Schedule-based Service Choreographies for Real-Time Control Loops

Thomas Kothmayr, Alfons Kemper
Technische Universität München, Germany
{kothmayr, kemper}@in.tum.de

Andreas Scholz, Jörg Heuer
Corporate Technology, Siemens AG
{andreas.as.scholz, joerg.heuer}@siemens.com

***Abstract*—Today's manufacturing industries are undertaking efforts to further increase the flexibility of their facilities. One way to achieve this goal is the development of distributed, service-oriented architectures (SOA). A challenge for SOAs is the continued support of the real-time requirements imposed by industrial control tasks. This paper presents an approach for executing service choreographies with strong real-time guarantees without a central point of control. Engineers can design the automation workflow as a graph of communicating tasks which are then assigned to devices in the network. Our method generates a cyclic, non-preemptive schedule for each device to achieve global cooperation between them. Using an approach that combines several heuristics, a valid solution for over 99% of the 1.2 million test cases was found. On average, this method was over two orders of magnitude faster than an approach based on MIP-solvers.**

I. Introduction

New manufacturing strategies, such as agile manufacturing and mass customization, encompass the vision of rapid engineering and innovation. This vision mandates further improvements in the area of manufacturing systems [1]. Traditional manufacturing systems provide cost-effective production at high volumes. These centralized, monolithic and scan-based control systems are optimized for a given physical and network configuration. The modularization and reuse of source code running on these systems is hindered by a tight coupling with their environment. Furthermore, installation and setup comprise up to one third of their life-cycle costs [2].

Advances in hardware to support manufacturing are already evident: Industrial Ethernet is gaining traction [3] while smarter embedded devices are performing an increasing number of orthogonal tasks. Recent developments include the roll out of smart field devices, like Ethernet-equipped sensors and actuators. In the automotive and avionic industries a different process with similar results can be observed. Subsystems are consolidated through a single real-time communication network to reduce weight and costs. In both domains the result is a hardware architecture of multiple networked processing units. The pressure to innovate is now focused on software where development approaches are shifting to model-driven [4] or service-oriented [5] methodologies. Both approaches mandate decomposing an embedded application into a set of interdependent tasks. The two most common design paradigms for distributed real-time systems are thus the traditional hierarchical approach and the emerging approach, which we refer to as the SOA approach in this paper.

978-1-4673-7929-8/15/$31.00

The SOA-paradigm is designed to address the weaknesses, such as inflexibility and high setup costs, of the traditional approach to automation. The strengths of SOA lie in its flexibility and adaptability. Messages are sent in a push-based manner making more efficient use of the limited communication resources than the traditional scan or pull-based communication pattern. Extensive research efforts have been undertaken to push the performance envelope of SOA (or more specifically, web-service) technology on constrained embedded devices [6]. These endeavors have seen SOA solutions developed for lower and lower layers of the automation pyramid [5]. It is, however, difficult to verify the emergent properties of distributed event-based systems. Implementations with centralized control are therefore likely to be favored in the near future [7]. Centralized control is realized through a service orchestration which is executed by an orchestration engine that invokes each of the individual services composed in a workflow [8]. While such an orchestration represents an improvement over the traditional model, the full impact of the SOA paradigm could be realized through service choreographies providing global cooperation without central coordination.

Our proposal for a real-time service oriented architecture (rtSOA) tries to reconcile both approaches by enabling global coordination of field devices through deterministic communication and computation schedules with verifiable real-time properties. Similar to the orchestration in the SOA paradigm, the schedules are derived from a model-based representation of the control loop. During design and development, the control loop is modeled as a directed acyclic graph (DAG) of dependent tasks, similar to an automation workflow incorporating individual services in a SOA approach We therefore also refer to the task-DAG as workflow. The workflow carries global timing information, such as its global deadline and period. Timing restrictions on a per-job level are derived from global constraints when binding a set of workflows to devices connected through a real-time network. This binding is performed by a skilled engineer who is supported by the rtSOA planning tool. rtSOA generates a static, cyclic, non-preemptive schedule for each device that includes all relevant tasks from a real-time workflow. The network communication is implicitly included in the generated schedules as each task is scheduled to finish before a certain communication deadline and the receiving tasks on different devices are only scheduled to start after the delivery of the relevant data from their predecessors.

When specifying workflows, an engineer is freed from timing constraints imposed by the hardware and network. New devices can quickly be integrated into the existing infrastructure by deploying those sub-tasks of a workflow that are specific to the device and subsequently generating new schedules that include the device. While we focus on applications in the context of manufacturing, rtSOA could also be applied in other areas, e.g. the automotive or avionic industries. The architecture of a system developed following the rtSOA approach is depicted in Figure 1.

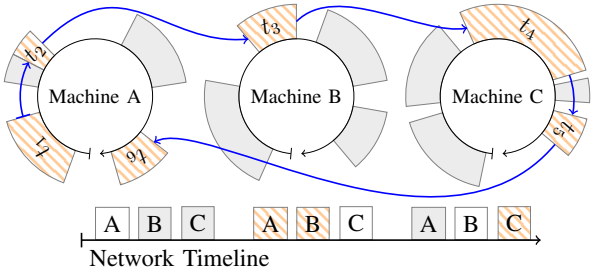


Fig. 1: Devices cooperate in a distributed service choreography by locally executing cyclic schedules and communicating over a deterministic real-time TDMA network. Gaps in the machine time-line and white slots on the network time line represent unused network and computational resources. The hatched areas indicate resources used by an example workflow whereas the gray areas indicate resources used by other applications. Arrows represent data dependencies.

Related work on SOAs, which we outline in Section II, has often followed an approach based on a central service orchestrator. Our architecture, detailed in Section III, is targeted at constrained embedded devices and must therefore deliver a lightweight service choreography that is, nevertheless, able to achieve hard real-time guarantees. From a suitable system specification, a planning process is triggered which produces a cyclic, non-preemptive schedule for each device. The feasibility of this architecture hinges on the ability to quickly find a valid schedule across all involved devices. We have evaluated heuristics for deadline assignment [9], [10] and scheduling [11] in previous work [12], [13]. This paper proposes the use of multi-processor scheduling algorithms to combine both tasks. We outline the employed heuristics discussed in the literature and our own heuristics in Section IV. The evaluation of 1.2 million different system configurations (Section V) shows that our approach generates valid schedules for over 99% of the examined cases. This constitutes a significant improvement over our previous work [12] which only achieved valid solutions in 85% on the same data set.

