Palermo. The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems. *Journal of Systems Architecture*, 60(1):55–78, 2014.

[23] Kaj Hänninen, Jukka Mäki-Turja, Mikael Sjödin, Mats Lindberg, John Lundbäck, and Kurt-Lennart Lundbäck. The Rubus Component Model for Resource Constrained Real-Time Systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, 2011.

[24] S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, and K. L. Lundbäck. Provisioning of predictable embedded software in the vehicle industry: The rubus approach. In *IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, 2017.

[25] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2):134–152, December 1997.

[26] Wilhelm Schäfer and Heike Wehrheim. *Model-Driven Development with Mechatronic UML*, pages 533–554. Springer Berlin Heidelberg, 2010.

[27] The AUTOSAR Consortium. Autosar technical overview. In *Version 4.3.*, May 2016. `http://autosar.org`.

[28] Anders Wall. *Architectural Modeling and Analysis of Complex RealTime Systems*. PhD thesis, Mälardalen University, Västerås Sweden, 2003.

[29] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara. Recent advances and trends in on-board embedded and networked automotive systems. *IEEE Transactions on Industrial Informatics*, 2019.

[30] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture*, 80:104 – 113, 2017.

[31] M. Becker, S. Mubeen, D. Dasari, M. Behnam, and T. Nolte. A generic framework facilitating early analysis of data propagation delays in multi-rate systems (invited paper). In *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–11, 2017.

[32] Saad Mubeen, Thomas Nolte, Mikael Sjödin, John Lundbäck, and Kurt-Lennart Lundbäck. Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints. *Software & Systems Modeling*, 18(1):39–69, Feb 2019.

[33] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003.

[34] P. B. Kruchten. The 4+1 View Model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[35] E. Andrianarison and J.D. Piques. SysML for embedded automotive systems: a practical approach. In *Conference on Embedded Real Time Software and Systems*. IEEE, 2010.

[36] Ulf Eliasson, Rogardt Heldal, Patrizio Pelliccione, and Jonn Lantz. Architecting in the Automotive Domain: Descriptive vs Prescriptive Architecture. In *12th Working IEEE/IFIP Conference on Software Architecture, 2015*.

[37] Johan Kraft, Anders Wall, and Holger Kienle. Trace recording for embedded systems: Lessons learned from five industrial projects. In *Proceedings of the First International Conference on Runtime Verification (RV 2010)*. Springer-Verlag (Lecture Notes in Computer Science), November 2010.

[38] Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. A review of priority assignment in real-time systems. *Journal of Systems Architecture*, 65:64–82, 2016.

[39] Bjorn B. Brandenburg and Mahircan Gul. Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations. In *IEEE Real-Time Systems Symposium, 2016*.

[40] Georg Franz Heinrich Macher, Andrea Höller, Eric Armengaud, and Christian Josef Kreiner. Automotive Embedded Software: Migration Challenges to Multi-Core Computing Platforms. In *Proceedings INDIN 2015*, pages 110–118, 2015.

[41] Farhang Nemati, Moris Behnam, and Thomas Nolte. Efficiently migrating real-time systems to multi-cores. In *2009 IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–8.

[42] Sara Abbaspour Asadollah, Hans Hansson, Daniel Sundmark, and Sigrid Eldh. Towards Classification of Concurrency Bugs Based on Observable Properties. In *2015 IEEE/ACM 1st International Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS)*, pages 41–47.

[43] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. Cheddar: a flexible real time scheduling framework. In *SIGAda*, pages 1–8. ACM, 2004.

[44] C. Norstrom, A. Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications*, pages 182–189, 1999.

[45] Alessio Bucaioni, Saad Mubeen, Federico Ciccozzi, Antonio Cicchetti, and Mikael Sjödin. Modelling multi-criticality vehicular software systems: evolution of an industrial component model. *International Journal on Software and Systems Modeling*, 19:1283–1302, June 2020.

[46] F. A. Bianchi, A. Margara, and M. Pezzè. A survey of recent trends in testing concurrent software systems. *IEEE Transactions on Software Engineering*, 44(8):747–783, Aug 2018.

