Real-time Dynamic Reconfiguration for IEC 61499

Laurin Prenzel
Technical University of Munich
Munich, Germany
laurin.prenzel@tum.de

Simon Hofmann
Technical University of Munich
Munich, Germany
simon1.hofmann@tum.de

Sebastian Steinhorst
Technical University of Munich
Munich, Germany
sebastian.steinhorst@tum.de

***Abstract*—Reconfiguration is an important feature for industrial automation systems to provide flexibility, adaptability, and resilience. Dynamic reconfiguration of real-time systems requires both functional and temporal correctness. A lengthy disruption of real-time behaviors caused by a reconfiguration can cause the system to fail. We describe the scheduling problem for a component-based real-time industrial control system on the basis of the IEC 61499 and extend this problem to handle disruptions introduced by a dynamic reconfiguration. We show that the disruption can be quantified by applying the Priority Ceiling Protocol (PCP) and calculating the blocking time. Further, we show that the order of operations within a reconfiguration sequence can be optimized using the blocking time as objective function. An evaluation on two example systems shows that our model allows the application of common schedulability tests for Rate Monotonic scheduling. In our examples, the optimization reduces the blocking time of a task during reconfiguration compared to a heuristic topological ordering by up to 85%. This makes previously infeasible reconfigurations feasible. The results imply that timing analysis of dynamic reconfiguration for component-based real-time systems is attainable, and further research is necessary to extend it for distributed systems.**

***Index Terms*—Downtimeless System Evolution, Dynamic Software Updating, Dynamic Reconfiguration, Timing Analysis**

I. Introduction

Reconfiguration, and in particular dynamic reconfiguration, are important features for future industrial automation systems to be able to adapt to unpredictable events, faults, or attacks. While dynamic reconfiguration incurs lower costs by not requiring ramp-down and ramp-up phases, it comes with an additional risk for safety-critical real-time systems, since the safety and timeliness of the real-time process must be guaranteed. Yet, it may be the only option for systems that are uneconomic or impossible to restart. [1]

Recently, the process of dynamic reconfiguration or dynamic updating has seen attention in the topic of automatically generating safe update controllers [2], [3]. These controllers solve the problem of guaranteeing the functional correctness, which is by no means trivial. Nevertheless, just because a system can be updated consistently does not imply if the update can be applied in real-time. It has been shown that dynamic reconfiguration can cause significant disturbances to the real-time execution [4]–[6]. Unfortunately, the question if a dynamic update will succeed in time or not depends on a myriad of factors, most importantly: The system to be updated, the update to be applied, and the environment in which it runs. Particularly strict real-time constraints may not allow for the overhead introduced by a dynamic reconfiguration. Thus, evaluating the timeliness of a reconfiguration is a crucial step before a reconfiguration can be applied.

For industrial automation systems, the IEC 61499 provides a compelling framework for component-based distributed control applications [7]. The fractured application state and the concurrent execution facilitate the implementation of dynamic reconfiguration, yet this feature is rarely used and there is only limited tool support. One reason for this is that dynamic reconfiguration disturbs the real-time execution of critical tasks, and thus may compromise the safety of the system. While there are resources on real-time execution of the IEC 61499 [8]–[10], they do not address the timeliness of dynamic reconfiguration on a real-time control system. Further, current IEC 61499 execution environments do not fully support these strict real-time models, and have only limited support for dynamic reconfiguration.

In this paper, we model the execution of a component-based architecture, such as the IEC 61499, under preemptive Rate Monotonic (RM) scheduling with shared resources. This model allows us to incorporate the delay introduced by a sequence of reconfiguration operations into the schedulability condition. We derive an optimization problem which identifies a schedulable, optimal order of operations if there exists one. Two examples demonstrate that the optimization algorithm outperforms a heuristic algorithm by up to 85%. Our results indicate that dynamic reconfiguration of safety-critical real-time control systems is feasible and schedulability can be satisfied in architectures such as the IEC 61499 with strict timing constraints.

The system model and resulting scheduling problem are defined in Section II. We define the blocking duration and show its calculation in Section III. Section IV introduces the optimization problem of finding an optimal reconfiguration sequence, and the approach is demonstrated on two example systems in Section V. We outline related works in Section VI and Section VII concludes the paper.

