Mixed Criticality Scheduling in Time-Triggered Legacy Systems

Jens Theis and Gerhard Fohler
Technische Universität Kaiserslautern, Germany
Email: {jtheis,fohler}@eit.uni-kl.de

***Abstract*—Research on mixed criticality real-time scheduling has centered on an event-triggered (ET)/ priority-driven approach to scheduling. Regarding the time-triggered (TT) approach, which seems to have greater acceptability with certification authorities for safety critical domains, only first results have been presented, showing proof-of-concept of TT mixed criticality scheduling algorithms and comparing their resource utilization guarantees to those of ET ones. The algorithm is based on the offline construction of two coordinated schedule tables and an online mechanism. As a consequence, existing schedule tables for single criticality, possibly certified, have to be discarded.**

Here, we present an algorithm for the addition of mixed criticality scheduling to legacy TT systems. It leaves the existing schedule table unchanged, only provides analysis and adds a simple online mechanism to handle the changed criticality.

I. Introduction

In many domains such as avionics, industrial control, or health care, there is increasing demand for sharing computing platforms among applications with different importance and certification assurance levels. In the mixed criticality real-time scheduling model of [7], it is assumed that in order to certify a system as being correct, the certification authorities (CAs) mandate certain assumptions about the worst-case behavior of the system during runtime; these assumptions, e.g., for execution time, are typically far more conservative than the assumptions that the system designer would use during the process of designing, implementing, and testing the system if subsequent certification were not required. The difference in pessimism between designer's assumptions (likely at runtime) and CAs' mandate (most likely not at runtime) could be used to add non-critical activities and increase utilization. However, while the CAs are only concerned with the correctness of the safety-critical part of the system, the system designer is responsible for ensuring that the entire system is working correctly, including the non-critical parts. The current body of work in this area has focused on the event-triggered (ET)/priority-driven approach to scheduling.

Current practice in many domains, including (the safety-critical components of) automotive and avionics systems, which must meet multiple assurance requirements up to the highest criticality levels (e.g., DAL A in RTCA DO-178B or SIL4 in EN ISO/IEC 61508), however, favors a time-triggered approach (TT). In such TT systems, non-interference of safety-critical components by non-critical ones is ensured by strict isolation between components of different criticalities; Although such isolation facilitates the certification of the safety-critical functionalities, it can cause very low resource utilization.

A first result [1] shows proof-of-concept of mixed criticality real-time scheduling based on the TT approach. It is based on the offline construction of two coordinated schedule tables and an online mechanism to handle a change in criticality. In *legacy* TT systems, i.e. with existing, certified tables, this algorithm cannot be applied as it requires existing schedule tables to be changed, incurring substantial effort for recertification. At runtime, the low-criticality schedule table is executed until a high criticality job shows high criticality behavior and then the system switches to the high criticality schedule table. This solution shows a low runtime overhead but at cost of inflexibility.

The ET approach mixed critical EDF with mode switches was presented in [6]. In this approach, also two priority tables are created based on the deadlines of the jobs. When a high-criticality job exceeds its low criticality worst-case execution time (WCET), the system is switched to high criticality state with the high criticality priority table.

In this paper, we present a method to add the handling of criticality changes to existing schedule tables for legacy TT systems. It analyzes the existing table and properties of the high-criticality job set offline. A simple online mechanism then executes the jobs according to the existing table, manages a change of criticality, and then continues to execute the high-criticality job set. In case the existing schedule table is not suitable for the given mixed criticality job set, indications for its modification can be given. While in this case recertification may become necessary, the efforts will be lower than reconstruction of the schedule table from scratch.

Our method is based on slot-shifting [3] which was originally designed to add flexibility to TT systems with acceptable runtime overheads [5]. It takes the original task set and a constructed scheduling table[1] as input. As the table is constructed offline, complex constraints, such as distributed systems, end-to-end deadlines, precedence, etc. can be considered. It analyzes the table and the constraints to determine unused resources and leeways, which are represented as *spare capacities* offline. These can be used to provide flexibility and handle firm aperiodic tasks at runtime. Here, we build upon the offline analysis part and spare capacities to handle changes in criticality.

The remainder of this paper is structured as follows: Section II presents terms and notation used in this paper.