[47] The UML profile for MARTE: Modeling and analysis of real-time and embedded systems. OMG Group, 2010.

[48] Barbara A Kitchenham and Shari L Pfleeger. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London, London, 2008.

[49] Adesola Sola and Baines Tim. Developing and evaluating a methodology for business process improvement. *Business Process Management Journal*, 11(1):37–46, 2005.

Chapter 8

Paper D: Fogification of Industrial Robotic Systems: Research Challenges

Shaik Mohammed Salman, Vaclav Struhar, Alessandro V. Papadopoulos, Moris Behnam, Thomas Nolte.
In Proceedings of the Workshop on Fog Computing and the IoT.
(IoT-Fog '19)

Abstract

To meet the demands of future automation systems, the architecture of traditional control systems such as the industrial robotic systems needs to evolve and new architectural paradigms need to be investigated. While cloud-based platforms provide services such as computational resources on demand, they do not address the requirements of real-time performance expected by control applications. Fog computing is a promising new architectural paradigm that complements the cloud-based platform by addressing its limitations. In this paper, we analyse the existing robot system architecture and propose a fog-based solution for industrial robotic systems that addresses the needs of future automation systems. We also propose the use of Time-Sensitive Networking (TSN) services for real-time communication and OPC-UA for information modelling within this architecture. Additionally, we discuss the main research challenges associated with the proposed architecture.

8.1 Introduction

Industrial robots have become an integral part of the industrial automation environment with traditional application areas such as spot welding, spray painting and machining [1]. Currently, each robot comes with a dedicated controller that provides motion control, programming interfaces and physical interfaces for integrating field devices such as sensors and actuators via industrial networks [2]. The controller is designed to meet the real-time constraints demanded by motion control algorithms as well as real-time requirements of industrial networks. These controllers, however, have fairly limited computational resources restricting the integration of complex functionality such as image processing, multi-robot motion control and other complex applications [3]. Although multi-robot motion control within a single controller is possible with solutions such as ABBs multimove functionality, the number of robots that can be controlled is still limited. Additionally, flexible production requirements of future automation systems impose demands such as firmware updates and hardware maintenance without any production downtime [4]. Meeting such requirements within the existing architecture is non trivial. Supporting the required infrastructure for augmented reality based immersive human-machine interaction concepts as shown by Paelke et al. [5] and Guhl et al. [6] will also require significant computational capacity and communication bandwidth. While increasing hardware capabilities within the controller can be a presented as a solution, this only addresses some of the concerns, validating the need to investigate cloud and fog-based architectures.

While cloud computing offers significant computational resources on demand, it does not guarantee real-time performance as required by traditional control applications [7, 8, 9]. Fog computing [10], is a new paradigm that allows utilization of computational resources near the edge of the network close to the source of the data. It introduces an intermediate layer between the cloud and the end devices that consists of a number of devices, called fog nodes, that are interconnected to form a network and these devices offer their computational resources (e.g., CPU, storage), for use by applications within this network. While the well established cloud computing paradigm provides services ranging from collection of historical data to big data analysis, fog computing complements the cloud functionality by providing local data processing. This capability, along with real-time communication mechanisms such as TSN [11], enables the fog-based architecture to provide predictable communication times.

Authors of [12, 13] have discussed fog-based solutions for general robotic systems and highlighted the advantages of using fog-based architecture for such applications. While Hao et al. [14] provided a generic software architecture for fog

computing, Faragardi et al. [8] provided a time predictable framework for a smart factory integrating the fog and cloud layers. Skarin et al. [15] developed a test bed to study the feasibility of a fog-based approach for control applications, while Pallasch et al. [16] and Mubeen et al. [17] showed the feasibility of using an edge based solution for combining cloud and field devices. Vick et al. [18] presented the concept of "Using OPC-UA for Distributed Industrial Robot Control".

Based on the evidences provided by the above studies, in this paper, we propose the "Fogification" of industrial robotic system architecture to enhance the capabilities of industrial robotic systems by taking advantage of the fog computing paradigm. We use the term "Fogification" to define the integration of fog computing platform within industrial systems such that they benefit from the low-latency, distributed fog based resources within the local networks as well as from the high performance cloud computing environment. In order to highlight the advantages of using the fog-based architecture for industrial robotic systems, we analyse the existing architecture and identify its main limitations. Based on these limitations, we introduce a fog-based architecture for industrial robots and we identify the main research challenges associated with the proposed architecture.