II. System and Problem Definition

We first define our system model, emphasize the resulting scheduling problem that occurs during the reconfiguration, and argue about the timeliness of the reconfiguration. Our work builds on top of the IEC 61499 standard for distributed industrial control systems, but is applicable to other component-

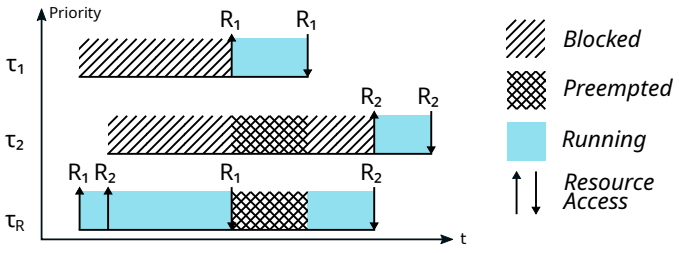


Fig. 1. A reconfiguration (τ_R) introduces additional blocking behavior within the real-time tasks (τ_1, τ_2). The blocking duration depends on the performed reconfiguration and must be assessed before the reconfiguration is applied.

based architectures as well. A version of the original scheduling problem was defined by [10].

A. System Model

The tasks of the system model are the distinct execution traces or event-chains within the component-based architecture. This is in line with previous models [10]. We model these tasks or traces as directed acyclic graphs (DAGs). The taskset

$$\tau = (\tau_1, \cdots, \tau_n) \tag{1}$$

consists of n tasks that *may* occur concurrently. Each task τ_i is characterized by (V_i, E_i, D_i), where the vertices $V_i = (e_{i,0}, \cdots, e_{i,j})$ are distinct executions of components, in our case IEC 61499 function blocks (FBs), the edges E_i are the precedence constraints between executions, and D_i is a deadline of the task that must be kept. Executions $e_{i,j} = (f_{i,j}, c^e_{i,j})$ are defined by the FB $f_{i,j}$ and a worst case execution time (WCET) $c^e_{i,j}$. FBs are shared resources that can only be used in one execution e at a time. The WCET $c^e_{i,j}$ depends on the algorithms, execution control, state, and the execution platform. We assume a fixed, known WCET for every execution.

Next, we model the reconfiguration sequence that modifies the application with a set of reconfiguration operations. The ordered sequence

$$S = \langle o_0, \cdots, o_n \rangle \tag{2}$$

consists of reconfiguration operations $o_i = (a_i, F_i, c^o_i)$, where a_i is an action, F_i is a set of affected FBs, and c^o_i is the WCET of the operation. The actions can interfere with the execution of a particular FB, e.g. a *stop* action will suspend all affected FBs until they receive a corresponding *start* action. A suspended FB is blocked from being executed and events must be preserved. This suspension is an important mechanism to guarantee the consistency of the reconfiguration [11]. It enables reconfiguration of an application from source to sink, which causes all events to be processed according to either the old version, or the new version, but not a mixture [3].

B. Scheduling Problem

The resulting scheduling problem is to determine whether all real-time tasks will satisfy their deadlines when the reconfiguration sequence is applied. By itself, this problem is not well defined, thus we base our analysis on the following assumptions:

1) There is only one single-threaded resource that executes both the tasks and the reconfiguration sequence using Rate Monotonic (RM) scheduling.
2) The resource can be preempted, but access to a FB is blocking, i.e. a FB can not be executed by a second task before the first task has released it.
3) Each real-time task τ_i has a unique rate T_i identical to its deadline D_i. It can not be triggered more frequently than its rate.
4) The reconfiguration sequence S is not a real-time task and has no deadline. Thus, it has the lowest priority and may be preempted by the real-time tasks. It is only executed once.