The contributions of this paper are: The demonstration that heuristics are an attractive alternative to state of the art mixed integer program (MIP) solvers for determining distributed service choreographies, an extensive evaluation of deadline assignment and distributed scheduling heuristics on over 1.2 million feasible workflows, and the identification of several new or adapted heuristics for scheduling of distributed workflows with fixed task-placement.

II. Related Work

A shift toward service-oriented architectures in the automation domain is evident both in academia as well as in industry. Section II-A will briefly outline important specifications in the industrial domain before Section II-B presents related work with a focus on EU research projects dealing with SOA for industrial applications. Section II-C places our work in context with other methods for deployment generation and scheduling.

A. Industry

The two most prominent specifications concerning service orientation for industrial devices are the Devices Profile for Web Services (DPWS) [14] and the Object Linking and Embedding for Process Control Unified Architecture (OPC UA) [15]. Both specifications are conceptually similar. That is, they implement a SOA through Web Services and rely on built-in base services for discovery and service reservation. The main difference between the two is their intended target area. OPC UA is a service-oriented version of the original OPC architecture and its main mission is to connect industrial devices to applications for control and supervision on higher levels of the automation pyramid [16]. It is therefore not directly aimed at the communication between the devices themselves. In contrast, DPWS is a web service middleware and profile that aims to constrain the WS-* set of standards to make them suitable for embedded use. DPWS is aimed directly at the devices performing the automation task on the lowest levels of the automation pyramid. Cândido et. al [16] give a more detailed comparison of DPWS and OPC UA.

In terms of architecture, rtSOA is closely related to DPWS, meaning it is also aimed at the device level and meant to enable peer-to-peer communication. However, rtSOA is aimed at hard real-time control loops instead of general purpose discovery and eventing which are the aims of DPWS. Discovery, and especially device description, could be implemented following the DPWS specification. Thus, rtSOA can be seen as orthogonal to industry standards such as DPWS.

B. EU projects

The SIRENA project provided the first embedded DPWS stack [17] and the closely related SODA project extended the SIRENA framework by providing a toolkit for manageability, orchestration and security [18]. Together, these projects proved the feasibility of web services on embedded devices. The RI-MACS project used DPWS compliant services for soft real-time and best-effort tasks [19]. For performance reasons, RI-MACS chose a separate communications stack in addition to DPWS to fulfill hard real-time requirements. The SOCRADES project built on the SIRENA and SODA results to further the vertical cross-layer integration between shop floor and enterprise systems [20]. The AESOP project investigated the feasibility and limits of using a SOA-based approach inside control loops [5]. By implementing several prototypes, the project closely investigated the performance implications of using Web-Services for the concurrent control of several thousand devices.

SIRENA, SODA and SOCRADES achieved the horizontal integration of industrial control devices with higher-level enterprise systems. Since our work on rtSOA is focused on providing hard real-time guarantees on the underlying device level, RI-MACS, and especially AESOP, can be considered conceptually similar projects. AESOP showed the feasibility of integrating embedded devices in a control loop through a SOA. In our opinion, message exchange in a control loop is possible by leveraging one of the protocol stacks investigated by AESOP [5]. Additionally, work performed in the context of the AESOP project has already pointed out that a distributed choreography approach to SOA is preferable to the classical orchestration-based approach [21]. Our work focuses on the central planning required to achieve performant, hard real-time choreographies for devices in a tight control loop.

C. Scheduling and deployment generation

The field of multicore real-time scheduling is closely related to our approach. Davis and Burns conducted a survey over hard real-time scheduling for multiprocessor systems [22]. Much of the work they outlined uses a preemptive task model. Since rtSOA is targeted at resource constrained devices, which might not have a real-time operating system (RTOS) that supports task preemption, we use a non-preemptive model. Multicore scheduling usually also neglects communication costs between tasks, because the tasks are assumed to share the same memory. We consider a distributed system model requiring detailed attention to communication costs and timing.

In this light, our work is closely related to the field of distributed scheduling. As in the multicore scheduling problem, a distributed scheduling algorithm essentially should solve two problems. 1) The allocation problem, i.e., deciding on which processor or machine a task should run, and 2) the priority problem, i.e., in which order the tasks should be executed [22]. Additionally, a distributed scheduling algorithm should consider the communication delay between machines. In this paper, we study the performance of several well-known distributed scheduling algorithms [23]–[26] in the context of fixed task-placement and TDMA-communication between devices (Section V). Fixed task-placement means that tasks are not allocated algorithmically, but manually (c.f. Section III).

Our previous work [12] followed a two-step heuristic approach where the heuristics were not aware of the underlying TDMA-network timing. In this paper we use the best-performing heuristic for soft real-time workflows [9] as a point of reference. We also introduce two new heuristics for distributed scheduling with fixed task-placement (Section IV).