8.2 Existing System Architecture

A single industrial robotic system typically consists of a mechanical unit called the manipulator, a controller, and a graphical controller interface device (see Fig. 8.1). The controller is the main processing unit that executes the control algorithms for robot motion. It also provides mechanisms to program the robot motion and to configure any additional behaviour required within the robot environment. The controller also provides multiple communication interfaces for interaction with fieldbus networks and the enterprise network including cloud based services.

In a multi-robot system, a number of robots are programmed together to accomplish a process application such as welding and painting. Here, the individual controllers are connected to each other to form a local network. To ensure synchronisation between different controllers, a fieldbus network and a PLC is utilised.

8.2.1 System Components

In this section, we give a brief overview of the different components of the current robotic system as shown in Fig. 8.1.

Manipulator

The manipulator is the mechanical arm with varying degrees of freedom.

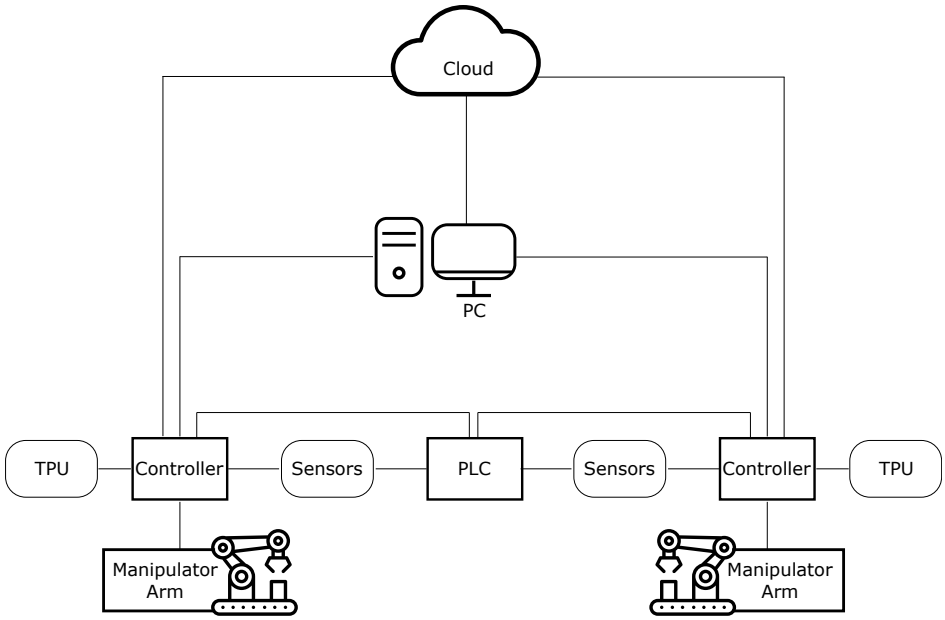

Figure 8.1. Existing Industrial Robotic System Architecture.

Controller

The controller is responsible for controlling the manipulator motion and providing required interfaces for interaction with the robot environment such as other robot controllers, devices such as PLCs, conveyors and other sensors and actuators.

Teach Pendant Unit (TPU)

The TPU acts as the human machine interface device for the controller. It is physically connected to the robot controller and can be used to manually move the manipulator, to configure different parameters of the system and to visualise the current state of the system via the device display.

Programmable Logic Controllers (PLC)

The PLCs are used for controlling the synchronisation and coordination between different devices within the robot environment.

Sensing Devices

Sensing devices are used to provide information about the robot environment. These are connected to the controller via industrial networks or PLCs.

PC Software

The PC software provides visualisation, simulation, configuration and editing tools for robot programming and monitoring.

Cloud

The controllers are capable of connecting to the cloud services where data from the controller is stored and used for analysis. Currently, the cloud services are mainly utilised for collecting system data and not for control.