Given that the reconfiguration is not a real-time task, its impact can nevertheless not be neglected, since it may introduce blocking behavior that causes a priority inversion, when the reconfiguration suspends particular FBs. To mitigate the priority inversion, we utilize the Priority Ceiling Protocol (PCP), which elevates the priority of a task if it blocks a resource with a higher priority ceiling. This behavior is depicted in Figure 1, where the sequence S blocks tasks τ_1 and τ_2, because it requested their resources before the tasks were triggered. [12] defines a schedulability condition for n periodic tasks using PCP and preemptive RM scheduling:

$$\forall i, 1 \leq i \leq n, \quad \underbrace{\frac{C_i}{T_i}}_{\text{Execution}} + \underbrace{\frac{B_i}{T_i}}_{\text{Blocking}} + \underbrace{\sum_{j=0}^{i-1} \frac{C_j}{T_j}}_{\text{Preemption}} \leq \underbrace{i(2^{1/i} - 1)}_{\text{Max. Utilization}} \tag{3}$$

This condition assumes a taskset $\tau = \{\tau_1, \cdots, \tau_n\}$, where τ_n has the lowest priority and τ_1 has the highest priority. This is compatible with our system model, which we can sort in ascending order by their rate $T_i = D_i$ to fit the requirement. The WCET of a task τ_i with m executions can then be calculated as

$$C_i = \sum_{j=1}^{m} c_{i,j}. \tag{4}$$

The condition in Equation 3 consists of three components. The first component is the contribution of the tasks WCET to the overall utilization. The second is the contribution of blocking by other, lower-priority tasks to the utilization. Third is the cumulative preemption of tasks with higher priority. The sum of all components must be lower than the maximum utilization for a taskset of i tasks using preemptive RM scheduling.

In our model, the blocking time

$$B_i = B(\tau_i) = B^{\text{FB}}(\tau_i) + B^R(\tau_i) \tag{5}$$

has two contributors: $B^{\text{FB}}(\tau_i)$ is caused by shared access to the FBs, and $B^R(\tau_i)$ is a result of the concurrent execution of

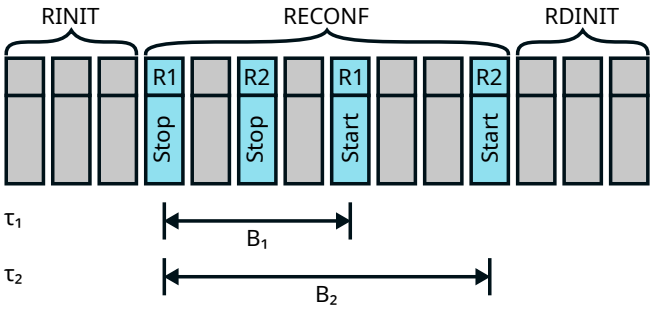


Fig. 2. The reconfiguration sequence S can be split into three phases: *RINIT*, *RECONF*, and *RDINIT* [13]. The blocking time of a task lasts while a resource with a higher priority ceiling is suspended.

the reconfiguration sequence S. Together, they determine how long a task is blocked by a lower-priority task due to the PCP.

Thus, the problem remains of determining the blocking durations $B^{\text{FB}}(\tau_i)$ and $B^R(\tau_i)$ for every task given a reconfiguration sequence. Once these values are known, the schedulability condition will decide whether or not the system is schedulable and can thus meet its deadlines. We show how the blocking durations can be determined in the next section. Afterwards, we propose an optimization problem to minimize the blocking time of each task by optimizing the order of reconfiguration operations.

III. Blocking Duration

The reconfiguration sequence will temporarily suspend FBs that may be used by other real-time tasks. This is a priority inversion, which can be solved using the Priority Ceiling Protocol (PCP). To satisfy the schedulability test in Equation 3, the blocking time $B(\tau)$ has to be determined. The contributors of $B(\tau)$ can be seen in Equation 5.

A. Function Block Blocking Time $B^{FB}(\tau_i)$

When two tasks share a FB, the higher priority task may have to wait until the lower priority task has released the FB. This execution can not be preempted, since this would lead to an inconsistent state. With $F(\tau_i)$ defined as the set of FBs used by task τ_i, this blocking time can be calculated as

$$B^{\text{FB}}(\tau_i) = \sum_{j=i+1}^{n} \left(\sum_{f_k \in F(\tau_j) \cap F(\tau_i)} c_{j,k} \right) \tag{6}$$

that is the sum of the WCET of all executions in tasks with lower priorities that use an FB that is also used in τ_i. More intuitively, it is the maximum duration a task may be blocked by another task with a lower priority, because they require access to the same FBs.