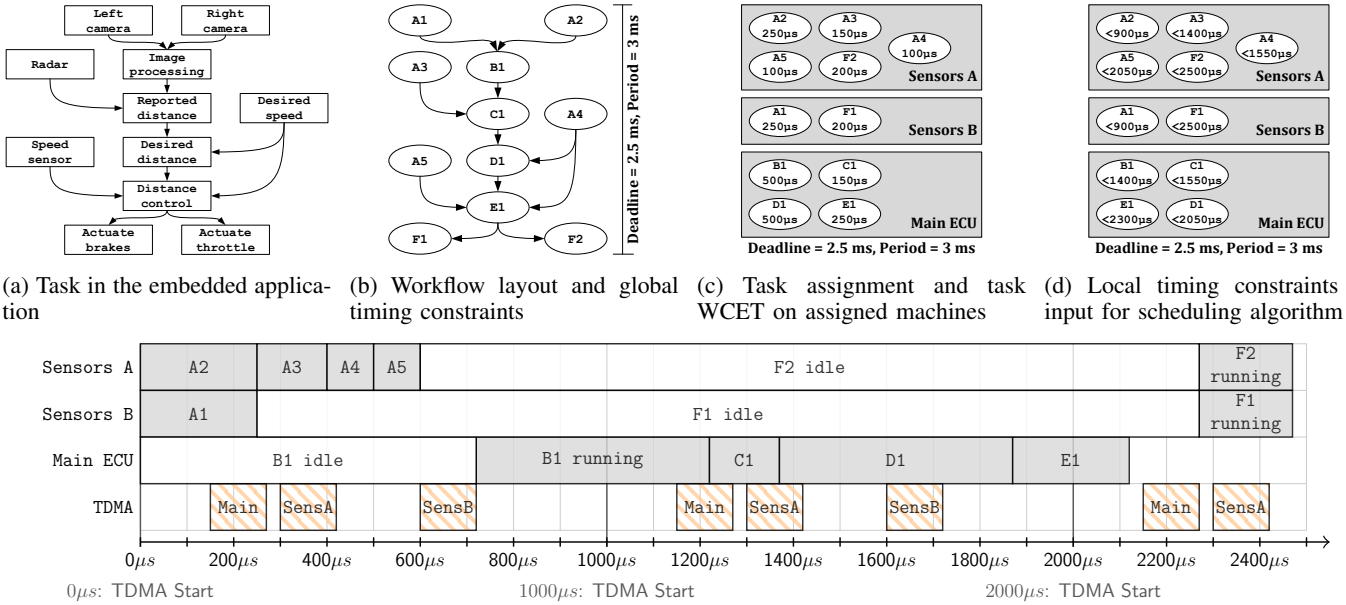
Another related area of research focuses on automatic system deployment and distributed scheduling. Voss and Schätz [4] use SMT-solvers to find both a suitable task placement and schedules for each device. Our method favors a fixed task-placement and subsequently generates individual device schedules through heuristics. Heuristics outperform solutions based on satisfiability solvers by orders of magnitude, in terms of run time, while solving 99% of the examined problems, as discussed in our previous work and in Section V of this paper.

III. System Architecture

There are two separate domains that need to be specified when deploying a SOA in the industrial context: the automation workflow and the infrastructure on which it will be executed. The infrastructure comprises the devices which partake in the automation process, the physical capabilities of each device (e.g., provision of sensor data, high computational capability, etc.) and the characteristics of the network which connects the devices. Previous work has demonstrated that web service technology can be leveraged in the embedded context and efforts to standardize DPWS are well under way. The capabilities of a device can therefore be described, advertised and discovered through industry standards (Section II-A). Service discovery at run-time is not required for planning and deploying the tasks of the control loop but offers possibilities for seamless integration into less time critical applications. Similarly, the rtSOA planner does not require run-time discovery of network particularities. Our working assumption is that the network configuration, including addressing, message delay and TDMA slot assignment, is made available to the planner together with matching device descriptions.

The goal of the rtSOA system is to provide a distributed service choreography: each device fulfills its part to cooperatively realize the control loop. The target platforms for rtSOA span from large control systems to very small embedded devices, such as smart sensors or actuators. We use the term smart device to describe a sensor or actuator attached to a system on a chip with several kilobytes of memory, a CPU clock rate of a few Megahertz and integrated networking capability. We do not assume that any advanced real-time operating system (RTOS) is available. The output of the rtSOA planning stage is a static, non-preemptive, cyclic schedule for each device. The job timing in each schedule is adjusted in such a way that the devices cooperate in a distributed service choreography without a centralized point of control. Advanced RTOS features are thus not required but can be leveraged to provide additional quality of service (QoS) levels beneath the critical RT task. Multicore CPUs are not explicitly modeled but could be represented by pinning a separate schedule to each core. Timing side effects from parallel execution are outside of the scope of our current architecture.

Because we need to provide real-time guarantees for the control loop in a distributed system, the network also needs to offer these guarantees. We therefore assume all devices in the control loop are connected by a real-time capable network with bounded message delays. The predominant message exchange mode in industrial control applications is cyclic; thus, we also assume a cyclic communication model. In this model, each network cycle is divided into a number of timeslots that are assigned to a device, i.e., TDMA. We do not assume a master-slave relationship on the system or network level. Each device can potentially send data to any other device in the network. The tight time-synchronization required for distributed service choreographies are also needed by the TDMA-network so there is no additional overhead.



(a) Task in the embedded application

(b) Workflow layout and global timing constraints

(c) Task assignment and task WCET on assigned machines

(d) Local timing constraints as input for scheduling algorithm

(e) Simulation of a three machine schedule for the workflow in Figures (a) - (d), generated by heuristics. Grey blocks denote task execution, white blocks show time spent waiting for messages from preceding tasks. Hatched blocks show the TDMA-slots.

Fig. 2: A complex workflow in an adaptive cruise control scenario

Figure 2 depicts an overview of the planning steps necessary in our architecture. We chose an adaptive cruise control system as an example. In this example, a 3D-vision system is used together with a radar system to measure the distance to vehicles in front of the object vehicle and regulate vehicle acceleration and deceleration accordingly. The resulting workflow is shown in Figure 2a. The global deadline and period of the workflow are derived from physical requirements and / or control theory. Once the automation task has been modeled according to the modular decomposition (Figure 2a), these global deadlines are attached to the workflow (Figure 2b). Because embedded systems often require platform specific implementations for each functional module and the modules themselves make use of sensors and actuators, the assignment of jobs to machines can be viewed as a design-time decision performed by a skilled engineer. For any given assignment, the worst case execution time (WCET) of each task in the workflow can be measured or estimated. This estimation leads to the situation shown in Figure 2c where the global deadline and period of the workflow are known and the machine placement and WCET of each workflow task have been determined. Afterwards, the heuristics take over: a deadline assignment algorithm (Section IV-A) can be used to generate local constraints (Figure 2d), which are then used to generate a matching task ordering. Alternatively, a suitable distributed scheduling algorithm (Section IV-B) can directly determine the task ordering. In the final step, the system is verified through discrete event simulation, the output of this simulation step for our example is shown in Figure 2e. The fundamental difference between the planning and the execution phase of the control loop is important: whereas planning and scheduling are performed in a centralized, offline fashion, the execution of the workflow is distributed without a central point of control.

In communicating systems with tight timing requirements, the network configuration plays an essential role in finding valid schedules. We cannot simply place an upper limit on the communication delay and add it to the WCET of each task as this action would prevent us from finding a feasible schedule in Figure 2e and many other situations. Instead, timing information about each individual TDMA slot has to be considered during the schedule synthesis. As shown in our example, the slots may be distributed irregularly. For example, when an application is sharing the same communication medium with a legacy application and was granted only the previously unused time slots. Another example would be communication protocols, such as Flexray, which set aside a portion of each cycle for lower priority traffic. We therefore consider the available TDMA slots as an input to our schedule synthesis instead of searching for a suitable slot assignment for a given schedule. Once a candidate service choreography has been generated it should be verified in terms of its functional and temporal correctness. We use a simulation environment to achieve this task. To verify temporal correctness, only the temporal characteristics of each job on its assigned machine are required (i.e., its WCET). This allows for an automated generation of a simulation model by the rtSOA planner. The schedules generated by the planner are simulated on the infrastructure described by its input and are checked for deadline violations. Compared to our previous work [12], we follow the same approach as outlined in Figure 2, but through the use of more advanced scheduling algorithms, we are able to generate a higher number of feasible schedules (Section V). Another benefit of directly using distributed scheduling algorithms (Section IV-B) after the state shown in Figure 2c is the elimination of step 2d. The heuristics will directly generate schedules for the step shown in Figure 2e.

IV. HEURISTICS

This section provides an intuitive, high level overview of the heuristics employed in our evaluation. Our previous work [13] identified Potts' heuristic [11] as the best choice for scheduling after the use of a deadline assignment heuristic as described in Section IV-A. Non-preemptive Earliest Deadline First (EDF) is a good alternative. This second step is not required when using a distributed scheduling heuristic as described in Section IV-B.

The problem is modeled as a directed, acyclic graph of tasks which we call a workflow. The edges in the graph represent data flow between individual tasks. Each task is annotated with a worst case execution time (WCET). The range of possible start and completion times of the task can be constrained by setting a task release time and deadline. Each workflow is annotated with a global deadline and the period after which the workflow is repeated.

The overall goal is to find a suitable task ordering for a given assignment of tasks to machines. Tasks on a single machine cannot overlap or be preempted. This task ordering must fulfill all local deadlines (on the task level), release times and global deadlines (on the workflow level). A task cannot be started before all transitively preceding tasks have been completed.

A. Deadline Assignment Heuristics

In previous work [12], we determined that deadline assignment heuristics derived from the work of Kao and Garcia-Molina [9] are effective methods for our application. The three best heuristics were the Proportional Deadline (**PD**), Equal Slack (**EQS**) and Equal Flexibility (**EQF**) heuristics. All three heuristics work by grouping tasks into levels by the maximum hop distance from either the sources of a DAG (EQS, EQF) or its sinks (PD). The PD heuristic then simply assigns deadlines proportional to the level of each task. For example, in a graph with 5 levels, each level would be assigned 20% of the available run time. In contrast, the EQS and EQF heuristics distribute slack, defined as the difference between the sum of the task's execution time and the overall deadline of the workflow-DAG, between the levels. EQS assigns an equal amount of slack to each level in the DAG whereas EQF scales the assigned slack by the weight of the execution time of each level. With EQF, levels with longer WCETs get more of the slack.

We modified these heuristics further by introducing a corrective factor for communication over the TDMA network. In a first step, the mean communication delay μ is calculated for each machine in the TDMA network. This number specifies the average time a task has to wait for the next available TDMA slot on a given machine. If any task in a level has a successor on a different machine, the maximum μ of the communicating machines is added as a "hidden" level representing a virtual processing time. The TDMA-modified Equal Slack (**EQS-TDMA**) and TDMA-modified Equal Flexibility (**EQF-TDMA**) heuristics then allocate the slack between the normal, non-hidden, levels the same way as their non-modified counterparts. This method leads to an improved performance of the heuristics as explained in Section V.

We also propose deadline assignment based on the Earliest Release and Latest Finish Time (**ERT-LFT**) of each task. The earliest release time (ERT) of each task is determined by traversing the DAG from its sources and calculating the minimum time at which all previous tasks have finished and, if the tasks are located on other machines, have sent their data via the next available TDMA-slot. Similarly, the latest finish time (LFT) is obtained by attaching a deadline to each task such that its successor can finish during the global workflow deadline. In the ERT-LFT heuristic, the ERT is used as the release time of a task while the LFT is used as its deadline.

B. Distributed Scheduling Heuristics

Scheduling algorithms for tasks with communication usually comprise two different phases [22]. First, a task selection phase, also called the prioritization phase, which determines which task should be scheduled when. The second phase, a processor selection phase, determines the processor on which the task should be executed. In our scenario, the processor selection is fixed *a priori*, which means we focus on the task ordering mechanism of each heuristic.

The Heterogeneous Earliest Finish Time (**HEFT**) [24] and Dominant Sequence Clustering (**DSC**) [26] heuristics are examples for list-scheduling algorithms that maintain a fixed priority list of tasks that is calculated once. Both heuristics use the length of the longest path from a task t, including the communication times, to a sink of the DAG for their task prioritization. We will use $exit(t)$ to denote this path. HEFT simply ranks tasks by increasing $exit(t)$. The DSC heuristic uses the ERT of t plus $exit(t)$ as the priority of t.

The Mobility Directed (**MD**) [25] heuristic chooses tasks based on their mobility, defined as the difference between a task's LFT and its earliest start time (EST), divided by the task's WCET. Although the EST is similar to the ERT, it is recalculated after each task selection and takes into account the scheduling time of the other workflow tasks.