8.2.2 Classification of Controller Functions

The functionality of the existing controller firmware can be classified under three main categories, i.e, control, configuration and communication.

Control

The control functionality is responsible for path planning, trajectory generation and low level control [19] and it requires real-time capabilities from the system. Currently, the controller system provides real-time guarantees via a real-time operating system.

Configuration

The configuration functionality allows the users to configure the system behaviour such as defining the maximum speed of the robots and providing information about the robot environment in terms of available sensors and the connected networks. The configuration functionality does not require real-time guarantees and is usually carried out offline while the system configuration is updated when the manipulators are not in motion.

Communication

The communication functionality refers to user interaction features such as the robot programming language, the interface for the teach pendant unit, communication with different field devices via fieldbus networks and connectivity to enterprise

networks and the cloud services. These communication functions impose both real-time requirements as well as non real-time requirements on the system. Connectivity to fieldbus interfaces, for example, requires real-time guarantees while the connectivity to enterprise networks can be non real-time.

8.2.3 Limitations

The existing architecture relies heavily on the controller component to achieve the functional behaviour of the system. In addition to limited resources on the controller, the existing architecture has certain limitations in terms of supporting flexible production requirements of future automation systems [4]. Some of the key limitations are discussed below.

L1: *Computational Resources* - The available computational resources within the existing controllers are not sufficient for the implementation of complex functionality. For example, computationally demanding tasks such as image processing when using vision based sensors cannot be carried out within the controller due to the limited resources.

L2: *Fleet Management* - In the existing setup, introducing new functionality or improving performance of the existing system via over the air software updates is limited since doing so requires production downtime. Also, controllers have limited connectivity to networks outside the factory environment, restricting the robot vendors from updating the controller firmware.

L3: *Environment Interaction* - Currently, the data from advanced sensors such as cameras is not directly shared with all the controllers which are part of the same environment. Normally, another computer is necessary to do the pre-processing. This can introduce latencies and affect the ability to have comprehensive information for better path planning and control.

L4: *Hardware Dependency* - The controller firmware is designed to make optimal use of the available controller hardware. Replacing the hardware with different hardware without significant changes to the controller firmware is non-trivial.

L5: *Connectivity* - In order to support communication with various industrial networks along with enterprise and cloud connectivity, the controllers need to provide multiple communication interfaces. Maintaining multiple interfaces increases the total system cost. Additionally, the current solutions do not allow for a seamless communication of the mobile platform, and they are typically based on wired, rather than wireless connection.

8.3 Fog Based System Architecture

In the fog-based system architecture, we propose the utilisation of the fog layer as the computational platform and TSN [11] as the communication mechanism for both fog-to-fog communication as well as fog-to-field communication. To support a standard mechanism for information exchange, we propose the use of OPC-UA data modelling standards [18]. We describe the key components of the architecture in Fig. 8.2.