B. Reconfiguration Blocking Time $B^R(\tau_i)$

The blocking time $B^R(\tau_i)$ depends on the reconfiguration sequence S. An example is given in Figure 2. Every reconfiguration sequence can be split into three phases: An initialization phase *RINIT*, a critical phase *RECONF*, and a deinitialization phase *RDINIT* [13]. The critical *RECONF* phase begins when the first shared resource, in this case a FB, is stopped. It ends, when the last FB is started, and the execution of the real-time tasks can continue. During the *RDINIT* phase, leftover components are cleaned up. Using preemptive scheduling, *RINIT* and *RDINIT* do not disturb the real-time execution, since they can be prolonged indefinitely.

Intuitively, $B^R(\tau_i)$ is the duration of the sequence S during which there is a suspended FB with a higher priority ceiling than the priority of task τ_i. Formally, we first define the priority function

$$\pi^{\text{FB}}(f) = \max_{\tau_i | f \in F(\tau_i)} \pi^{\tau}(\tau_i), \tag{7}$$

which returns the priority ceiling of a FB f based on the priority $\pi^{\tau}(\tau_i)$, which can be assigned to each task according to its rate / deadline. We use the notation $S_{p,q} = \langle o_p, \cdots, o_q \rangle$ to represent the subsequence of $S = \langle o_0, \cdots, o_n \rangle$, where $0 \leq p \leq q \leq n$. We assume the existence of a suspension function $\text{sus}(S_{0,q})$, which returns the set of FBs that are suspended after the occurrences of the sequence $S_{0,q}$. This function must check if a FB was stopped or added during the sequence but not yet started. Then, the function

$$\pi^R(o_i) = \max_{f \in \text{sus}(S_{0,i})} \pi^{\text{FB}}(f) \tag{8}$$

calculates the priority ceiling of a reconfiguration operation as the maximum priority ceiling of any FB suspended during the occurrence of operation o_i. $\pi^R(o_i)$ decides whether operation o_i can preempt another real-time task. We can now define the blocking sequence

$$S_{\text{block}}(\tau_i) = \langle o_i \in S | \pi^R(o_i) \geq \pi^{\tau}(\tau_i) \rangle, \tag{9}$$

which defines the sequence of operations that will block task τ_i. From the point of view of τ_i, this is the disturbance of the reconfiguration S. Finally, the blocking time of τ_i,

$$B^R(\tau_i) = \sum_{o_j \in S_{\text{block}}(\tau_i)} c_j^o, \tag{10}$$

is calculated as the sum of the WCET of all blocking operations from the point of view of task τ_i.

For a taskset $\tau = (\tau_1, \cdots, \tau_n)$, where $D_1 < D_i < D_n$ and, thus, τ_1 has the highest priority and τ_n the lowest, the blocking times will follow the same order

$$B^R(\tau_1) \leq B^R(\tau_i) \leq B^R(\tau_n). \tag{11}$$

This is because any subsequence of S that blocks τ_i must also block τ_{i+1}, since $\pi^{\tau}(\tau_i) > \pi^{\tau}(\tau_{i+1})$.

C. Reconfiguration Feasibility

The original schedulability condition as proposed by [12] was given in Equation 3. By adjusting this condition, we define the laxity of a task τ_i as

$$L(\tau_i) = \underbrace{T_i i(2^{1/i}-1)}_{\text{Max. Utilization}} - \underbrace{T_i \sum_{j=0}^{i-1} \frac{C_j}{T_j}}_{\text{Preemption}} - \underbrace{C_i}_{\text{Execution}} - \underbrace{B(\tau_i)}_{\text{Blocking}}. \quad (12)$$

This laxity is the time that τ_i could execute longer while still keeping its deadline. For the system to be schedulable, the laxity must be positive for all tasks $\tau_i \in \tau$:

$$L(\tau_i) \overset{!}{\geq} 0. \quad (13)$$

We have shown how to compute the blocking time $B(\tau_i)$ and laxity $L(\tau_i)$ given a reconfiguration sequence S. This allows the application of the schedulability condition in Equation 3, which indicates whether the system can be scheduled with preemptive, single-threaded RM scheduling. In the next section, we demonstrate how the order of reconfiguration operations can be optimized to minimize the disruption of each task during the reconfiguration and to find a feasible sequence within the dependency graph.