This work has been supported in part by the European project DREAMS under project No. 610640.

[1] It does not depend on a particular offline table construction algorithm.

Afterwards, Section III shows the slot-shifting offline phase for the mixed criticality job set. The results of the offline phase are incorporated into the online phase, i.e. the actual runtime scheduling, which is presented in Section IV. In Section V, we recapitulate the results and the applicability of our solution. Finally, Section VI concludes the paper.

II. Terminology and Notations

In this paper, we assume a dual-criticality system with the criticality levels LO and HI. Unless otherwise defined, we represent absolute times (e.g. release times) by lower case variables and relative times (e.g. WCET) by upper case variables.

The dual criticality jobs J_i with $i \in \{1, .., n\}$ are characterized by the 5-tuple $\langle \chi_i, r_i, d_i, C_i(\text{LO}), C_i(\text{HI}) \rangle$, with

- $\chi_i \in \{\text{LO}, \text{HI}\}$: criticality level of job i
- $r_i \in \mathbb{R}^+$: release time of job i
- $d_i \in \mathbb{R}^+$: absolute deadline of job i with $d_i > r_i$
- $C_i(\text{LO})$: LO criticality WCET
- $C_i(\text{HI})$: HI criticality WCET

which is in line with the denomination in the work of Baruah and Fohler [1]. For LO criticality jobs, we assume $C_i(\text{LO}) = C_i(\text{HI})$, whereas for HI criticality jobs $C_i(\text{LO}) \leq C_i(\text{HI})$.

Slot-shifting [3] uses *slots* as granularity for scheduling decisions. A slot $s = i$ is the time interval $[i; i+1)$. This interval contains the worst case time to schedule jobs Δt_S and the minimum guaranteed time to execute jobs Δt_E. The length of Δt_S and Δt_E is determined by the designer depending on the system under development. The WCET is given as multiples of Δt_E. The length of a slot is defined as $|s| = \Delta t_S + \Delta t_E$.

Another important concept of slot-shifting are *capacity intervals* which are used to manage the execution of jobs. Section III-A explains this concept in detail. The general notation of capacity intervals is: start and end of a capacity interval I_i is denominated by: $start(I_i)$ and $end(I_i)$, respectively. A capacity interval I_i represents the time window $[start(I_i); end(I_i))$. As a result, the length of a capacity interval I_i is calculated by $|I_i| = end(I_i) - start(I_i)$. I_c represents the current capacity interval, i.e. in which capacity interval the current point in time is located.

III. Offline Phase

In the offline phase of slot-shifting, we can resolve complex constraints – e.g. precedence constraints – which is not the scope of this paper. The interested reader is referred to [3]. Slot-shifting works on job-level, i.e. the instances of a periodic task are scheduled as single jobs. Scheduling on job-level allows for scheduling of time-triggered and event-triggered tasks and jobs. In this section, we show how to determine capacity intervals and how to calculate spare capacities which form the basis for the slot-shifting runtime scheduler.

A. Capacity Intervals

We divide the schedule into disjoint *capacity intervals* I_i with $i \in \{0, .., m\}$ based on the release times and deadlines of the jobs. It is important to highlight that capacity intervals are not identical to the execution windows, i.e. the time between release and deadline of a job. In the following, capacity intervals are briefly referred to as intervals. Each deadline of a job marks the end $end(I_i)$ of an interval I_i. Each job $J_k, k \in \{1, .., n\}$ is assigned to an interval with $end(I_i) = d_k$. Jobs with the same deadline belong to the same interval. The earliest start time $est(I_i)$ of an interval I_i is determined by the minimum of all release times of jobs assigned to this interval:

$$est(I_i) = \min_{\forall J_k \in I_i} (r_k) \tag{1}$$

The start of an interval is determined by the maximum of its earliest start time and the end of the previous interval:

$$start(I_i) = max\left(end(I_{i-1}), est(I_i)\right) \tag{2}$$

The gaps between the determined intervals above are defined as *empty intervals*, i.e. there is no job assigned to them. An interval I_i is called *independent* if there is no interval I_e with $e < i$ and $end(I_e) > est(I_i)$ and there is no interval I_l with $i < l$ and $end(I_i) > est(I_l)$. The length $|I_i|$ of an interval is calculated by equation 3.