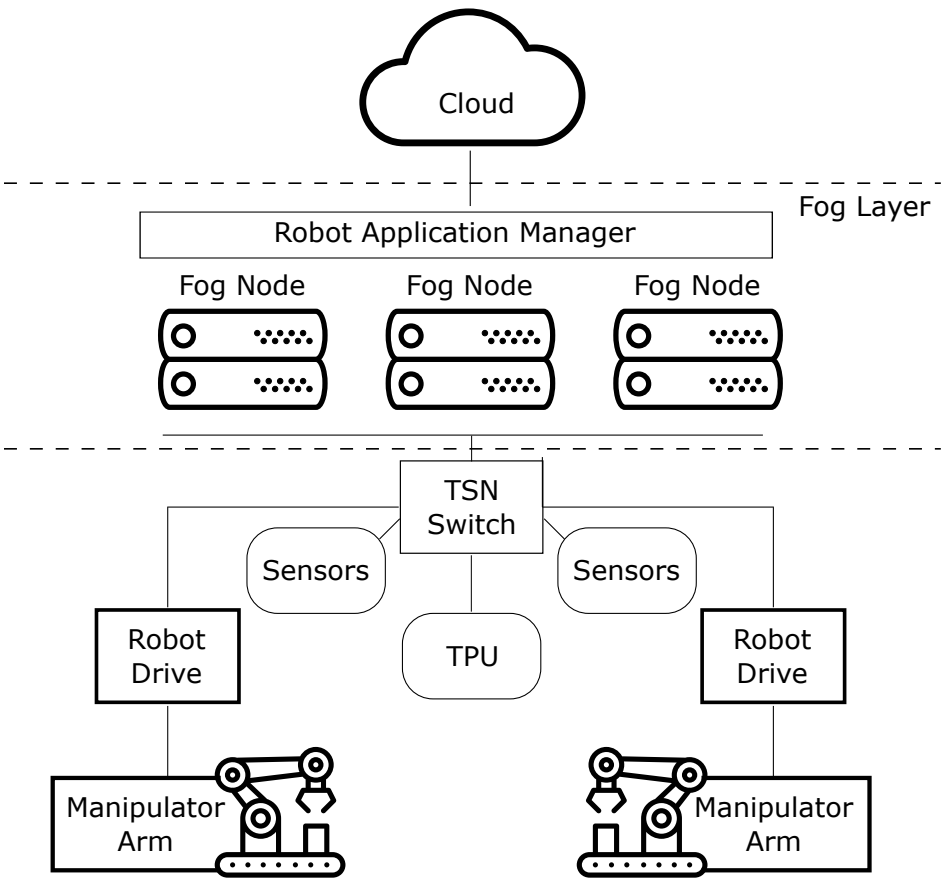

Figure 8.2. Proposed fog based architecture for Industrial Robotic System.

8.3.1 Computational Platform

In the fog-based system architecture, the controller functionality is provided by a fog-based computational platform. This platform will provide the required

computing capacity to execute the system functions. Fig. 8.2 shows the fog-based system architecture. Here, the controller component is replaced by the robot drive system and the entire controller functionality is moved to the fog layer. Replacing the individual controllers with a fog-based platform provides multiple advantages, such as compute capacity on demand and additional storage. This addresses limitation L1. The robot application manager, as shown in Fig. 2, acts as an orchestrator that distributes the controller functionality within the fog layer and manages the firmware updates by interacting with the cloud layer. It stores the updates in one of the fog nodes and applies these changes to the system when it is idle, addressing the limitation L2.

8.3.2 Communication Interfaces

System components such as the manipulator and the TPU, which are normally physically connected to individual controllers, will form a local network with the fog platform via TSN [11]. Also, field level sensors that are normally interfaced via fieldbus networks and connected to individual controllers will now be connected via the TSN network to the fog platform. It can also be possible to integrate the fieldbus network to the fog platform via gateways. The information from such sensors can now be shared by different applications running on the platform for improved system performance, addressing the limitation L3. Using OPC-UA standards for data modelling will provide a standard mechanism for information exchange, replacing the multiple communication protocols currently used, addressing a part of the limitation L5.

8.3.3 Software Deployment

To execute the controller functions in the fog layer, the application needs to be designed such that it is hardware agnostic, addressing the limitation L4. In the new architecture, we propose the use of a service oriented approach to meet the functional requirements [18]. For example, the robot program interpreter can be deployed as an independent service that can execute on one of the fog nodes. The trajectory generator, waiting for the input from the robot program interpreter, executes on another node within the network, while the low level control component is executing on yet another node. The real-time communication interface, responsible for data exchange with the sensors and actuators, can run as a background service, which can be subscribed to by other services for data manipulation and control activities. An important assumption we make here is that the real-time requirements of all the above components can be met.

8.4 Research Challenges

Introduction of fog computing in the proposed industrial system architecture brings a myriad of research challenges that needs to be addressed. The need to meet the extra-functional properties of the system, such as safety, security [20], availability and reliability [11], makes the deployment of fog computing complex and needs many considerations.

In this section, we identify some of the research challenges that must be addressed in order to build the proposed industrial robot system.