IV. Optimization Problem

To achieve a consistent reconfiguration, specific precedence constraints have to be kept. This applies to the order in which components must be modified. If they are modified in the wrong order, events may be processed in unpredictable ways. For this reason, the system should be adapted from event source to event sink and the precedence constraints can be expressed in a dependency graph [3].

There are multiple solutions to find a sequence of operations that satisfies the precedence constraints which enable a consistent reconfiguration. A heuristic algorithm is presented in [3]. Each reconfiguration action is assigned a priority, and feasible operations are selected one-by-one. This approach is fast and efficient, but it does not take into account any timing constraints or deadlines and is vulnerable to priority inversion between operations.

In this paper, we have presented a schedulability condition to decide the timeliness of a reconfiguration sequence within a given taskset. We now present an optimization problem to find an optimal ordering of the reconfiguration operations using the previously defined schedulability condition. The overall search space is defined by the dependency graph. In Section V, we compare the results from our optimization with the heuristic solution.

A. Constraints

A valid reconfiguration sequence must satisfy both the functional and temporal correctness criteria. Functionally, the dependencies must be satisfied. Temporally, the system must remain schedulable. From Equations 12 and 13 the maximum feasible blocking time of each task can be computed as

$$B^{R,\max}(\tau_i) = T_i i(2^{1/i}-1) - T_i \sum_{j=0}^{i} \frac{C_j}{T_j} - B^{\text{FB}}(\tau_i). \quad (14)$$

Thus, the constraint

$$B^{R}(\tau_i) \leq B^{R,\max}(\tau_i) \quad (15)$$

enforces that any sequence must satisfy the schedulability condition in Equation 3.

We optimize the start times t_{o_i} of each operation o_i. For this purpose, there must be constraints to prevent overlapping executions and to enforce the dependencies set by the dependency graph. Overlapping executions require that

$$\forall i, j \in [0, \cdots, n], i \neq j : t_{o_i} \notin [t_{o_j}; t_{o_j} + c_j^o]. \quad (16)$$

To enforce the dependencies, the starting points of the dependent operations must come after the operation has finished, or formally

$$\forall i, j \in [0, ..., n], o_i \rightarrow o_j : t_{o_j} \geq t_{o_i} + c_j^o i, \quad (17)$$

where $o_i \rightarrow o_j$ indicates that o_j depends on o_i.

B. Minimizing the Blocking Time

Given the previously defined constraints, the task is now to find a sequence with a minimal overall blocking time, or a maximal laxity. We chose the objective function as the aggregated, weighted blocking time

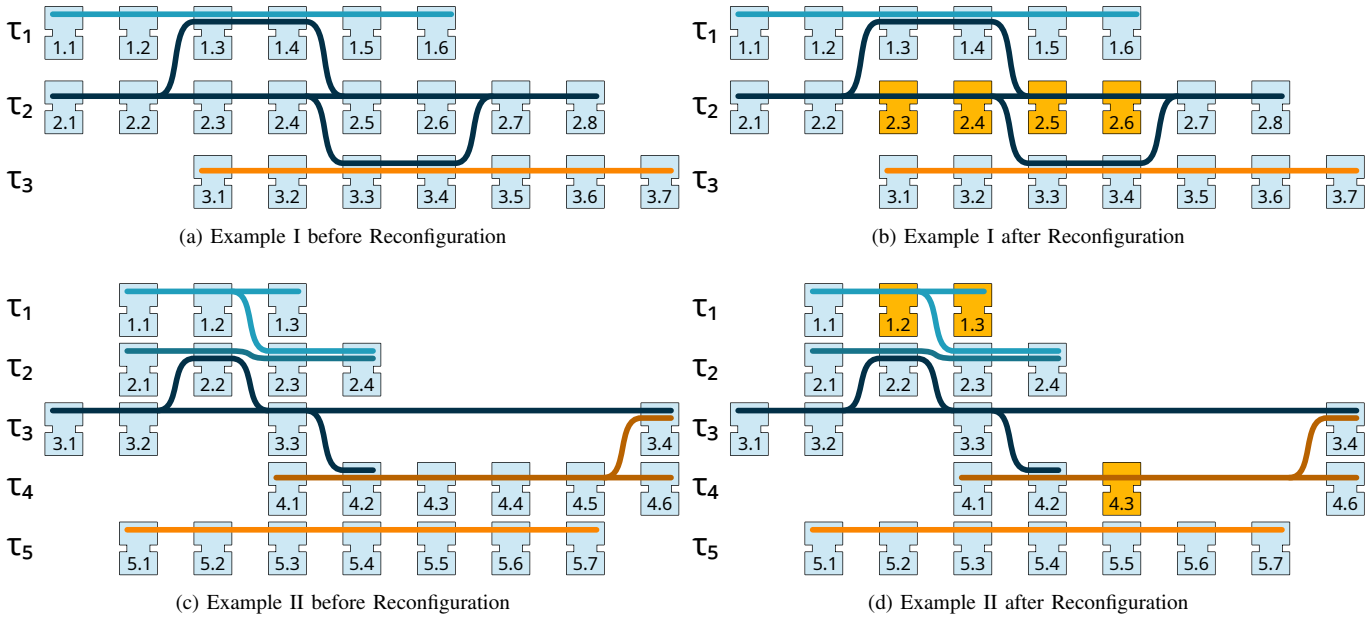
$$\min_S \sum_{i=1}^{n} \frac{B^R(\tau_i)}{B^{R,\max}(\tau_i)} \quad (18)$$