$$|I_i| = end(I_i) - start(I_i) \tag{3}$$

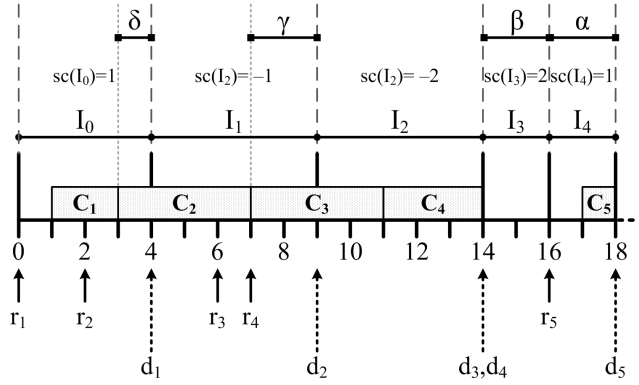
Based on the observation that demand-based schedulability tests need only to check intervals until deadlines of jobs [2], capacity intervals, which partition considered demand based on job deadlines, simplify the maintenance of demand requirements and scheduling of the demand, respectively, at runtime.

B. Spare Capacities

In the following, we explain the general concept of *spare capacities* based on non-mixed-criticality jobs (as shown in the original slot-shifting [3]). After that, we present how this is applied to mixed criticality job sets. The spare capacity $sc(I_i)$ of an interval I_i represents the amount of available resources within this interval after guaranteeing TT jobs. The difference between the length of the interval and the amount of demand of jobs assigned to this interval determines the amount of available resources. Further, it is possible that the demand within an interval is greater than the length of the interval such that the spare capacity will be negative. In the following, we show how spare capacities are calculated in detail and in combination with section IV-A, we show the consequences of negative spare capacities. We calculate spare capacities beginning with the last interval I_m until the first interval I_0.

Figure 1 shows an example to illustrate the calculation of spare capacities. As shown in section III-A, we determine the

intervals $I_0 - I_4$. In this example, we schedule the jobs as late as possible to exemplify the calculation of spare capacities. Jobs cannot be scheduled before their release time; otherwise, the schedule is not feasible and the job set is not schedulable. In the figure, time span α represents an independent interval. The spare capacity of an independent interval is calculated by the difference between the length of the interval and the sum of WCETs of jobs assigned to this interval. Time span β shows an empty interval, i.e. there are no jobs assigned to it, and hence, the spare capacity of this interval is the length of the interval. The consequences of negative spare capacities are illustrated by time span γ. The amount of demand assigned to I_2 is more than the length of the interval. As a result, interval I_1 has to lend capacity to the succeeding interval I_2. The consequence of the negative spare capacity is called *borrowing*. Time span δ shows the consequences of borrowing propagation, i.e. interval I_1 lent capacity to I_2 such that I_1 itself has to borrow capacity from the earlier interval I_0. Eventually, interval I_0 is long enough to lend capacity to I_1 and schedule the jobs assigned to I_1. Thus, the spare capacity of I_0 is non-negative.



J_i	r_i	d_i	C_i
J_1	0	4	1
J_2	2	9	4
J_3	6	14	4
J_4	7	14	3
J_5	16	18	1

Figure 1. Spare capacity calculation example: for simplicity of the example without different criticality levels

We can test the schedulability based on the spare capacities, under the condition of obeying release times and deadlines. Spare capacities of independent intervals have to be non-negative. Further, after borrowing and borrowing propagation there must be an earlier interval with non-negative spare capacity.
Equation (4) shows the calculation of spare capacities for non-mixed-criticality jobs, which is in the following applied to mixed criticality job sets. The calculation includes the requirements described by Figure 1.

$$sc\left(I_i\right) = |I_i| - \sum_{J_k \in I_i} C_k + min\left(sc\left(I_{i+1}\right), 0\right) \qquad (4)$$

For the mixed criticality case, we use the calculations of the spare capacities presented above. We calculate two spare capacity values for each interval: $sc^{\text{LO}}(I_i)$ and $sc^{\text{HI}}(I_i)$. The calculation of spare capacities is done by equations (5) and (6). Spare capacities $sc^{\text{LO}}(I_i)$ represent available capacities based on designer's assumptions, i.e. considering $C(\text{LO})$, with all jobs. Additionally, $sc^{\text{HI}}(I_i)$ represents available resources based on CAs' assumptions, i.e. considering $C(\text{HI})$, with only HI criticality jobs.