8.4.1 Orchestration and Inner Fog Architecture

The fog layer is a distributed environment consisting of many heterogeneous fog nodes offering their computational capacity. For an efficient use of the resources, a fog orchestrator needs to be defined. Fog orchestration [21] is a key component that partitions the workload between fog nodes, keeps track of available resources, cooperates with the cloud and manages unexpected situations that occur in the fog layer. Choosing the right orchestration techniques is crucial as it affects the behavior of the entire system. Here, the challenge is to find an appropriate architecture, in a holistic way, that will meet all the functional/extra-functional requirements of the presented industrial system.

Skarlat et al. [22] proposed a hierarchical approach where fog orchestrator manages a fog colony (logical unit of fog devices) and divides work among this fog colony, neighbouring colonies or cloud. Whereas in [23], the fog orchestrator works only as a workload balancer that couples a fog node with a device requiring computational resources. Nodal collaboration [24] defines how fog nodes communicate to each other. It provides two basic models: peer-to-peer, where each node can directly communicate to each other, and the client-server model, where there is a hierarchy of servers providing services and client nodes consuming them.

8.4.2 Real-Time Guarantees

The proposed system must meet strict timing constraints and these constraints are valid for both the computation time and data transmission time in the network. Therefore, we need to find appropriate scheduling and analysis mechanisms for the conjunction of both the computational and the transmission part. The data transmission time can be bounded by use of TSN. Pop et al. [11] proposes the use of TSN in fog Computing, whereas in the paper [25], optimization strategies for TSN are shown.

Additionally, the problem of variable timing constraints during the operational process needs to be tackled. For example, the robots may require control instructions at changeable rates, i.e., complex small-grained movements or movements at high speed demand instructions at high rates, while slow speed motions need lower pace of instructions from the Trajectory Generator.

8.4.3 Resource Isolation and Virtualization

In the fog layer, the concurrently running applications may influence each other, especially during high utilization. It is given by the fact that the fog nodes are realised using common hardware, utilizing common resources as system buses, CPUs and memory. A proper mechanism that ensures that a number of concurrently running application on a single fog node do not interfere with each other in an unpredictable manner must be designed. Additionally, the applications must finish the computation within a given time and improper isolation of resources may introduce unpredictable delays and affect the timing requirements. Moreover, failure of an application must not lead to failure of other applications running on a single node.

8.4.4 Resource Estimation and Workload Optimization

To ensure real-time performance, a proper analysis of the system must be done to estimate the resources necessary for an application. Also, the system workload can vary significantly depending on the applications running at any given time. Appropriate strategies to deal with these situations must be developed. One of the solutions can be dynamical provisioning of fog nodes [26], where, if the workload is high (or is predicted to be high), additional fog nodes are started up.

8.4.5 Monitoring and Optimization

The architectural transition from a dedicated controller to the whole distributed layer of fog nodes introduces additional complexities to the robot control. There may be errors in the system due to faulty resource allocation, faults in communication between the robots and the fog, faults in communication between fog nodes and in virtualization, etc. Thus, the whole fog layer needs thorough monitoring to enable traceability of functional/non-functional properties of the system to address the errors due to these faults. Additionally, monitoring is a key component to enable optimization and dynamic reconfiguration of the fog layer.

8.4.6 Safety

The system should identify and recover from unexpected states caused by failure of fog nodes (hardware or software) or communication links. Employing a single orchestrating node that manages the whole fog network introduces a single point of failure that may paralyze the whole fog layer and affect the safety of the system. Techniques to address such scenarios need to be investigated.

8.4.7 Security

The use of fog computing allows centralized updating and deploying applications in the fog environment. An attacker can remotely take over the manufacturing process and make the system unsafe. This may be a potential security risk that must be taken into account. Therefore, a proper access control and intrusion detection mechanism must be implemented at different layers of the architecture.

8.5 Conclusion and Future Work

The traditional robot system architectures need to evolve to a new architectural paradigm to meet the demands of flexible production environments. The proposed fog-based system architecture addresses such demands while introducing new research challenges that need to be addressed. In our future work, we intend to address the research challenges associated with the fog-based architecture.

Acknowledgement

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation

Bibliography

[1] Antoni Grau, Marina Indri, Lucia Lo Bello, and Thilo Sauter. Industrial robotics in factory automation: From the early stage to the Internet of Things. In *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 6159–6164. IEEE, 2017.

