Machine ballets don't need conductors

Towards scheduling-based service choreographies in a real-time SOA for industrial automation

Thomas Kothmayr, Alfons Kemper
Chair for Database Systems,
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 industry is under pressure to increase the flexibility of its factory lines. One approach to achieve this goal is the shift from centralized control systems towards distributed, service-oriented architectures (SOA). To fully leverage the benefit of this new paradigm, the SOA should extend down to the device level and even include resource-constrained devices, such as smart sensors and actuators. In this paper, we present our approach for a lightweight distributed service choreography without a central point of control. It is based on network-aware precalculation of a static, non-preemptive schedule for each device and is thus suitable even for constrained devices. In contrast to previous work, our focus lies on the planning components required for achieving a service choreography. Since scheduling is a central part of our architecture, and we expect it to be executed many times during the planning process, we evaluate different heuristics for this task.**

I. Introduction

Manufacturing and process industries are under pressure from shortened product life-cycles, smaller lot sizes and demands for reducing the time to market for each new product. Traditional manufacturing plants do not scale well under these new requirements because the cost for installation and setup are constituting a considerable portion of their total life-cycle costs [1]. The problem stems from the inflexible design of traditional manufacturing systems: Centralized, monolithic and scan-based control systems are optimized for a given physical and network configuration. They are tightly coupled with their environment which hinders modularization and reuse.

The hardware side is already changing today: industrial Ethernet is gaining traction [2] while smarter embedded devices are performing an increasing number of orthogonal tasks, for example wireless sensor networks (WSNs) monitoring machine health. These developments shift the pressure to the software side where the traditional design paradigm of centralized control for the main automation tasks still reigns. IT outside of manufacturing often has to deal with similar issues and has responded by employing service-oriented architectures (SOAs) to integrate different systems into changing business processes. In industry, SOAs are already being introduced for non-critical functions outside of the main automation task, such as reporting and supervision as well as integration with higher level enterprise systems [3]. We propose a service-oriented architecture, called rtSOA, for the hard real-time control tasks handled by embedded devices. The control task is described by a workflow of discrete but interdependent

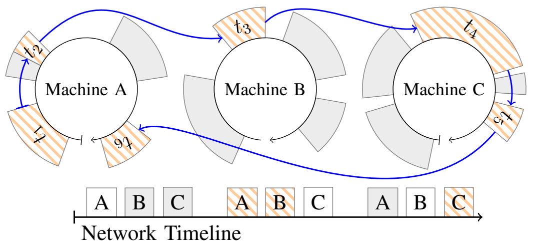

Fig. 1: A distributed workflow (striped, arrows indicate message flow) on three machines and a TDMA based network.

jobs. Timing restrictions on a per-job level are only derived from global constraints when binding a set of workflows to machines in a network. This approach allows for a separation of concerns: While specifying workflows, an engineer is freed from timing constraints from the hardware and network. To fully leverage the distributed control paradigm facilitated by the IoT model, devices should be able to follow a local schedule but achieve global cooperation without central coordination [4]. In SOA terms this is called a service choreography, which stands in contrast to service orchestration. Like dancers in a ballet, devices participating in a service choreography follow a local set of rules without a central entity overseeing the process. Devices in a service orchestration are managed by an orchestration server which is akin to a conductor guiding individual musicians in an orchestra. This would simply repackage the centralized model described above and incur similar problems. In our model only the planning is performed centrally. The output is a static, local schedule for each device. Devices cooperate in a distributed service choreography by executing these schedules locally and communicating over a deterministic real-time network, as illustrated in Figure 1.

Related work (Section II) has often followed the orchestration approach. Our architecture (Section III) is targeted at constrained embedded devices and must therefore deliver a lightweight service choreography that nevertheless is able to achieve hard real-time guarantees. From a suitable system specification (Section III-A) a planning process is triggered (Section III-B) which produces a cyclic, non-preemptive schedule for each device. The feasibility of this architecture hinges on the ability to quickly solve a large number of local scheduling problems. We therefore evaluate scheduling heuristics in Section IV. Our conclusions and open research questions are presented in Section V.

II. Related Work

A shift towards 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.