$$sc^{\text{LO}}\left(I_i\right) = |I_i| - \sum_{J_k \in I_i} C_k(\text{LO}) + min\left(sc^{\text{LO}}\left(I_{i+1}\right), 0\right) \quad (5)$$

$$sc^{\text{HI}}\left(I_i\right) = |I_i| - \sum_{\substack{J_k \in I_i \\ \wedge \chi_k = \text{HI}}} C_k(\text{HI}) + min\left(sc^{\text{HI}}\left(I_{i+1}\right), 0\right) \quad (6)$$

As Figure 1 (intervals I_0 and I_4) shows, slot-shifting allows for non-work-conserving scheduling based on spare capacities. Further, slot-shifting allows for scheduling of strictly periodic jobs, i.e. jobs that have to be executed directly at their periodic release.

IV. Online Phase

At runtime, we execute mixed criticality jobs based on the spare capacity in the current interval I_c and the deadlines of the jobs. In this section, we present how to select the next job for execution. Further, we show the update procedures for spare capacities depending on the job execution.

A. Decision Mode and Selection Function

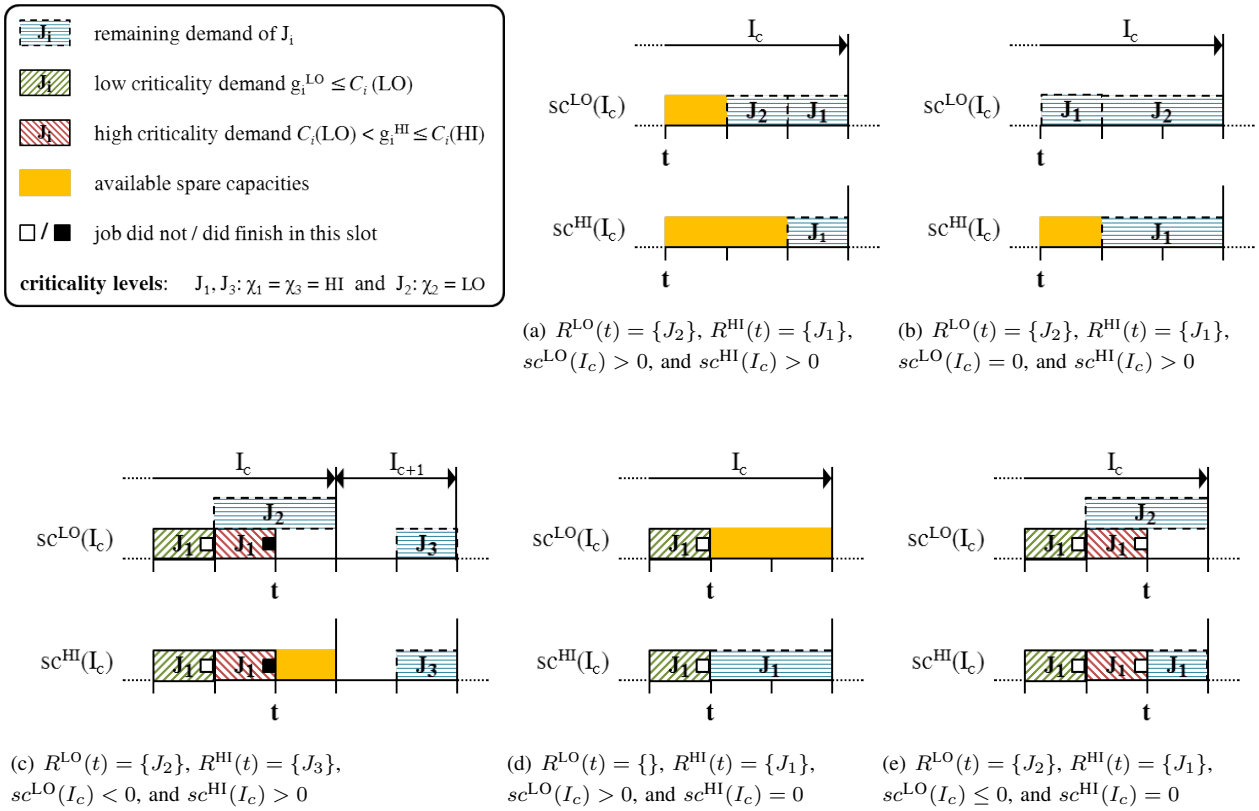
The decision mode of the slot-shifting runtime scheduler is preemptive at slot borders. We use three ready queues: $R^{\text{LO}}(t) = \{J_i | r_i \leq t \wedge \chi_i = \text{LO}\}$ contains TT LO criticality jobs. Further, $R^{\text{HI}}(t) = \{J_i | r_i \leq t \wedge \chi_i = \text{HI}\}$ is used for TT HI criticality jobs. An implementation with only one queue is also possible, but for simplicity of explanation, we will present the algorithm based on two queues. We define $R(t) := R^{\text{LO}}(t) \cup R^{\text{HI}}(t)$ which represents all guaranteed jobs. The scheduler selects the next executing jobs based on the ready queues, the intervals and the spare capacities. In contrast to standard slot-shifting, LO criticality spare capacities can be negative in the current interval I_c. This can occur when a HI criticality job executes for more than $C(\text{LO})$ and we have to continue its execution. As long as the LO criticality spare capacity in the current interval is negative, i.e. $sc^{\text{LO}}(I_c) < 0$, we can only execute HI criticality jobs.
Based on these observations, we can distinguish the following list of all possible decision cases at time t.