where the maximum blocking time of task τ_i is defined in Equation 14. We solve the optimization problem using single objective optimization using the tools of [14]. The constraints enforce that any solution will satisfy the schedulability condition in Equation 3. The objective function will reduce the impact of the reconfiguration on the real-time behavior. We demonstrate the results in a case study in the next section.

V. Evaluation

We demonstrate the schedulability condition and reconfiguration optimization on two example systems (Figure 3). Example I consists of 21 FBs and 3 tasks τ_1, τ_2, and τ_3. Example II consists of 24 FB and 5 tasks, τ_1 - τ_5. We assume a fixed WCET of every FB of $50\mu s$. The WCETs, blocking times $B^{\text{FB}}(\tau_i)$, and deadlines are summarized in Table I. The systems are inspired by real applications. To demonstrate the feasibility of the schedulability condition and optimization, naming and timing values are simplified without loss of generality.

We implement a reconfiguration for both examples. In Example I, four FBs are replaced by new FBs, which requires one state transformation each, and all tasks are affected. In Example II, there are two separate locations that are changed. In task τ_1, two FBs are replaced, and in task τ_4, three FBs are replaced by a single new FB. We generate dependency graphs that limit the feasible orderings of the necessary reconfiguration operations while guaranteeing consistency according to [3]. For instance, the reconfiguration is performed from event source to event sink, pushing out events following the old execution.



(a) Example I before Reconfiguration

(b) Example I after Reconfiguration

(c) Example II before Reconfiguration

(d) Example II after Reconfiguration

Fig. 3. The connection graph shows that the reconfiguration of Example I requires a change in components / FBs *2.3*, *2.4*, *2.5*, and *2.6*. The reconfiguration from Example II-1 to II-2 modifies components *1.2* and *1.3*, while combining *4.3*, *4.4* and *4.5* into a new *4.3*. Tasks are depicted as colored lines. In Example II, there is an additional task τ_5, which is not reconfigured.

TABLE I

SATISFACTION OF REAL-TIME CONSTRAINTS CAN BE GUARANTEED USING SCHEDULABILITY CONDITIONS, WHERE A LAXITY $L(\tau)$ MUST BE POSITIVE. A RECONFIGURATION MAY LEAD TO A NEGATIVE LAXITY. WHILE A HEURISTIC ORDERING OF RECONFIGURATION OPERATIONS CAN NOT SATISFY REAL-TIME CONSTRAINTS, OUR PROPOSED OPTIMIZED ALGORITHM CAN.

	Tasks			Undisrupted Schedulability		Heuristic Solution		Optimized Solution		Improvement
τ	$D = T$	C^τ	B^{FB}	$B^R(\tau)$	$L(\tau) = B^{R,\max}$	$B^R(\tau)$	$L(\tau)$	$B^R(\tau)$	$L(\tau)$	%
Example I										
τ_1	1000	300	100	0	600.00	1280	−680.00	190	410.00	85.16%
τ_2	4000	600	100	0	1413.71	1470	−56.29	1240	173.71	15.65%
τ_3	5500	350	0	0	1463.70	1470	−6.30	1320	143.70	10.20%
Example II										
τ_1	1000	250	100	0	650.00	1280	−630.00	480	170.00	62.50%
τ_2	2500	200	50	0	1196.07	1340	−143.93	660	536.07	50.75%
τ_3	4000	300	50	0	1449.05	1600	−150.95	1300	149.05	18.75%
τ_4	5000	350	0	0	1409.14	1660	−250.86	1370	39.14	17.47%
τ_5	7000	350	0	0	1529.44	1660	−130.56	1510	19.44	9.04%

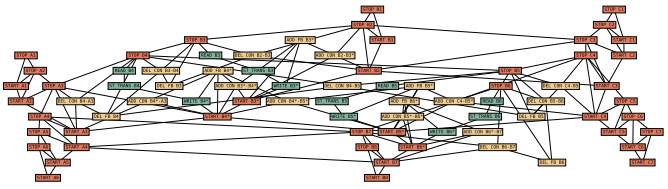


Fig. 4. The dependency graph of Example I limits the feasible orderings that satisfy the consistency requirements. Higher nodes restrict lower nodes. The colors indicate the priority of each operation according to the heuristic in [3].

Thus, while the reconfiguration flushes out old events, new events will follow the new system specification.

The undisrupted systems satisfy the schedulability condition (Eq. 3, see Table I). The system is not overloaded with work, which would make reconfiguration infeasible, and it is not idle either. The laxity in the undisturbed case is equal to the maximal blocking time of each task that may be caused by the reconfiguration.

A. Results

As can be seen in Table I, the heuristic ordering of reconfiguration operations failts to satisfy the schedulability condition in both examples, since the laxity is negative. In Example I, the blocking time of the heuristic solution is significantly larger than the maximum blocking time $B^{R,\max}$. This indicates that the reconfiguration may lead to one or multiple missed deadlines. The optimized sequence that was ordered as presented in the previous section is able to meet the deadline in both examples.

B. Discussion

The heuristic algorithm does not consider the deadlines of each task. Thus, it will not favor any task in particular, but try to start components as soon as possible and stop components as late as possible. This algorithm can find a decent solution very quickly, but fails to find the optimal solution. It will generate a sequence, even if a deadline would be missed. Our proposed optimization algorithm finds the global minimum and satisfies all constraints. In our examples, it will seek to disrupt time-critical tasks less. This can resolve the priority inversion caused by the heuristic algorithm. If there is no schedulable solution, it will terminate quickly. In our examples, the optimization terminates within minutes with the optimal solution, although a *feasible* solution can be received earlier.

VI. Related Work

There are several works on real-time execution of the IEC 61499. Apart from the discussions about suitable execution semantics (see [15], [16]), [8] proposes a real-time execution model based on cyclic execution and a first-in-first-out (FIFO) event dispatcher. The model does not consider blocking, and while no large extension to the IEC 61499 is necessary, it still requires the WCET and cycle time to be known. The model is further extended by [9] and [10], which introduce the concept of event-chains for the purpose of scheduling. The topic of blocking is shortly addressed, but the consequences of blocking reconfiguration sequences is not engaged. Further, current execution environments for the IEC 61499 have only rudimentary support for RM or Earliest Deadline First (EDF) scheduling, and the necessary information (deadlines, rates, priorities) are not part of the IEC 61499 models.

In this paper, we extend these models to permit schedulability analysis of dynamic reconfiguration scenarios. The choice of RM scheduling may reopen discussions about execution semantics, since it affects the order in which events are processed. For example, since events are scheduled but data transmissions are not, a delayed event may deal with other data than the application developer anticipated. This could be overcome with good system design.

VII. Conclusion

Dynamic Reconfiguration is an important feature to enable the adaptation of industrial automation systems during their life cycles. Two of the main concerns are the functional consistency and the disturbance of the real-time constraints. The disturbance dictates if a reconfiguration can be applied safely.

In this paper, we model the execution of a component-based architecture, such as the IEC 61499, as a scheduling problem that can be solved using preemptive RM scheduling and a PCP for the access to shared components / FBs. We show how the maximum blocking time during a reconfiguration can be calculated, and demonstrate that an optimal reconfiguration ordering can be found. Two examples indicate that an optimized sequence can make reconfiguration feasible where a heuristic solution does not suffice.

Currently, our schedulability condition is only applicable to single-threaded, preemptive RM scheduling. Extensions to distributed control systems are necessary and validation on real applications are needed, although current IEC 61499 runtime environments do not support preemptive RM scheduling.

Dynamic reconfiguration and general adaptability are indispensable for future industrial control systems. The IEC 61499 and other component-based architectures provide superior support for these features, yet the real-time behavior must be deterministic and precisely defined. There is a need for better behavioral modeling and execution environments with strict, deterministic real-time execution. By further automating the development process, current industrial systems could be elevated to greater resilience, while giving the same safety guarantees.

References

[1] H. Seifzadeh, H. Abolhassani, and M. S. Moshkenani, "A survey of dynamic software updating," *Journal of Software: Evolution and Process*, vol. 25, no. 5, pp. 535–568, May 2013.

[2] L. Nahabedian, V. Braberman, N. D'Ippolito, S. Honiden, J. Kramer, K. Tei, and S. Uchitel, "Dynamic update of discrete event controllers," *IEEE Transactions on Software Engineering*, pp. 1220–1240, 2018.

[3] L. Prenzel and S. Steinhorst, "Automated dependency resolution in dynamic reconfiguration for IEC 61499," in *Proceedings of the Conference on Emerging Technologies and Factory Automation*. IEEE, 2021.

[4] Y. Vandewoude, P. Ebraert, Y. Berbers, and T. D'Hondt, "Tranquility: A low disruptive alternative to quiescence for ensuring safe dynamic updates," *IEEE Transactions on Software Engineering*, vol. 33, no. 12, pp. 856–868, Dec. 2007.

[5] X. Ma, L. Baresi, C. Ghezzi, V. Panzica La Manna, and J. Lu, "Version-consistent dynamic reconfiguration of component-based distributed systems," in *ACM SIGSOFT symposium and the European conference on Foundations of software engineering - SIGSOFT/FSE '11*. ACM Press, 2011, p. 245.

[6] L. Baresi, C. Ghezzi, X. Ma, and V. P. L. Manna, "Efficient dynamic updates of distributed components through version consistency," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 340–358, 2017.

[7] G. Lyu and R. W. Brennan, "Towards IEC 61499-based distributed intelligent automation: A literature review," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2295–2306, 2021.

[8] A. Zoitl, G. Grabmair, F. Auinger, and C. Sunder, "Executing real-time constrained control applications modelled in IEC 61499 with respect to dynamic reconfiguration," in *International Conference on Industrial Informatics*. IEEE, 2005.

[9] A. Zoitl, R. Smodic, C. Sunder, and G. Grabmair, "Enhanced real-time execution of modular control software based on IEC 61499," in *IEEE International Conference on Robotics and Automation*. IEEE, May 2006, pp. 327–332.

[10] A. Zoitl, *Real-time Execution for IEC 61499*. Instrumentation, Systems, and Automation Society, 2009.

[11] J. Kramer and J. Magee, "The evolving philosophers problem: dynamic change management," *IEEE Transactions on Software Engineering*, vol. 16, no. 11, pp. 1293–1306, Nov. 1990.

[12] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, Sep. 1990.

[13] C. Sünder, V. Vyatkin, and A. Zoitl, "Formal verification of downtimeless system evolution in embedded automation controllers," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 1, pp. 1–17, Jan. 2013.

[14] L. Perron and V. Furnon, "OR-Tools." Google, 2021.

[15] T. Strasser, A. Zoitl, J. H. Christensen, and C. Sünder, "Design and execution issues in IEC 61499 distributed automation and control systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 41, no. 1, pp. 41–51, 2011.

[16] L. Prenzel, A. Zoitl, and J. Provost, "IEC 61499 runtime environments: A state of the art comparison," in *International Conference on Computer Aided Systems Theory*. Springer, 2019, pp. 453–460.