A. Industry

The two most prominent specifications concerning service orientation for industrial devices are the Devices Profile for Web Services (DPWS) [5] and the Object Linking and Embedding for Process Control Unified Architecture (OPC UA) [6]. Both are conceptually similar: They implement a SOA through Web Services and rely on built-in base services for discovery and service reservation. In both cases, SOAP over HTTP is the standard message binding and messages are either encoded with XML or in a binary format for increased performance. The main difference is in their intended target area. OPC UA is a service-oriented version of the original OPC architecture and its main mission is still to connect industrial devices to control and supervision applications [7]. It is therefore not directly aimed at the communication between the devices themselves and follows a client-server architecture. 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. It is aimed directly at the devices performing the automation task and therefore supports a peer-to-peer as well as a client-server architecture. DPWS is therefore more general-purpose whereas the application for OPC-UA is more specific. OPC-UA includes a dedicated Object Model whereas the data model in DPWS is not standardized. Cândido et. al [7] give a more detailed comparison of DPWS and OPC UA.

Our vision for rtSOA is more 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 cyclic hard real-time control tasks instead of general purpose discovery and eventing like DPWS. To ensure that rtSOA can provide the required real-time guarantees we are relying on an offline planning phase which will generate a fixed resource reservation for each device. Discovery and especially device description (see Section III-A) could be implemented following the DPWS specification. Thus, rtSOA can be seen as orthogonal to industry standards like DPWS.

B. Academia

A large body of work has been published around service-oriented architectures for the automation domain, including work performed in several EU programs. The SIRENA project provided the first embedded DPWS stack [8] and the closely related SODA project extended the SIRENA framework by providing a toolkit for manageability, orchestration and security [9]. Together, these projects proved the feasibility of Web Services on embedded devices. The RI-MACS project uses DPWS compliant services for soft real-time and best-effort tasks [1]. For performance reasons, it opts for a separate communications stack besides 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 [3]. The AESOP project investigates the feasibility and limits of using a SOA-based approach inside control loops [10]. By implementing several prototypes, it closely investigates 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 are conceptually similar projects. AESOP showed the feasibility for integrating embedded devices in a control loop through a SOA. Our work therefore argues that message exchange in a control loop is possible by leveraging one of the protocol stacks investigated by AESOP [10]. Additionally, work performed inside the AESOP project has already pointed out that a distributed, choreography-based approach to SOA is preferable to the classical orchestration-based approach [4]. Our work focuses on the central planning required to achieve performant, hard real-time choreographies for devices in a tight control loop.

Research efforts outside of the mentioned EU projects include the RT-Llama [11] iLAND [12] middlewares for a real-time SOA. As our work, RT-Llama also follows a reservation-based approach and uses end-to-end workflow reservation to achieve timing guarantees. However, its execution model requires devices to have at least two cores [11] which makes RT-Llama not applicable to deeply embedded devices in industrial automation. In this domain, multicore architectures are neither always available, due to cost restrictions for small embedded devices, nor always desirable due to the increased difficulty multicores cause in timing analysis [13]. The iLAND middleware supports time-bounded reconfiguration in distributed soft real-time SOAs [12]. Apart from the focus on soft real-time systems, it is also conceptually similar to our approach because it deals not only with issues of service composability but also takes end-to-end timing constraints into account. The bounded-time online reconfiguration performed by iLAND is not an explicit goal of the rtSOA architecture, although our planning algorithms should be sufficiently fast to allow a smooth development flow. In contrast to iLAND, rtSOA is aimed at hard real-time systems.

III. System Architecture

The target platforms for our real-time SOA are devices in factory and process automation. This spans from large PC control systems to tiny embedded devices, such as smart sensors or actuators. We use the term smart device to describe a sensor or actuator that is 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.

Because our goal is to provide real-time guarantees in a distributed system, the network also needs to offer such guarantees. We therefore assume all devices SOA 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, we thus also assume a cyclic communication model. In this model, each network cycle is divided into a number of time-slots 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.

Based on these underlying assumptions, the main question that still needs to be answered is *"How can a hard real-time automation task be executed on a distributed IoT architecture with heterogeneous devices?"*. First, the automation task needs to be specified in such a way that all relevant information is available in the planning stage. We outline the specification process in Section III-A before detailing the steps taken in the planning phase in Section III-B.