a) $R(t) = \{\}$: slot is idle because there are no ready jobs.

b) $R(t) \neq \{\} \wedge sc^{\text{HI}}(I_c) > 0$:

1) $sc^{\text{LO}}(I_c) > 0$:
We select the ready job with the earliest deadline in $R(t)$. Figure 2(a) illustrates this situation with an example.

2) $sc^{\text{LO}}(I_c) = 0$:
We select the job with the earliest deadline in $R(t)$ because there are available resources for HI criticality jobs and thus, no need to prioritize them. On the contrary, there is no leeway for LO criticality demand



(a) $R^{\text{LO}}(t) = \{J_2\}$, $R^{\text{HI}}(t) = \{J_1\}$, $sc^{\text{LO}}(I_c) > 0$, and $sc^{\text{HI}}(I_c) > 0$

(b) $R^{\text{LO}}(t) = \{J_2\}$, $R^{\text{HI}}(t) = \{J_1\}$, $sc^{\text{LO}}(I_c) = 0$, and $sc^{\text{HI}}(I_c) > 0$

(c) $R^{\text{LO}}(t) = \{J_2\}$, $R^{\text{HI}}(t) = \{J_3\}$, $sc^{\text{LO}}(I_c) < 0$, and $sc^{\text{HI}}(I_c) > 0$

(d) $R^{\text{LO}}(t) = \{\}$, $R^{\text{HI}}(t) = \{J_1\}$, $sc^{\text{LO}}(I_c) > 0$, and $sc^{\text{HI}}(I_c) = 0$

(e) $R^{\text{LO}}(t) = \{J_2\}$, $R^{\text{HI}}(t) = \{J_1\}$, $sc^{\text{LO}}(I_c) \leq 0$, and $sc^{\text{HI}}(I_c) = 0$

Figure 2. Examples situation when selecting the next job for decision cases with $R(t) \neq \{\}$ for an exemplary point in time t — J_1, J_3: HI **criticality jobs,** J_2: LO **criticality job**

and thus, we have to select a guaranteed job. Figure 2(b) illustrates this situation with an example.

3) $sc^{\text{LO}}(I_c) < 0$:
We select the job with the earliest deadline in $R^{\text{HI}}(t)$. There is not enough available LO criticality spare capacity in the current interval to complete remaining LO criticality jobs and hence, they are skipped. Figure 2(c) illustrates this situation with an example.

c) $R(t) \neq \{\} \wedge sc^{\text{HI}}(I_c) = 0$:

1) $sc^{\text{LO}}(I_c) > 0$:
The job with the earliest deadline in $R^{\text{HI}}(t)$ is selected because we have to execute a HI criticality job to guarantee completion with CAs' assumptions. Figure 2(d) illustrates this situation with an example.