Earliest Task First (**ETF**) [23] picks a task among the ready tasks, meaning tasks whose predecessors have already been scheduled, by choosing the task with the minimum EST. Ties are broken by the task with the smallest LFT minus WCET.

We propose two additional scheduling heuristics: an adapted version of **Potts'** heuristic [11] that works in a distributed environment with fixed task-placement, and the Least Delay heuristic (**LD**).

The adapted Potts' heuristic is set up by picking the task with the minimum LFT from the available ready set, meaning the current time on the task's machine is equal or greater than the task's EST. This initial step is also known as Schrage's heuristic [11]. In most cases, this initial step yields valid schedules, meaning the DAG sinks do not violate the workflow deadline in the schedules. If the first pass of Schrage's heuristic was unsuccessful, Potts' heuristic then analyzes the resulting schedules and looks at task A, called the critical task, violating its LFT. This means there could be another task B with a smaller EST than A but with a larger latest starting time (LST) scheduled before A because A was not ready at that moment.

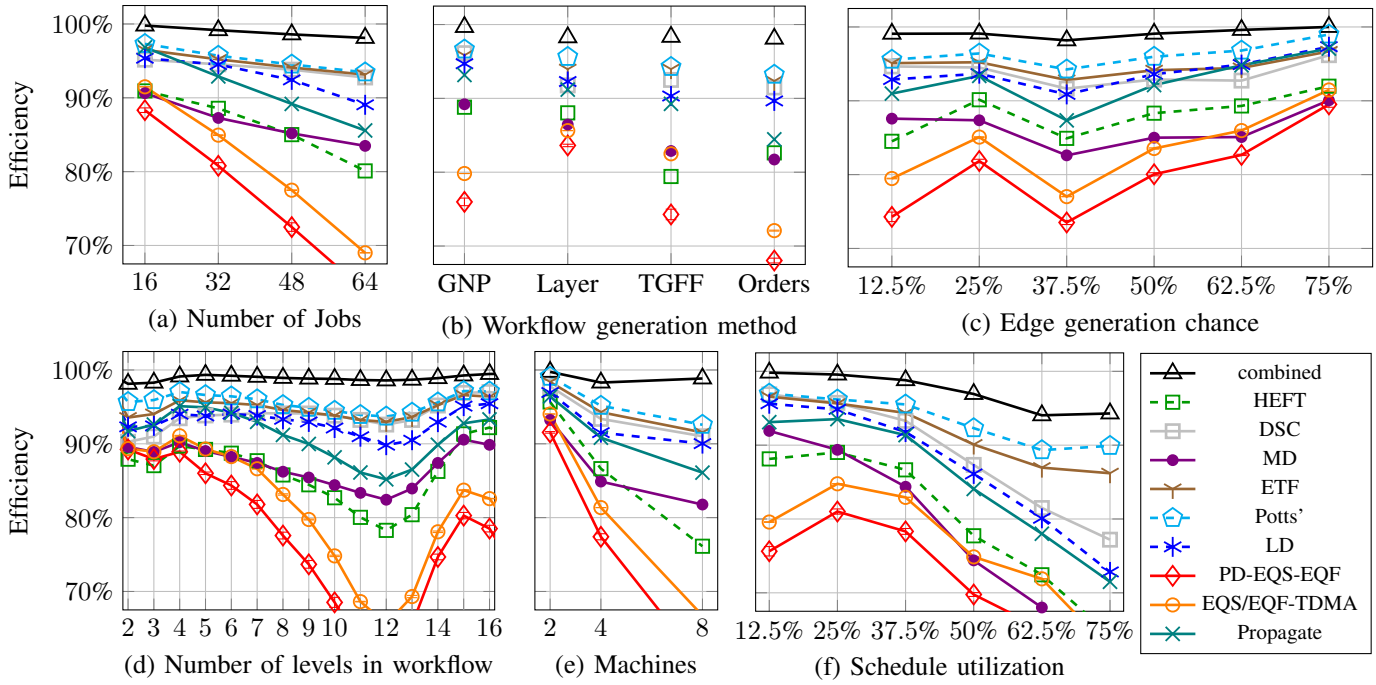


Fig. 3: Evaluation of deadline assignment heuristics

If such a task B, also called the interference task, exists on the same machine as A, we introduce an additional edge in the DAG from A to B to ensure that A will be scheduled before B. If no such task exists on the same machine as the critical task, we look for an interference task B' on a different machine. We then take B' as the new critical task and try to locate an interference task C on the same machine as B'. If C exists, we introduce an additional edge from C to B' and continue as previously described. The modified workflow is then rescheduled with Schrage's heuristic.

The LD heuristic tries to determine the implications of scheduling each task in the ready set. The heuristic schedules each of the tasks in the ready set in a "what-if" manner and determines the EST of all tasks in the DAG based on this speculative scheduling. The resulting EST of each sink task is compared to its EST before the speculative scheduling, yielding a value for the expected $delay_t$ of the sink task t. The maximum $delay_t$ yields the delay for the entire workflow. The heuristic now chooses the task from the ready set that resulted in the minimum workflow delay. Naturally, this heuristic has a high run-time complexity, but our evaluation (Section V) shows it can generate solutions in many cases where the other heuristics failed to do so.

V. Evaluation

Since industrial use cases span a wide range of potential layouts of the resulting task graphs, we rely on synthetic benchmarks which are based on several well-known graph generation methods [27]. The **Erdős-Rènyi** $G(n, p)$ method generates an unbiased DAG out of all possibilities. Therefore, 50% of the total number of test cases were generated with this method. The **Layer-by-Layer** method allows specifying the maximum depth of the graph and was developed specifically for validation of scheduling algorithms. 20% of the test cases were generated with this method. Similarly, **Task Graphs for Free** (TGFF) is another method of generating task graphs for the validation of scheduling methods. Another 20% of the test cases were generated with TGFF. The **Random Orders** method generates a partial order (i.e. a DAG) by intersecting several total orders, which are constructed by shuffling the nodes of the graph. This method generates graphs with all transitive edges and was used for generating the last 10% of the test cases.