A. Specification

There are two separate domains that need to be specified when deploying a SOA in the industrial context: the process and the environment in which it will be executed. The environment consists of the devices which partake in the automation process, the physical capabilities of each device (e.g., provide 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 standardization efforts for 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 deployment of the control cycle 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.

There are two obvious starting points for specifying the composition of services into automation tasks: Either from the domain of PLC languages or Web-Service composition standards. Many of today's automation processes are programmed following one of the five languages of the IEC 61131-3 standard. Of these the best conceptual matches to service composition are either sequential function charts (SFCs) or function block diagrams (FBDs). In enterprise IT, the Business Process Model and Notation (BPMN) is a leading standard for the graphical specification of service workflows and can be mapped to the Business Process Execution Language (BPEL). Neither domain provides a direct candidate for the modeling of distributed automation processes in a real-time SOA. BPMN is a better fit for the composition of services, but most execution models are aimed at a Web-Service orchestration

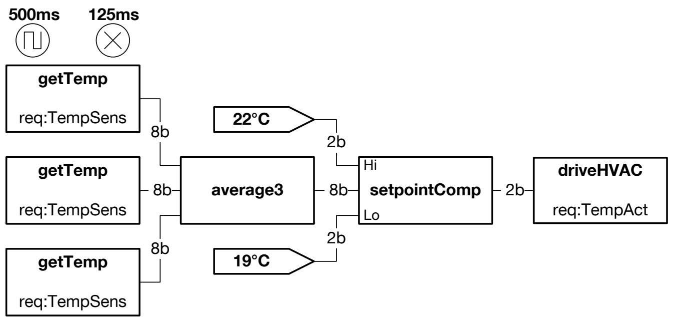

Fig. 2: Example workflow in annotated FBD. The workflow has a period of $500ms$ with a $125ms$ deadline. The connectors between blocks carry the message size in byte. The `setpointComp` block is initialized with constants.

instead of choreography. Additionally, BPMN lacks support for the timing constraints central to industrial automation and the conceptual overhead of the business domain is unnecessary for the automation domain. In our view, a service composition based on concepts from IEC 61131-3 SFCs or FBDs is more promising for the automation domain. Function blocks could be mapped to individual services and annotated with their physical requirements (e.g., for a sensor to be present on a device) and their worst case execution time (WCET) on the devices for which an implementation exists. The connections between blocks would need to be annotated with the maximum size of messages passed between two blocks. The FBD would finally be annotated by global timing requirements for the period and relative deadline of the workflow it contains. Figure 2 is giving an example of modeling a workflow with FBDs. In our current architecture we are placing one further restriction on the workflow structure: The workflow has to follow an in-tree structure, meaning that a service can have an arbitrary number of predecessors but only one successor. This is intended to facilitate easier analysis and planning because there is only one information sink in each workflow. Future work will asses whether this requirement can be relaxed to general directed acyclic graphs (DAGs).

B. Service Choreography

To turn the specification from the previous section into a runnable system that provides real-time guarantees more planning is required. The goal of the rtSOA system is to provide a service choreography, meaning that each device fulfills its part to cooperatively realize the automation task. The output of the rtSOA planning stage is a static, non-preemptive, cyclic schedule for each device. Powerful devices with advanced RTOS features are thus not required but can be leveraged to provide additional quality of service (QoS) levels beneath the critical RT task. Multiple CPUs or multicore CPUs on a single device are currently not modeled explicitly but could be leveraged by pinning a separate schedule to each core. Timing side effects from parallel execution [13] are still an active research area and outside of the scope of our current architecture. Figure 3 is giving an overview of the planning steps necessary in our architecture.