2) $sc^{\text{LO}}(I_c) \leq 0$:
We select job with the earliest deadline in $R^{\text{HI}}(t)$ because of the reasons mentioned in case c1. Figure 2(e) illustrates this situation with an example.

d) $R(t) \neq \{\} \wedge sc^{\text{HI}}(I_c) < 0$: The HI criticality spare capacity cannot be less than zero. This could only happen if we execute a HI criticality job J_i for more than $C_i(\text{HI})$ which we assume is prevented by the system.

B. Spare Capacity Maintenance

As a consequence of the process to select the next executing job, we have to update the spare capacities depending on the criticality level and type of the job.

No execution: If an idle slot has been scheduled, we decrease both $sc^{\text{LO}}(I_c)$ and $sc^{\text{HI}}(I_c)$ by one slot.

Guaranteed job execution: If a guaranteed job J_i, either TT or firm ET, has been scheduled, then we have to differentiate whether J_i is assigned to the current interval I_c or to a later interval I_k. Further, the fact whether a HI criticality job exceeded $C(\text{LO})$, i.e. showed HI behavior, influences the maintenance of spare capacities.

A) $J_i \in I_c$

1) J_i did not exceed $C_i(\text{LO})$:
In both spare capacity calculations, $sc^{\text{LO}}(I_c)$ and $sc^{\text{HI}}(I_c)$, the scheduled demand has already been considered and hence, both spare capacities in the current interval remain unchanged.

2) J_i exceeded $C_i(\text{LO})$:
As the job executes for more than the considered amount of LO criticality execution time, we have to decrease the LO criticality spare capacity in the current interval by one slot. On the contrary, for the HI criticality spare capacity $sc^{\text{HI}}(I_c)$, this scheduled execution has already been considered and thus, $sc^{\text{HI}}(I_c)$ is unchanged.

B) $J_i \in I_k$ with $I_k \neq I_c$

1. J_i did not exceed $C_i(\text{LO})$:
The scheduled demand has not been considered in $sc^{\text{LO}}(I_c)$. As a consequence, we decrease $sc^{\text{LO}}(I_c)$ by one slot. Additionally, we increase $sc^{\text{LO}}(I_k)$ where the demand has been originally considered in the spare

capacity calculations. In other words, one slot of execution is swapped between the current interval I_c and the assigned job interval I_k.
Furthermore, there is the aspect of borrowing which has to be considered: If $sc^{\text{LO}}(I_k)$ was less than zero before increasing by one in the current step, then I_k was borrowing capacity from at least one earlier interval. Thus, we have to increase the spare capacities by one if there was borrowing or borrowing propagation in one or several of the intervals from I_c to I_{k-1}.
For the HI criticality spare capacities, we have to apply the same procedure as for the LO criticality ones.

2. J_i exceeded $C_i(\text{LO})$:
The demand has not been considered in the LO criticality spare capacities, neither in $sc^{\text{LO}}(I_c)$ nor in $sc^{\text{LO}}(I_k)$. Thus, only the spare capacity in the current interval $sc^{\text{LO}}(I_c)$ is decreased by one.
For the HI criticality spare capacities, we have to apply the same procedures as in step B1; which are:
The scheduled demand has not been considered in $sc^{\text{HI}}(I_c)$ and thus, we have to decrease $sc^{\text{HI}}(I_c)$ by one slot. Further, $sc^{\text{HI}}(I_k)$, where the demand has been originally considered, is increased by one.
Still, there may be the aspect of borrowing which has to be considered: If $sc^{\text{HI}}(I_k)$ was less than zero before increasing by one in the current step, then I_k was borrowing capacity from at least one earlier interval. Thus, we have to increase the spare capacities by one if there was borrowing or borrowing propagation in one or several of the intervals from I_c to I_{k-1}.

After execution of a job in the current slot and before the scheduling process continues in the next slot, we have to compare the actual execution time of the job with the WCET for the LO and the HI criticality case if the job finished in the current slot. If TT jobs complete earlier than their specified WCET ($C(\text{LO})$ for LO and $C(\text{HI})$ for HI criticality spare capacities), the difference between actual execution time and specified WCET can be added to the spare capacities of the corresponding intervals, as shown in [4].

As a conclusion, we can make efficient use of the available resources by the update mechanism presented above. Further, the LO criticality and HI criticality spare capacities allow for flexibility to react to the actual job behavior.

V. Discussion

In contrast to [1], there is no need to construct two schedule tables for the different requirements of the designer and the CAs. We only need to calculate two sets of spare capacities for the existing intervals whereas the intervals are identical for designer's and CAs' assumptions. At runtime, the certified schedule table is unchanged. Based on the spare capacities the runtime mechanism handles the requirements of LO and HI criticality jobs within the constraints of the offline computed, certified schedule table. This simplifies offline preparation and certification.

The runtime overhead of slot-shifting has been compared to Linux' Completely Fair Scheduler and Litmus RT, with the conclusion that the overhead of slot-shifting is in the same order as the reference schedulers [5]. The necessary changes to extend the implemented slot-shifting version to our mixed criticality slot-shifting affect only the selection of the next job to execute and the update of the spare capacities. In both job selection and spare capacity update, only a second spare capacity has to be checked and updated, respectively.

The presented algorithm can be easily applied to resume the LO criticality state after a switch to HI. The available spare capacities are used to execute HI criticality jobs that exceed the WCET based on the designer assumptions such that we do not need to drop LO criticality jobs as long as there are enough spare capacities available to not harm the execution of HI criticality jobs. If there are no spare capacities left, we continue the execution with the restricted job set of only HI criticality jobs to guarantee them. The maintenance of spare capacities within the capacity intervals allows for switching back to the LO criticality system state as soon as we can guarantee the execution of all jobs based on designer assumptions. This is achieved by updating the LO criticality spare capacities although only HI criticality jobs are executed. The LO criticality spare capacities indicate the point in time when the WCET of LO criticality jobs can be guaranteed for them, again. The maximum number of spare capacity updates per slot occurs when there is borrowing propagation and hence, is bounded by the number of jobs. This is based on the fact that job deadlines refer to end of intervals. More intervals than jobs are only possible if there are empty intervals but empty intervals are not affected by borrowing and borrowing propagation. As a consequence, the complexity of the selection function and the spare capacity maintenance can be bounded by a linear function.

As a result, we can make use of TT legacy systems and add flexibility without harming certified schedule tables at runtime. Additionally, the implementation of slot-shifting shows an applicable runtime overhead such that slot-shifting represents a valid choice for safety-critical TT legacy systems with mixed criticality job sets.

VI. Conclusion

In this paper, we have presented a method for including mixed criticality real-time scheduling following the model of [7] to legacy TT systems. Earlier results for TT systems [1] require the offline construction of two coordinated schedule tables from scratch for mixed criticality task sets. In case a schedule table already exists and has been certified, this approach, necessitates complete recertification.

In contrast, the method presented in this paper takes an existing schedule table and the properties of the HI criticality job set as input. It performs analyses and provides a simple online mechanisms to include additional criticality job sets. The jobs of the original schedule table are executed as before. When a change of criticality arises, the online mechanism manages it, and then continues to execute the HI criticality jobs. In case the existing schedule table is not suitable for the given mixed criticality job set, indications for its modification can be given. While in this case recertification may become necessary, the efforts will be lower than reconstruction of the schedule table from scratch.

Future work will include analysis of the existing work regarding limitations of the presented method and comparison

to pure TT and ET approaches. Additionally, the inclusion of aperiodic jobs will be in the focus of our work. Further, the extension to arbitrary criticality levels will be included in future research.

REFERENCES

[1] Sanjoy Baruah and Gerhard Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Real-Time Systems Symposium*, 2011.

[2] Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems Journal*, 1990.

[3] Gerhard Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. *Real-Time Systems Symposium*, 1995.

[4] Damir Isovic and Gerhard Fohler. Handling mixed sets of tasks in combined offline and online scheduled real-time systems. *Real-Time Systems Journal*, 2009.

[5] Stefan Schorr and Gerhard Fohler. Integrated time- and event-triggered scheduling an overhead analysis on the arm architecture. In *International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013.

[6] Dario Socci, Peter Poplavko, Saddek Bensalem, and Marius Bozga. Mixed critical earliest deadline first. In *Euromicro Conference on Real-Time Systems*, 2013.

[7] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. *Real-Time Systems Symposium*, 2007.