We generated a total of 1 200 000 *feasible* workflows using the aforementioned ratios with either 16, 32, 48 or 64 tasks which were randomly assigned to either 2, 4 or 8 machines, which were connected via TDMA. The feasibility of the workflows was verified through a MIP-solver[1], the details of the problem formulation are given in our previous work [12]. To avoid bias in the evaluation, the amount of feasible workflows is largely the same for each combination of machine count and task count, i.e. there are 100 000 workflows for each combination. The workflows all have a common global deadline and period of 10 ms, which is also the length of a TDMA round. TDMA slots of 120 μs length (the time needed to transmit 1500 bytes, which is the largest allowed UDP packet size, over 100 Mbit Ethernet) were assigned in round-robin style to the machines. The workflows were then run through the heuristics pipeline, described in Section III, to determine the percentage of feasible solutions to be found by a heuristic. We call this measure the *efficiency* of the heuristic. A value of 100% means a heuristic was able to solve all of the same problems that the MIP-solver determined as feasible, a value of 50% means half of the problems were solved. The efficiency of the MIP-solver is 100%, therefore it is not shown in the performance plots below.

[1]Gurobi, version 6.0 (http://www.gurobi.com/)

TABLE I: Cross Evaluation of Heuristics[2]

	EQS	EQF	PD	EQS_TDMA	EQF_TDMA	Propagate	HEFT	DSC	MD	ETF	Potts'	LD
EQS	-	better: 4.34% worse: 4.28%	better: 6.59% worse: 7.37%	better: 6.75% worse: 2.61%	better: 6.74% worse: 2.62%	better: 15.8% worse: 1.37%	better: 12.5% worse: 3.01%	better: 18.2% worse: 0.84%	better: 12.5% worse: 2.54%	better: 18.7% worse: 0.58%	better: 19.1% worse: 0.46%	better: 17.3% worse: 1.13%
EQF	better: 4.28% worse: 4.34%	-	better: 6.57% worse: 7.40%	better: 6.73% worse: 2.64%	better: 6.72% worse: 2.66%	better: 15.8% worse: 1.38%	better: 12.5% worse: 3.04%	better: 18.2% worse: 0.85%	better: 12.5% worse: 2.55%	better: 18.6% worse: 0.58%	better: 19.0% worse: 0.46%	better: 17.3% worse: 1.13%
PD	better: 7.37% worse: 6.59%	better: 7.40% worse: 6.57%	-	better: 9.49% worse: 4.57%	better: 9.48% worse: 4.58%	better: 16.9% worse: 1.68%	better: 12.6% worse: 2.27%	better: 18.9% worse: 0.73%	better: 14.0% worse: 3.28%	better: 19.7% worse: 0.81%	better: 20.1% worse: 0.68%	better: 18.7% worse: 1.26%
EQS_TDMA	better: 2.61% worse: 6.75%	better: 2.64% worse: 6.73%	better: 4.57% worse: 9.49%	-	better: 3.36% worse: 3.38%	better: 12.2% worse: 1.87%	better: 9.50% worse: 4.10%	better: 14.4% worse: 1.13%	better: 9.60% worse: 3.71%	better: 14.8% worse: 0.84%	better: 15.1% worse: 0.68%	better: 13.7% worse: 1.65%
EQF_TDMA	better: 2.62% worse: 6.74%	better: 2.66% worse: 6.72%	better: 4.58% worse: 9.48%	better: 3.38% worse: 3.36%	-	better: 12.2% worse: 1.87%	better: 9.51% worse: 4.08%	better: 14.4% worse: 1.12%	better: 9.62% worse: 3.70%	better: 14.8% worse: 0.84%	better: 15.2% worse: 0.68%	better: 13.7% worse: 1.64%
Propagate	better: 1.37% worse: 15.8%	better: 1.38% worse: 15.8%	better: 1.68% worse: 16.9%	better: 1.87% worse: 12.2%	better: 1.87% worse: 12.5%	-	better: 3.19% worse: 8.16%	better: 5.45% worse: 2.51%	better: 3.29% worse: 7.78%	better: 5.02% worse: 1.40%	better: 5.21% worse: 1.09%	better: 5.35% worse: 3.64%
HEFT	better: 3.01% worse: 12.5%	better: 4.04% worse: 12.5%	better: 2.27% worse: 12.6%	better: 4.10% worse: 9.50%	better: 4.08% worse: 9.51%	better: 8.16% worse: 3.19%	-	better: 8.76% worse: 0.85%	better: 6.15% worse: 5.66%	better: 9.80% worse: 1.21%	better: 10.2% worse: 1.14%	better: 9.23% worse: 2.54%
DSC	better: 0.84% worse: 18.2%	better: 0.85% worse: 18.2%	better: 0.73% worse: 18.9%	better: 1.13% worse: 14.4%	better: 1.12% worse: 14.4%	better: 2.51% worse: 5.45%	better; 0.85% worse: 8.76%	-	better: 1.16% worse: 8.58%	better: 3.00% worse: 2.32%	better: 3.37% worse: 2.19%	better: 3.00% worse: 4.23%
MD	better: 2.54% worse: 12.5%	better: 2.55% worse: 12.5%	better: 3.28% worse: 14.0%	better: 3.71% worse: 9.60%	better: 3.70% worse: 9.62%	better: 7.78% worse: 3.29%	better: 5.66% worse: 6.15%	better: 8.58% worse: 1.16%	-	better: 8.97% worse: 0.87%	better: 9.58% worse: 0.97%	better: 8.65% worse: 2.45%
ETF	better: 0.58% worse: 18.7%	better: 0.58% worse: 18.6%	better: 0.81% worse: 19.7%	better: 0.84% worse: 14.8%	better: 0.84% worse: 14.86%	better: 1.40% worse: 5.02%	better: 1.21% worse: 9.80%	better: 2.32% worse: 3.00%	better: 0.87% worse: 8.97%	-	better: 1.45% worse: 0.95%	better: 2.63% worse: 4.54%
Potts'	better: 0.46% worse: 19.1%	better: 0.46% worse: 19.0%	better: 0.68% worse: 20.1%	better: 0.68% worse: 15.1%	better: 0.68% worse: 15.2%	better: 1.09% worse: 5.21%	better: 1.14% worse: 10.2%	better: 2.19% worse: 3.37%	better: 0.97% worse: 9.58%	better: 0.95% worse: 1.45%	-	better: 2.22% worse: 4.63%
LD	better: 1.13% worse: 17.3%	better: 1.13% worse: 17.3%	better: 1.26% worse: 18.2%	better: 1.65% worse: 13.7%	better: 1.64% worse: 13.7%	better: 3.64% worse: 5.35%	better: 2.54% worse: 9.23%	better: 4.23% worse: 3.00%	better: 2.45% worse: 8.65%	better: 4.54% worse: 2.63%	better: 4.63% worse: 2.22%	-

[2]Each cell describes the percentage of cases where the heuristic from the header led to a feasible solution when the one from the leftmost column did not.