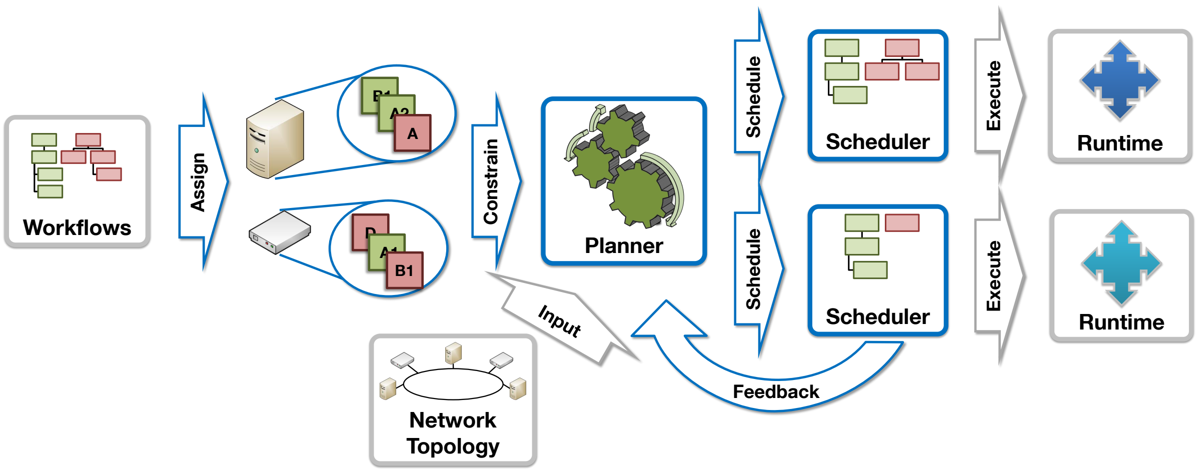

Fig. 3: Architecture overview

The first step in Figure 3 is the assignment of jobs to devices. This is performed by an engineer and the system only checks the validity of this assignment, i.e. whether or not all jobs are assigned and if a job's device offers all capabilities required by the job.

At this stage, each workflow is only annotated with end-to-end constraints: A global deadline for the workflow and a period. For a real-time service choreography, local constraints for each job must be derived. This planning step requires global knowledge and is centrally performed by the rtSOA planning component. rtSOA takes into account the position of a job in a workflow and the network slots assigned to the device on which the job is placed. From this information local deadlines and release times for each job in a workflow are created. This deadline assignment problem is part of our ongoing research.

The last step is the scheduling of the newly constrained jobs on each device. The scheduling problem is NP-hard but several good heuristics exist. Since we are only interested in finding a feasible schedule, the most important characteristic of a scheduling algorithm is which percentage of all feasible schedules it can find in a given time budget. In this respect, heuristics outperform exhaustive options, such as Linear Programming. We provide more details in Section IV. Should the heuristics fail in finding a feasible schedule, the deadline assignment algorithm from the previous step can assign more slack to jobs with violated deadlines to try and find an overall service choreography.

Once a candidate choreography has been generated it should be verified in terms of its functional and temporal correctness. We propose the use of a simulation environment to achieve this task. The simulator would have to simulate the devices involved in the choreography as specified by the network and device configuration. To verify temporal correctness, only the temporal characteristics of each job on its assigned machine are required (i.e., its WCET), so no complex system behavior needs to be modelled in the simulation. This allows for an automated generation of a simulation model from the rtSOA planner input and output: The schedules generated by the planner are simulated in the environment described by its input and are checked for deadline violations. By further refining the simulation, effects like the influence of jitter can be studied.

IV. Scheduling

Generating a static schedule for each device is an important step in our planning approach. We use schedulability as an admission test when assigning services to a device, in addition to generating a schedule as the basis for the rtSOA service choreography. Therefore we have two requirements for the scheduling algorithm that we employ: It must be able to find a feasible solution with a high probability and it must do so in a very short time. We compared heuristic approaches with a linear program (LP) solver in earlier work [14] and found that the heuristics significantly outperformed the LP approach in terms of efficiency, meaning their ability to find a feasible solution within a given time budget. We more closely examine the performance of heuristics for single machine scheduling of deadline, release time and in-tree precedence constrained tasks in this paper. The problem description has been taken from our previous work [14].

A. Formal Problem Description

The scheduling problem analyzed here is as follows: Find a feasible schedule for a set of jobs $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ with a fixed integer processing time $wcet_i$ on a single, non-preemptive machine. Additional constraints are: Each job j_i has either exactly one successor j_k, written as $j_i \prec j_k$, or no successor. A job may be annotated with a deadline d_i and a release time r_i. Since the generated schedule is expected to be executed cyclically, a maximum schedule length $\mathcal{D}$ is defined. It is enforced by the root element Ω with $wcet_\Omega = 0, d_\Omega = \mathcal{D}$ and $r_\Omega = \mathcal{D}$. If $\emptyset \prec j$ set $\Omega \prec j$, thus transforming the structure of $\mathcal{J} \cap \Omega$ into a single in-tree. If no values for r_i or d_i are given, 0 and $\mathcal{D}$ are assigned by default. The objective is minimizing the number of tardy jobs ΣU_j. We are only interested in finding a schedule that has no tardy jobs at all. In the traditional notation of scheduling theory we can express our problem as $1|r_j, \textit{in-tree}|\Sigma U_j$.

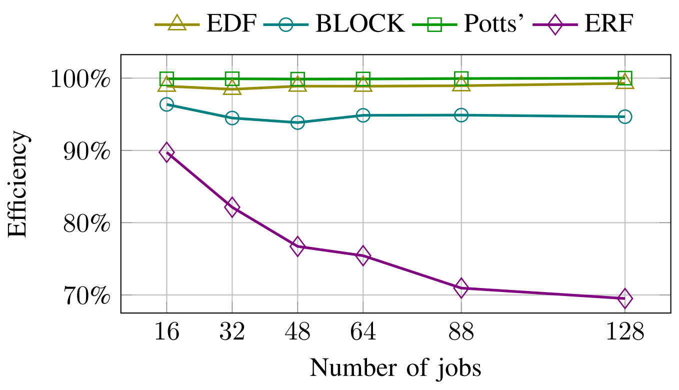

Fig. 4: Percent of test cases solvable by each method

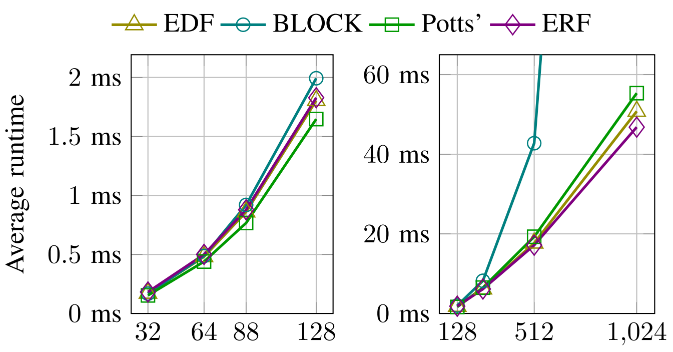

Fig. 5: Average run time of the heuristics

B. *Heuristic Approaches*

We compare earliest release time time first (ERF), earliest deadline first (EDF), the BLOCK heuristic [15] and Potts' algorithm [16].

Earliest release time time first: The ERF heuristic simply sorts all jobs in $\mathcal{J} \cup \Omega$ by ascending order of their release time. If two jobs have the same release time, then their deadlines are used as a tiebreaker. Precedence constraints should be mapped to modified release times by applying: $\forall j_i, j_k \epsilon \mathcal{J} \cup \Omega : j_i \prec j_k \implies r'_k = max\{r_k, r_i + wcet_i\}$

Earliest deadline first: In contrast to the ERF heuristic we cannot simply sort jobs by their deadline because that could lead to violation of precedence constraints. EDF instead chooses the leaf of the tree with the earliest deadline j_d and the leaf with the earliest release time j_r. If j_r can be scheduled before j_d without conflict, i.e., $r_r + wcet_r \leq r_d$, schedule j_r first, otherwise j_d. The scheduled leaf is then removed from the tree and if all predecessors of a job have been scheduled that job is then added to the set of available leaves. Effective deadlines for each job should be computed beforehand as: $\forall j_i, j_k \epsilon \mathcal{J} \cup \Omega : j_i \prec j_k \implies d'_i = min\{d_i, d_k - wcet_k\}$

BLOCK heuristic: The BLOCK heuristic [15] first sets up a schedule by ERF and divides it into blocks of jobs which are executed with no time delay between them. If the schedule is invalid, the heuristic adjusts the block by scheduling jobs with higher deadline towards the end of the block.

Potts' algorithm: Potts' algorithm [16] is setup by sorting tasks by their deadlines in topological order. If the schedule is invalid, it analyzes the critical sequences of the schedule, i.e., blocks of jobs where at least one job has an invalid deadline (= critical job j_{crit}). An interference job j_{int} is a job within a critical sequence that is scheduled before j_{crit} but has a higher deadline than d_{crit}. $r_{int} < r_{crit}$ must hold, since j_{int} would otherwise not have been scheduled this early. Interference jobs are thus scheduled after their corresponding critical jobs to reduce the amount of tardy jobs.

C. *Evaluation*

The evaluation was performed on randomly generated in-tree workflows with 16 to 128 jobs. The run time of the heuristics was additionally evaluated for up to 1 024 jobs. The amount of deadline and release time constraints per workflow is uniformly distributed between one and $|\mathcal{J}|$. Similarly, the schedule utilization ($\Sigma wcet_i / \mathcal{D}$) was uniformly distributed between zero and one. Since the goal of the evaluation is to evaluate the efficiency of a scheduling algorithm, i.e., for how many of the feasible workflows it can find a valid schedule, infeasible workflows had to be discarded first. As no optimal algorithm for the non-preemptive case exists we employ the preemptive least laxity first algorithm (LLF) to filter out definitely unschedulable workflows. LLF has been shown to be optimal for the preemptive single machine case [17]. Any workflow that is not schedulable in the preemptive case will remain so in the non-preemptive case. LLF discarded about half of the generated workflows as unsolvable, the remaining 20 503 jobs where then scheduled with each of the methods described in Section IV-B. For only two of these jobs no solution could be found with any of the employed methods, meaning that we have 20 501 jobs which comprise our set of feasible test cases. The test machines are equipped with an Intel Q6700 CPU at 2.66GHz and 8 gigabytes of RAM.

As shown in Figure 4, EDF, BLOCK and Potts' algorithm all perform well with over 95% efficiency on average. Potts' algorithm consistently performs at near 100% efficiency, there were only 18 out of 20 501 test cases where it did not find a solution, which equals an overall efficiency of over 99.9%. The combination of Potts' Algorithm, BLOCK and EDF finds a solution for all but 8 test cases. Concerning the average run time (Figure 5), all heuristics are performing well up until 256 jobs. The BLOCK heuristic is starting to take considerably longer afterwards because its worst case run time complexity is $\mathcal{O}(n^3)$ whereas its typical complexity is closer to $\mathcal{O}(n^2)$. The same is true for Pott's algorithm from a complexity standpoint, but it holds up better in terms of run time. EDF scales as expected from its complexity of $\mathcal{O}(n^2)$. ERF is actually underperforming in our implementation, considering its theoretical complexity of $\mathcal{O}(n\ log(n))$.

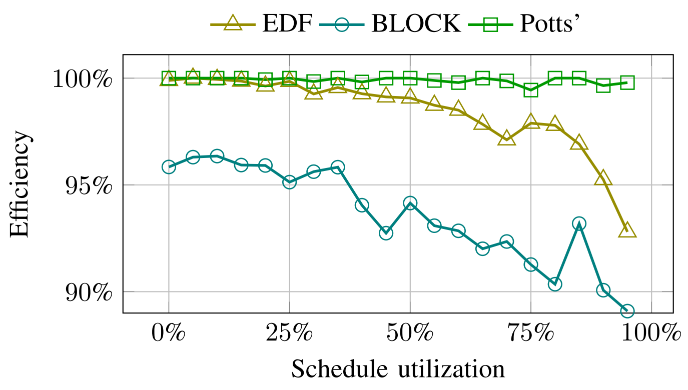

Fig. 6: Percentage of solvable schedules found plotted against schedule utilization. A utilization of 100% indicates that there is no idle time in the schedule.