[2] Brendan Galloway and Gerhard P. Hancke. Introduction to Industrial Control Networks. *IEEE Communications Surveys & Tutorials*, 15(2):860–880, 2013.

[3] Vaclav Struhar, Alessandro V. Papadopoulos, and Moris Behnam. Fog computing for adaptive human-robot collaboration. In *International Conference on Embedded Software 2018*, 2018.

[4] Thomas Goldschmidt, Stefan Hauck-Stattelmann, Somayeh Malakuti, and Sten Grüner. Container-based architecture for flexible industrial control applications. *Journal of Systems Architecture*, 84:28–36, 2018.

[5] Volker Paelke. Augmented reality in the smart factory: Supporting workers in an industry 4.0. environment. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–4. IEEE, 2014.

[6] Jan Guhl, Son Tung, and Jorg Kruger. Concept and architecture for programming industrial robots using augmented reality with mobile devices like microsoft HoloLens. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2017.

[7] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726–740, 2014.

[8] Hamid Reza Faragardi, Saeid Dehnavi, Mehdi Kargahi, Alessandro V. Papadopoulos, and Thomas Nolte. A time-predictable fog-integrated cloud

framework: One step forward in the deployment of a smart factory. In *2018 Real-Time and Embedded Systems and Technologies (RTEST)*, pages 54–62. IEEE, 2018.

[9] Daniel Hallmans, Kristian Sandström, Thomas Nolte, and Stig Larsson. Challenges and opportunities when introducing cloud computing into embedded systems. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 454–459. IEEE, 2015.

[10] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. 2018.

[11] Paul Pop, Michael Lander Raagaard, Marina Gutierrez, and Wilfried Steiner. Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking (TSN). *IEEE Communications Standards Magazine*, 2(2):55–61, 2018.

[12] Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Fog-Enabled Multi-Robot Systems. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2018.

[13] Siva Leela Krishna Chand Gudi, Suman Ojha, Benjamin Johnston, Jesse Clark, and Mary-Anne Williams. Fog Robotics for Efficient, Fluent and Robust Human-Robot Interaction. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–5. IEEE, 2018.

[14] Zijiang Hao, Ed Novak, Shanhe Yi, and Qun Li. Challenges and Software Architecture for Fog Computing. *IEEE Internet Computing*, 21(2):44–53, 2017.

[15] Per Skarin, William Tarneberg, Karl-Erik Å rzén, and Maria Kihl. Towards Mission-Critical Control at the Edge and Over 5G. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 50–57. IEEE, 2018.

[16] Christoph Pallasch, Stephan Wein, Nicolai Hoffmann, Markus Obdenbusch, Tilman Buchner, Josef Waltl, and Christian Brecher. Edge Powered Industrial Control: Concept for Combining Cloud and Automation Technologies. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 130–134. IEEE, 2018.

[17] S. Mubeen, P. Nikolaidis, A. Didic, H. Pei-Breivold, K. Sandström, and M. Behnam. Delay mitigation in offloaded cloud controllers in industrial iot. *IEEE Access*, pages 4418–4430, 2017.

[18] A. Vick and J. Krueger. Using OPC UA for Distributed Industrial Robot Control. In *ISR 2018; 50th International Symposium on Robotics*, pages 1–6, 2018.

[19] Michal Matuga. Control and positioning of robotic arm on CNC cutting machines and their applications in industry. In *2018 Cybernetics & Informatics (K&I)*, pages 1–6. IEEE, 2018.

[20] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero. An experimental security analysis of an industrial robot controller. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 268–286, 2017.

[21] Karima Velasquez, David Perez Abreu, Marcio R. M. Assis, Carlos Senna, Diego F. Aranha, Luiz F. Bittencourt, Nuno Laranjeiro, Marilia Curado, Marco Vieira, Edmundo Monteiro, and Edmundo Madeira. Fog orchestration for the internet of everything: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 2018.

[22] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards QoS-Aware Fog Service Placement. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 89–96. IEEE, 2017.

[23] Y. Lin and H. Shen. Cloud fog: Towards high quality of experience in cloud gaming. In *2015 44th International Conference on Parallel Processing*, pages 500–509, 2015.