The performance of the heuristics plotted against different metrics is shown in Figure 3. In addition to the individual heuristics, the combined metric (—△—), meaning the percentage of workflows solved by at least one of the heuristics, is also depicted in this figure. We have combined the EQS, EQF and PD heuristics into a single plot line (—◇—) because they show very similar performance. The standard deviation is shown as small error bars in this plot. We also combined the EQS-TDMA and EQF-TDMA measurements (—○—) for the same reason. Overall, the soft real-time deadline assignment heuristics (—◇—) are the worst performing methods. Introducing hidden levels for the EQS-TDMA and EQF-TDMA heuristics (—○—), as described in Section IV-A, offers roughly a 5% increase in efficiency but does not make the heuristics competitive. Our proposed ERT-LFT heuristic (—×—) is the best-performing deadline assignment heuristic in our evaluation and consistently outperforms the MD (—●—) and HEFT (- □ -) scheduling heuristics in terms of efficiency. However, the ERT-LTF heuristic is only competitive with more effective heuristics, such as the adapted Potts' heuristic (- ⬠ -), for easier problems with a smaller number of jobs in the workflow or a small number of machines in the network (Figure 3 a and b).

In general, the difficulty of synthesizing schedules for workflow and device combination is determined by the number of jobs in the workflow (Figure 3 a), the number of machines in the network (Figure 3 b) and the average schedule utilization of the machines after the workflow tasks have been assigned to them (Figure 3 f). The combination of heuristics (—△—) still shows a high overall efficiency when the number of machines or the number of jobs in the workflow is increased. By combining several heuristics the decreasing efficiency of the individual heuristics in the aforementioned cases can be compensated because each heuristic is exploring different areas of the total search space. This is true to a lesser degree for problems with high schedule utilization, as even a single wrong task ordering can invalidate the overall schedule in such cases. Problems with a lower utilization offer more flexibility in this regard. However, the efficiency of combined heuristics is over 93% even for those cases with the highest utilization.

There was a difference between the three best performing heuristics: ETF (—⋎—) and Potts' (- ⬠ -) are more capable of dealing with high-load cases than the DSC heuristic (—□—) which shows comparable performance in most other cases (Figure 3 f). The LD heuristic (- ✱ -) is often close to this group but does not perform as well with larger workflow sizes. The amount of edges in a workflow has little influence on the performance of the heuristics (Figure 3 c). Realistic examples are probably closer to the lower end of the edge-percentage scale: 100% would denote a fully connected DAG with $n * (n-1)$ edges for a DAG with n nodes. The combination of heuristics also has stable performance, regardless of the number of levels in a workflow (Figure 3 d). The shape of the workflow graph has little influence on the efficiency of the heuristics overall. The graphs generated by the layer-by-layer method are more difficult for most heuristics, however Potts' heuristic (- ⬠ -) is able to compensate for this, leading to a continued high efficiency of the combined metric (Figure 3 b).

Table I shows a pairwise comparison of all heuristics on the entire benchmark set. Even the best heuristics (Potts' or ETF) are unable to find a solution for the scheduling problem in some cases were the overall worst heuristics (EQS, EQF or PD) are successful. This fact is the basis for our approach which uses a combination of different heuristics in succession. This also justifies the inclusion of the LD heuristics with a relatively high run-time cost (Table II). The LD heuristic, however, explores a different area of the search space than many other heuristics, meaning it has the highest number of unique solutions (4363) generated by any heuristic. DSC and Potts' follow in second and third place with 2876 and 2393 unique solutions, respectively. ETF and ERT-LFT also have a relatively high number of unique solutions with 1206 and 1197, respectively. The remaining heuristics have between 400 to 200 unique solutions while EQS and EQF only have 83 and 81, respectively, unique solutions.

The use of heuristic always implies a trade off of better run-time characteristics for reduced optimality. Table II shows the geometric mean of the heuristics' performance as well as the run time of the Gurobi MIP-solver. The measurements were

TABLE II: Geometric Means of the Heuristic Runtimes

	EQS	EQF	PD	EQS-TDMA	EQF-TDMA	Propagate	HEFT	DSC	MD	ETF	Potts'	LD	Gurobi
16 Tasks	0.04 ms	0.04 ms	0.04 ms	0.06 ms	0.06 ms	0.14 ms	0.10 ms	0.12 ms	0.24 ms	0.12 ms	0.19 ms	0.77 ms	7.15 ms
32 Tasks	0.10 ms	0.09 ms	0.09 ms	0.12 ms	0.12 ms	0.52 ms	0.22 ms	0.28 ms	0.64 ms	0.38 ms	0.65 ms	4.11 ms	57.7 ms
48 Tasks	0.16 ms	0.16 ms	0.16 ms	0.20 ms	0.19 ms	1.23 ms	0.41 ms	0.53 ms	1.13 ms	0.69 ms	1.49 ms	11.2 ms	196 ms
64 Tasks	0.24 ms	0.24 ms	0.24 ms	0.28 ms	0.28 ms	2.37 ms	0.66 ms	0.80 ms	1.67 ms	1.09 ms	2.78 ms	23.5 ms	486 ms

obtained on an Intel Core i7-3930K. It is apparent that the heuristics outperform the MIP-solver by two to three orders of magnitude. Even heuristics with a higher run-time complexity, such as the LD heuristic or Potts' heuristic, have a significantly shorter run time than the solver. This situation supports our argument to combine heuristics. The cumulative run time distribution of the DSC and Potts' heuristic as well as the MIP-solver is shown in Figure 4. DSC's run time only varies in a narrow band since it is a relatively simple list scheduling algorithm. Contrastingly, Potts' run time varies more because it possibly performs multiple scheduling iterations. The solver has the largest spread, ranging from under $50\,ms$ to over $100\,s$ for 64 job workflows, because it explores the entire possible state space.

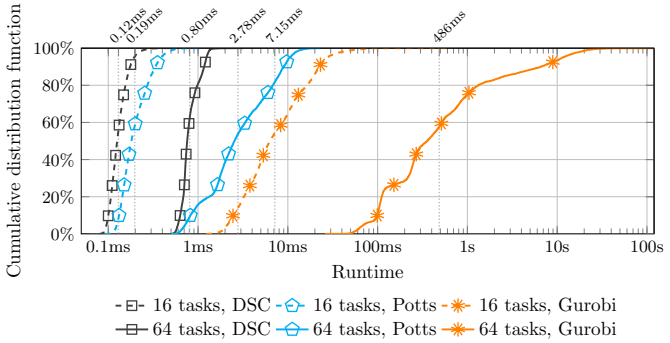


Fig. 4: Cumulative run time distribution

VI. Conclusions and Future Work

We have presented a heuristics-based approach for generating schedule-based service choreographies for hard real-time control loops without a central point of control. Our evaluation shows that a combination of heuristics can solve over 99% of 1.2 million test cases. The heuristics proved to be, on average, two to three orders of magnitude faster than an approach based on an MIP satisfiability solver. Hence, a combination of heuristics is a feasible choice when the run-time of the scheduling method is relevant. One such use case would be an interactive tool that supports engineers in deployment planning and run time budgeting. In future work, we will develop a prototype for such a tool. We will also evaluate the automatic allocation of tasks to machines.

References

[1] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg, "How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective," *WASET Int. J. Mech., Aerospace, Ind. and Mechatronics Eng.*, vol. 8, no. 1, Jan 2014.

[2] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, Feb 2005.

[3] P. Danielis, J. Skodzik, V. Altmann, E. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," in *ETFA*, Sept 2014.

[4] S. Voss and B. Schatz, "Deployment and scheduling synthesis for mixed-critical shared-memory applications," in *ECBS*, 2013.

[5] F. Jammes, B. Bony, P. Nappey, A. Colombo, J. Delsing, J. Eliasson, R. Kyusakov, S. Karnouskos, P. Stluka, and M. Till, "Technologies for SOA-based distributed large scale process monitoring and control systems," in *IECON*, 2012.

[6] C. Lerche, N. Laum, G. Moritz, E. Zeeb, F. Golatowski, and D. Timmermann, "Implementing powerful web services for highly resource-constrained devices," in *PERCOM Workshops*, March 2011.

[7] N. Kaur, C. McLeod, A. Jain, R. Harrison, B. Ahmad, A. Colombo, and J. Delsing, "Design and simulation of a SOA-based system of systems for automation in the residential sector," in *ICIT*, 2013.

[8] F. Jammes, H. Smit, J. Lastra, and I. Delamer, "Orchestration of service-oriented manufacturing processes," in *ETFA*, Sept 2005.

[9] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 12, 1997.

[10] J. Jonsson and K. G. Shin, "Robust adaptive metrics for deadline assignment in distributed hard real-time systems," *J. Real-Time Syst.*, vol. 23, no. 3, 2002.

[11] C. Potts, "Analysis of a heuristic for one machine sequencing with release dates and delivery times," *Operations Research*, no. 6, 1980.

[12] T. Kothmayr, A. Kemper, A. Scholz, and J. Heuer, "Synthesizing schedules through heuristics for hard real-time workflows," in *ICIT*, 2015.

[13] T. Kothmayr, A. Kemper, A. Scholz, and J. Heuer, "Machine ballets don't need conductors: Towards scheduling based service choreographies in a real-time SOA for industrial automation," in *ETFA*, 2014.

[14] OASIS, "Devices profile for web services specification (version 1.1)," http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01, July 2009.

[15] OPC Foundation, "OPC unified architecture (OPC UA) specifiations," http://www.opcfoundation.org/UA, 2008.

[16] G. Candido, F. Jammes, J. de Oliveira, and A. Colombo, "SOA at device level in the industrial domain: Assessment of OPC UA and DPWS specifications," in *INDIN*, July 2010.

[17] H. Bohn, A. Bobek, and F. Golatowski, "SIRENA - service infrastructure for real-time embedded networked devices: A service oriented framework for different domains," in *ICN/ICONS/MCL*, April 2006.

[18] J.-F. Martínez, M. López, V. Hernández, K. Jean-Marie, A.-B. García, L. López, C. Herrera, and C.-J. Sánchez-Alarcos, "A security architectural approach for DPWS-based devices," in *CollECTeR Ibéroamérica*, 2008.

[19] R. Checcozzo, F. Rusina, L. Mangeruca, A. Ballarino, C. Abadie, A. Brusaferri, R. Harrison, and R. Monfared, "RI-MACS: An innovative approach for future automation systems," *IJMMS*, vol. 2, no. 3, 2009.

[20] L. Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio, "SOCRADES: A web service based shop floor integration infrastructure," in *The Internet of Things*, ser. LNCS. Springer, 2008.

[21] G. Starke, T. Kunkel, and D. Hahn, "Flexible collaboration and control of heterogeneous mechatronic devices and systems by means of an event-driven, SOA-based automation concept," in *ICIT*, 2013.

[22] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM CSUR*, vol. 43, no. 4, Oct. 2011.

[23] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM J. on Compututing*, vol. 18, no. 2, Apr 1989.

[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, Mar 2002.

[25] M.-Y. Wu and D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 3, Jul 1990.

[26] T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 9, Sep 1994.

[27] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *SIMUTools*, 2010.