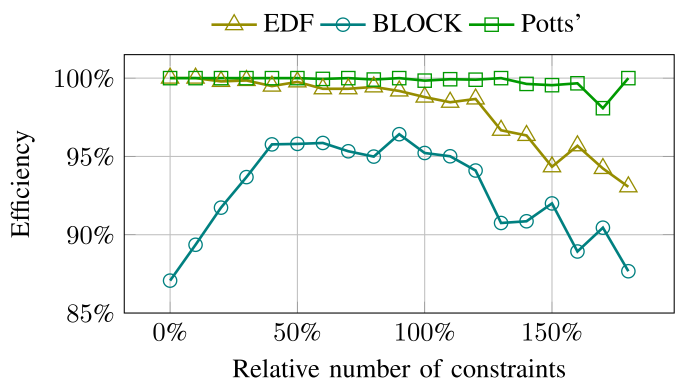

Fig. 7: Percentage of solvable schedules found plotted against percentage of constraints. The maximum amount of 200% and is reached when every node has a release time and deadline

Figure 6 shows that Potts' algorithm also performs very well as the schedule utilization increases. EDF generally performs well until 80% utilization whereas the BLOCK heuristic is showing decreased performance for more than 35% utilization. The picture is similar when looking at the amount of deadline and release time constraints attached to a workflow in Figure 7. The analysis is stopped after 165% because there where less than 50 workflows that could be aggregated into each data point from then on. Potts' performs very stable again whereas EDF is starting to drop of earlier. Interestingly, BLOCK performs worse with only a small number of constraints.

In conclusion, we will use Potts' algorithm for all scheduling purposes in the rtSOA planner. It consistently outperforms all other heuristics and has stable performance for large scheduling problems. This allows us to forgo the use of exhaustive search methods, such as LPs while still finding a feasible schedule in almost all cases. The EDF heuristic can be used as a second chance algorithm if Potts' is unable to find a schedule. The BLOCK heuristic could be used as a third try for smaller problem instances.

V. Conclusion and Future Work

We have presented our architecture for a real-time, service-oriented architecture for industrial automation. It is based on a distributed service choreography which is realized through explicit generation of a non-preemptive schedule. This schedule can be executed by constrained devices with very few requirements in terms of operating system support. We have shown that heuristics fulfill the requirements of our architecture both in terms of run time and the efficiency in finding feasible schedules. Our future work can therefore focus on another key element of the rtSOA planning process: automatically assigning local timing constraints to individual services so that all global timing constrains are fulfilled. We further plan to implement a demonstrator to show the feasibility of our approach in the real world. Further work will address the issue of reliability by generating choreographies with failover and adaptive service placements.

References

[1] 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," *International Journal of Mechatronics and Manufacturing Systems*, vol. 2, no. 3, 2009.

[2] T. Moore, "The world market for industrial ethernet and fieldbus technologies," IHS/IMS Research, Tech. Rep., 2013.

[3] 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 Berlin Heidelberg, 2008.

[4] 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.

[5] OASIS, "Devices Profile for Web Services Specification (Version 1.1)," http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01, July 2009.

[6] OPC Foundation, "OPC Unified Architecture (OPC UA) Specifications," http://www.opcfoundation.org/UA, 2008.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] M. Panahi, W. Nie, and K.-J. Lin, "The design of middleware support for real-time SOA," in *ISORC*, 2011.

[12] M. Garcia Valls, I. Lopez, and L. Villar, "iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems," *Industrial Informatics*, vol. 9, no. 1, Feb 2013.

[13] R. Wilhelm and J. Reineke, "Embedded systems: Many cores - many problems." in *SIES*, 2012.

[14] T. Kothmayr, J. Hirscheider, A. Kemper, A. Scholz, and J. Heuer, "Comparing heuristics and linear programming formulations for scheduling of in-tree tasksets," in *RTAS WiP*, 2014.

[15] Z. Liu, "Single machine scheduling to minimize maximum lateness subject to release dates and precedence constraints," *Computers & Operations Research*, vol. 37, no. 9, 2010.

[16] L. A. Hall and D. B. Shmoys, "Jackson's rule for single-machine scheduling: making a good heuristic better," *Mathematics of Operations Research*, vol. 17, no. 1, 1992.

[17] A. K. Mok and M. L. Dertouzos, "Multiprocessor scheduling in a hard real-time environment," in *Texas Conf. Comput. Syst.*, 1978.