[24] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog Computing: A Taxonomy, Survey and Future Directions. In Beniamino Di Martino, Kuan-Ching Li, Laurence T. Yang, and Antonio Esposito, editors, *Internet of Everything*, Internet of Things, pages 103–130. Springer Singapore, Singapore, 2018.

[25] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner. Design optimisation of cyber-physical distributed systems using ieee time-sensitive networks. *IET Cyber-Physical Systems: Theory Applications*, 1(1):86–94, 2016.

[26] Said El Kafhali and Khaled Salah. Efficient and dynamic scaling of fog nodes for iot devices. *The Journal of Supercomputing*, 73(12):5261–5284, 2017.

Appendix

Table 8.1. Survey questionnaire and the mapping between the questions and the different stages of the methodology.

No.	Methodology Stage/ Purpose	Question
1	Participant information	Before you proceed, please take the time to read the paper describing the methodology.
2	Participant information	Name of the organization:
3	Participant Relevance	Does your application have real-time components?
4	Participant Relevance	Is your application designed to run on multi-core platforms?
5	Participant Relevance	Have you in the past, migrated your application to a multi-core platform?
6	Participant Relevance	Are you considering migrating the application to a multi-core platform?
7	Exploratory	Did you follow any specific methodology or guidelines to migrate the application to a multi-core platform?
8	Exploratory	Will you recommend the existing approach to others?
9	Exploratory	If you'd like to provide more information about the used methodology, please do so here.
10	Problem relevance	Select the preferred option : a) I prefer to redesign and redevelop the application from scratch for a multi-core platform. b) I prefer to redesign but also reuse the existing code for a multi-core platform.
11	Architecture Abstraction and Representation	Can your application be described with timing properties such as "worst case execution times", "period", "deadlines"?
12	Architecture Recovery (runtime analysis)	Is it possible to identify a particular build of the application that can be used to recover the timing requirements of the application and can such timing requirements be used to create a model of the application?
13	Architecture recovery (documentation)	Is the design of your application documented?
14	Architecture recovery (documentation)	Does the application design documentation contain timing properties?
15	Architecture recovery (documentation)	The behaviour of your application can be:(choose one) a) accurately inferred from the design documentation b) cannot be accurately inferred from the design documentation
16	Architecture Recovery (runtime analysis)	Does your application provide diagnostic logs of runtime behaviour?
17	Architecture Recovery (runtime analysis)	The code instrumentation : a) is fully reliable. b) may not be fully reliable.
18	Architecture Recovery (runtime analysis)	Does your application have multiple configurations?
19	Architecture Recovery (runtime analysis)	Does the runtime behaviour of the application depend on the configuration?
20	Architecture Recovery (runtime analysis)	Do you test all possible configurations of the applications?
21	Architecture Recovery (runtime analysis)	Which configuration do you test
22	Architecture Transformation (Ranking of solutions)	Do you have any existing process/guidelines in place to evaluate and choose between different solutions that may be specific to multi-core platforms?
23	Architecture Transformation (Ranking of solutions)	Is it possible to define measurable parameters that will suit your application's timing requirements to choose one solution over the other?
24	Architecture Transformation (Ranking of solutions)	Is it possible to prioritize the measurable parameters that will suit your application requirements to choose one solution over the other?
25	Verification Migration	Does your application have a verification and validation process in place for checking functional correctness?
26	Verification Migration	Will you reuse the existing tests to verify the behaviour on multi-core platforms?
27	Feasibility	Feasibility: Can the methodology described be followed?
28	Feasibility	Please briefly describe the reason behind your answer here:
29	Usability	Usability: Is the methodology workable? Are the steps and tools easy to use and apply?
30	Usability	Please briefly describe the reason behind your answer here:
31	Usefulness	Usefulness: Is the methodology worth following? Does the methodology produce results that the business will find helpful?
32	Usefulness	Please briefly describe the reason behind your answer here:
33	Overall comments	Which part of the methodology will you like to improve? (you can choose multiple options)
34	Overall comments	Please provide any suggestions and improvements you want to see in the methodology here: