

# **Real-Time Systems: the past, the present, and the future**

Proceedings of a conference organized in celebration of Professor Alan Burns' sixtieth birthday

ISBN-13: 978-1482707786

ISBN-10: 1482707780

## *Preface*

A conference was held at The University of York on March 14<sup>th</sup>, 2013, to mark the occasion of Alan Burns' sixtieth birthday. This conference was titled *Real-Time Systems: the past, the present, and the future*; its purpose was to recognize and celebrate Alan's many contributions to the discipline of real-time computing. This volume contains technical papers contributed by some of Alan's colleagues that, in their opinions, reflect Alan's impact and influence on real-time computing. As can be seen from the wide range of topics addressed in these papers, Alan's work has had an immense impact upon a wide variety of aspects of real-time computing.

## Table of Contents

|   |     |
|---|-----|
| Utilization-based schedulability assessment for dynamic distributed embedded systems . . . . .  | 1   |
| <i>Luis Almeida</i>   |     |
| Timing of CAN under the Influence of Random Error Events . . . . .  | 14  |
| <i>Philip Axer and Rolf Ernst</i>   |     |
| Independence - a misunderstood property of and for probabilistic real-time systems . . . . .  | 29  |
| <i>Liliana Cucu-Grosjean</i>  |     |
| Burns Standard Notation for real time scheduling . . . . .  | 38  |
| <i>Robert I. Davis</i>  |     |
| MAST: Bringing Response-Time Analysis into Real-Time Systems Engineering . . . . .  | 42  |
| <i>Michael González Harbour, J. Javier Gutiérrez, Julio L. Medina, J.Carlos Palencia, J. María Drake, Juan Rivas, Patricia López Martinez, and César Cuevas</i> |     |
| Resource Reservation for Mixed Criticality Systems . . . . .  | 60  |
| <i>Giuseppe Lipari and Giorgio C. Buttazzo</i>  |     |
| On the Average Complexity of the Processor Demand Analysis for Earliest Deadline Scheduling . . . . .   | 75  |
| <i>Giuseppe Lipari, Laurent George, Enrico Bini, and Marko Bertogn</i>  |     |
| Perspectives on the Analysis and Design of Real-Time Distributed Systems with UML-based standard modelling languages . . . . .                                  | 87  |
| <i>Julio L. Medina</i>  |     |
| Operating system support for Ada timers . . . . .   | 105 |
| <i>Mario Aldea Rivas and Michael González Harbour</i>   |     |
| On the Processor Utilization Bound of the $C = D$ Scheduling Algorithm . . . . .  | 119 |
| <i>J. Augusto Santos-Jr, George Lima, and Konstantinos Bletsas</i>  |     |
| Models for Real-Time Workload: A Survey . . . . .   | 133 |
| <i>Martin Stigge and Wang Yi</i>  |     |
| The Utilization Bound of CAN: Theory and Practice . . . . .   | 160 |
| <i>Eduardo Tovar, Nuno Pereira, and Ricardo Gomes</i>   |     |
| When Theory Meets Technology . . . . .  | 178 |
| <i>Tullio Vardanega</i>   |     |

|  |     |
|--|-----|
| A Linear Model for Setting Priority Points in Soft Real-Time Systems . . . . . | 192 |
| <i>Bryan C. Ward, Jeremy P. Erickson and James H. Anderson</i>                 |     |

# Utilization-based schedulability assessment for dynamic distributed embedded systems

Luis Almeida

IT / DEEC - Faculdade de Engenharia  
University of Porto, Porto, Portugal  
lda@fe.up.pt

**Abstract.** Distributed embedded systems are becoming progressively more dynamic, in an attempt to improve resource efficiency, increase functionality and reduce time-to-market. In these systems, components can be connected, change mode or disconnected at runtime. Some of the functionality, though, requires guaranteed timings through all such changes, which calls upon fast on-line schedulability tests. In this paper we consider four existing utilization-based tests with release jitter, a particularly relevant feature in distributed systems. Then, we carry out an extensive comparison using random task sets to characterize their relative merits, particularly under different release jitter patterns.

**Keywords:** real-time scheduling, admission control, distributed embedded systems

## 1 Introduction

Utilization-based schedulability tests are known since the 70s [10] and they are extremely simple to compute given their linear time complexity. When used on-line in the scope of admission controllers of open systems, their time complexity can even be made constant by keeping track of the total utilization of the currently running tasks. However, this simplicity comes at a cost, which is a lower accuracy than other more complex schedulability tests, such as those based on response time or processor demand analysis. This disadvantage grows as the task model considers more aspects that are typical in real applications such as blocking, deadlines different from periods, offsets, non-preemption and arbitrary priority assignment policies [5].

Furthermore, recent developments in response time analysis brought to light new linear methods that are faster to compute, despite a small loss in accuracy [8] [4]. These tests are still more complex to evaluate than utilization-based tests but the difference became relatively small and their accuracy is still better.

Nevertheless, utilization-based tests can still be an effective way to improve efficiency in the use of system resources when the task set in the system can change on-line, either due to admission/removal of tasks or because the tasks exhibit multiple modes of operation.

Moreover, such tests are also adequate when it is important to manage utilization, for example in the scope of dynamic QoS (bandwidth) management, with tasks that allow variable worst-case execution times by switching between different algorithms or that can be executed at different rates. In fact, utilization-based tests handle bandwidth directly thus, for example, when a task leaves the system, the bandwidth that is freed can be assigned to the remaining ones, according to an adequate policy, from fixed priority/importance, to elastic, weighted, even, etc [6] [9]. This assignment cannot be done in a comparable manner using other kinds of tests that require more costly trial and error approaches.

However, distributed real-time systems entail one additional difficulty, which is the frequent occurrence of release jitter, both in the tasks and messages they use to communicate. In fact, messages are frequently released by tasks that do not execute periodically because of variable interference of higher priority ones in the processor where they reside and, similarly, tasks are frequently triggered by the reception of messages that do not arrive periodically for similar reasons [12].

The incorporation of release jitter in utilization-based tests was not done until very recently. In this paper we revisit the existing work on utilization-based schedulability tests that account for release jitter and we compare the performance of such tests with random task sets, extending the comparison in [2] with an analysis of the release jitter impact.

## 2 Related work

Despite all the work on utilization-based schedulability tests for periodic task models carried out in the past four decades, to the best of the author's knowledge, there has never been an extension to include release jitter until very recently. In fact, the first published result in that direction seems to have been by Davis and Burns in 2008 [7] where they showed a pseudo-utilization-based schedulability test that accounts for release jitter based on the results in [1]. In fact, it is shown that the release jitter that affects each task can be subtracted to its period, or deadline if shorter than the period, and used directly in the denominator of the respective utilization term. The total sum gives a pseudo-utilization value that can be compared against the Liu and Layland bounds. The same work has an extensive comparison among different tests, focusing on the relative performance of response-time based tests.

On the other hand, the author was previously engaged in the development of a different analysis applicable to both Rate-Monotonic and Earliest Deadline First scheduling that accounts for release jitter as an extra task [11]. This analysis results in a set of  $n$  conditions for  $n$  tasks. However, the authors also showed how such conditions could be reduced to just one single test that upper bounds the  $n$  conditions. This single condition test was further improved in [2] with a tighter version.

The comparison between the  $n$  conditions test and the test in [7] is not obvious, as it will be shown further on, since none dominates the other in all sit-

uations. Moreover, the comparison becomes even less obvious given the different underlying priority assignment policies. In fact, while the former assumes usual *rate-monotonic* priorities, the latter assumes a '*deadline minus jitter*'-*monotonic* policy. Naturally, either test can be optimistic whenever used with a priority assignment different from the assumed one.

Nevertheless, both the  $n$  conditions of the former and the pseudo-utilization of the latter require some extra adaptation for use within the scope of dynamic bandwidth management schemes. Conversely, the single condition tests that handle bandwidth can be used directly. Unfortunately their performance is always worse as shown in [2].

In this paper we extend the comparison in [2] with a thorough analysis of the impact of the release jitter in the tests performance. We focus on the  $n$  conditions test in [11], the '*deadline minus jitter*' test in [7] and the single condition bandwidth test introduced in [2]. In particular, we discuss the situations that make each of the first two tests dominate each other.

### 3 Task model and previous analysis

We consider a set  $\Gamma$  of  $n$  periodic/sporadic tasks  $\tau_i (i = 1..n)$ , with period  $T_i$  and worst-case execution time  $C_i$ , which may suffer a jittered release of at most  $J_i$ , i.e., the task can be released at any point in time between its *virtual* periodic/sporadic activation and  $J_i$  time units later. The deadlines are currently considered to be equal to the respective periods. We consider the tasks in the set to be sorted by growing period, except when explicitly referred otherwise. In the context of Rate-Monotonic scheduling (RM)  $\tau_1$  will be the highest priority task and in the context of Earliest Deadline First scheduling (EDF)  $\tau_1$  will be the task with the highest preemption level. For the moment we also consider tasks to be fully preemptive and independent.

The test in [7] copes with deadlines equal to or less than the periods and blocking but, for the sake of comparison with the other tests, we will constrain it to our model. In this case we will assume the task set sorted by growing '*period minus jitter*'. The test then states that schedulability of the task set is guaranteed if condition (1) is satisfied. The test in [7] considers fixed priorities, only, but the same reasoning expressed therein allows to infer that it should also be applicable to EDF scheduling. Thus, we will also consider it in such case.  $U_{RM,EDF}^{lub}(n)$  refers to the least upper bound on utilization for RM scheduling with  $n$  tasks and for EDF, i.e.,  $n(2^{1/n} - 1)$  and 1, respectively.

$$\sum_{i=1}^n \frac{C_i}{T_i - J_i} \leq U_{RM,EDF}^{lub}(n) \quad (1)$$

On the other hand, the test in [11] states that if the  $n$  conditions in (2) are satisfied, then the task set is schedulable.

$$\forall_{i=1..n} \sum_{j=1}^i \frac{C_j}{T_j} + \frac{\max_{j=1..i} J_j}{T_i} \leq U_{RM,EDF}^{lub}(i) \quad (2)$$

The work in [11] also provides a simplified test based on a single condition as in (3).

$$\sum_{i=1}^n \frac{C_i}{T_i} + \frac{\max_{i=1..n} J_i}{T_1} \leq U_{RM,EDF}^{lub}(n) \quad (3)$$

This condition may become very pessimistic in cases with a wide dispersion of tasks periods. In fact, a task with a longer period can accommodate a longer release jitter, leading to a term  $\max_{i=1..n} J_i$  that can be close to, or longer than,  $T_1$ . This situation can be improved using a new simplified test with a single condition (4) that is more accurate than (3) despite more costly to compute.

$$\sum_{i=1}^n \frac{C_i}{T_i} + \max_{i=1..n} \frac{\max_{j=1..i} J_j}{T_i} \leq U_{RM,EDF}^{lub}(n) \quad (4)$$

It can be trivially verified that if condition (4) holds than all  $n$  conditions in (2) will also hold and thus this test also implies the schedulability of the task set. Note, also, that the term  $\max_{j=1..i} J_j$  is now divided by  $T_i$  that grows with  $i$  and thus the potentially longer jitter of slower tasks does not have such a strong effect as in condition (3).

## 4 Comparing the tests

In this section we compare the four conditions presented above in terms of schedulability accuracy, i.e., level of pessimism, and execution time. For both cases, we include one accurate analysis as a reference, namely a response time test for RM as in (5) [3] and a CPU demand test for EDF based on the analysis presented in [12] as in (6). For the sake of presentation, we will also refer to the test of condition (1) as RM whenever applied to fixed priorities, despite these being assigned in 'deadline minus jitter' order.

$$\forall_{i=1..n} Rwc_i \leq T_i \text{ **with** } Rwc_i = R_i + J_i \text{ **and** } R_i = \sum_{j=1}^{i-1} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil * C_j + C_i \quad (5)$$



$$\begin{aligned}
\forall t \in \Psi h(t) \leq t \text{ with } h(t) &= \sum_{i=1..n} \left\lfloor 1 + \frac{t - (T_i - J_i)}{T_i} \right\rfloor * C_i & (6) \\
\text{and } \Psi &= \{t \in [0, L] \wedge t = m * T_i + (T_i - J_i), \forall i = 1..n, m = 0, 1, 2, \dots\} \\
\text{with } L &= \sum_{i=1..n} \left\lfloor \frac{L + J_i}{T_i} \right\rfloor * C_i
\end{aligned}$$

For the comparison we use synthetic task sets with random tasks. The generation method is the following. We start by generating a target global utilization ( $U$ ), then we generate random tasks one by one with a period ( $T_i$ ) uniformly distributed within  $[1, 10]$  and utilization factor ( $U_i$ ) within  $(0, 0.2]$ . The utilization of the last task is adjusted if the total utilization is more than 1% above the desired target  $U$ . Then, we compute the respective execution times ( $C_i = T_i * U_i$ ) and we generate the values of release jitter ( $J_i$ ) randomly with uniform distribution.

The values of  $U$  go from 0.2 to 0.98 in steps of 0.02 and for each particular point we generate 5000 random task sets. We generated two sets of experiments with different patterns of jitter. In one case we considered the same interval in the generation of jitter for all tasks, namely randomly distributed within  $(0, 0.3]$ . This is a relatively small jitter that impacts mainly the tasks with shorter periods. We call this the *flat* jitter profile. On the other hand, we generated another profile with a stochastically growing value of jitter according to the task periods. Basically, the jitter  $J_i$  is randomly generated within  $(0, 0.5 * T_i]$ . This means that longer tasks can suffer longer release jitter. We call this, the *linear* jitter profile.

The simulation experiments were carried out using Matlab on a common laptop with a Centrino CPU operating  $1.2GHz$  and running the Windows<sub>XP</sub> operating system. The analysis execution times shown next were not optimized and serve mainly for a quick relative assessment.

Figure 1a shows the results obtained using the RM scheduling and the flat jitter profile. The rightmost curve shows the ratio of task sets that meet the response time test in (5) with both 'deadline minus jitter' priorities and rate-monotonic priorities. In this case, given the low amount of jitter, its impact in the priority assignment is minimal and, consequently, the response time test delivers very similar results with both policies (Ref1/2). This reference test assures that all task sets with this jitter profile and utilization roughly up to 74% are schedulable. Then we track how many of the task sets found schedulable by the reference test were also deemed schedulable by tests (1) to (4). This gives us a good measure of the level of pessimism embedded in these tests.

The values obtained for RM scheduling with the flat jitter profile are shown in Table 1. Test (2) performed best in this case, finding 75% of the sets considered schedulable by the reference test. Test (1) performed very closely, 73%. Then come tests (4) and (3) finding 62% and 55% of the reference schedulable sets, respectively. It is expected that these two tests perform poorer than test (2) since they are a simplification of it and it is also expected that both tests perform similarly since with the flat jitter profile there is a non-negligible probability that

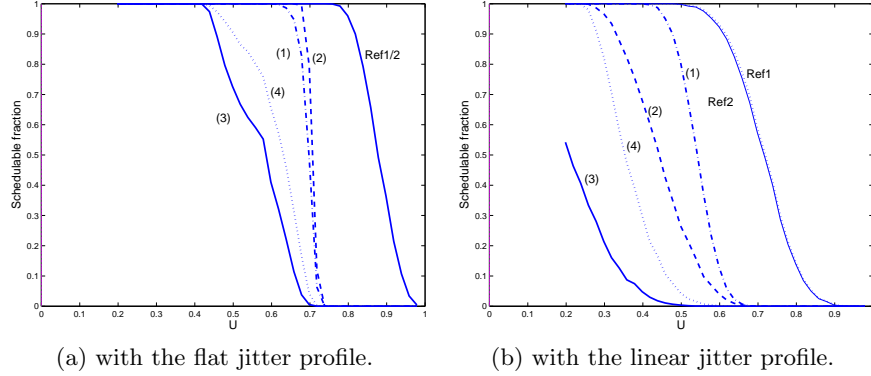


Fig. 1: Performance of different tests under RM

the maximum jitter will affect the task with the shortest period, which leads to a similar result with both tests. Table 1 also shows the average and maximum time taken by each analysis with each set.

|                            | Ref 1/2 | Test (1) | Test (2) | Test (3) | Test (4) |
|----------------------------|---------|----------|----------|----------|----------|
| % of schedulable task sets |         |          |          |          |          |
| found by the U-based tests | 100%    | 73%      | 75%      | 55%      | 62%      |
| analysis exec time (avg)   | 14.8ms  | 1ms      | 9ms      | 1.4ms    | 4.7ms    |
| analysis exec time (max)   | 96ms    | 47ms     | 79ms     | 47ms     | 64ms     |

Table 1: Summary of results with the flat jitter profile under RM

Interestingly, the situation changes substantially when we use the linear jitter profile, showing the expected strong impact of release jitter. In this case, the tasks with longer period may be affected by a longer release jitter in absolute terms, particularly, longer than the shorter periods in the system. Therefore, the impact of the term  $\max J$  in either tests (2), (3) and (4) can be significant, specially in test (3) in which such term is just divided by the shortest period. Consequently, this test becomes useless in practice whenever there are large release jitters in the system when compared to the shortest period. Conversely, test (4) achieves a performance inferior but closer to the original  $n$  conditions test (2), since the stochastically growing terms of jitter are also divided by growing periods. The best performance with this jitter profile was, however, achieved with test (1).

The results can be observed in Figure 1b and in Table 2. Note that in this case, the larger jitter already causes a very slight difference in the reference test depending on whether it uses 'deadline minus jitter' (Ref1) or rate-monotonic priorities (Ref2). Such difference favours the former case given the optimality of that priority assignment in the presence of jitter [1]. In the Table, the results of

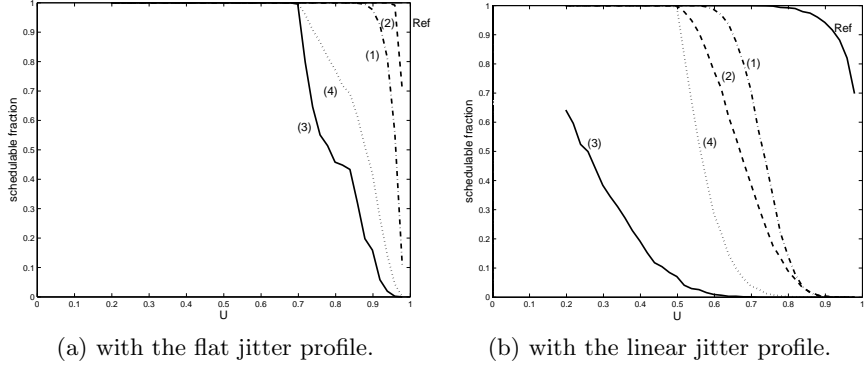


Fig. 2: Performance of different tests under EDF

test (1) are shown with respect to those of reference test Ref1 and those of tests (2) to (4) with respect to reference test Ref2.

|                            | Ref1 & Ref2 | Test (1) | Test (2) | Test (3) | Test (4) |
|----------------------------|-------------|----------|----------|----------|----------|
| % of schedulable task sets |             |          |          |          |          |
| found by the U-based tests | 100%        | 68%      | 50%      | 11%      | 34%      |
| analysis exec time (avg)   | 14.1ms      | 0.9ms    | 6.8ms    | 1.5ms    | 4.3ms    |
| analysis exec time (max)   | 94ms        | 47ms     | 78ms     | 47ms     | 48ms     |

Table 2: Summary of results with the linear jitter profile under RM

The results for EDF scheduling are shown in Figures 2a and 2b for the flat jitter and linear jitter profiles, respectively. In this case, just one reference test was used (Ref) since the test in (6) is independent of the order in which the tasks are considered. The numerical results for both cases are shown in Tables 3 and 4. Basically, we achieved very similar results to those of the RM scheduling case but with a shift to the right due to the higher schedulability capacity of EDF, thus with an increment in the percentages of schedulable configurations found. Anyway, the relative performances of the diverse tests are similar to those achieved with RM.

|                            | Reference | Test (1) | Test (2) | Test (3) | Test (4) |
|----------------------------|-----------|----------|----------|----------|----------|
| % of schedulable task sets |           |          |          |          |          |
| found by the U-based tests | 100%      | 96%      | 99%      | 77%      | 84%      |
| analysis exec time (avg)   | 47ms      | 1ms      | 8.8ms    | 1.5ms    | 4.9ms    |
| analysis exec time (max)   | 159ms     | 47ms     | 79ms     | 47ms     | 62ms     |

Table 3: Summary of results with the flat jitter profile under EDF.

|   | Reference | Test (1) | Test (2) | Test (3) | Test (4) |
|---|-----------|----------|----------|----------|----------|
| % of schedulable task sets found by the U-based tests | 100%      | 69%      | 62%      | 13%      | 49%      |
| analysis exec time (avg)                              | 76ms      | 1ms      | 8.3ms    | 1.5ms    | 4.7ms    |
| analysis exec time (max)                              | 204ms     | 47ms     | 110ms    | 47ms     | 62ms     |

Table 4: Summary of results with the linear jitter profile under EDF

## 5 Sensitivity to release jitter

In order to better assess the impact of the release jitter in the accuracy of the tests presented before, we carried out another set of comparisons in which we specify the release jitter term per task and vary it across a given range. In particular, we considered the two profiles already defined in the previous section, namely *flat* and *linear* jitter, as well as the three more performant tests, namely, tests (1), (2) and (4). In the former profile, all tasks were affected by a similar release jitter value ( $J_i$ ), obtained from a specified value  $J^s$  plus a small uniform random variation with a width equal to the release jitter increment  $J_{inc}$ , i.e.,  $J_i = J^s - J_{inc}/2 + J_{inc} * rand$ . The range considered for  $J^s$  was  $(0, 1]$  time units with increments of 0.05.

In the linear profile, we considered that each task suffers a release jitter  $J_i$  equal to a specified fraction  $\hat{J}^s$  of its period  $T_i$ , again with a uniform random variation bounded by the jitter fraction increment, i.e.,  $J_i = (\hat{J}^s - J_{inc}/2 + J_{inc} * rand) * T_i$ . The range considered for  $\hat{J}^s$  was  $(0, 0.5]$  with fraction increments of 0.05.

The generation of the random synthetic task sets followed the same method as used in the previous section with the exception that the tasks were generated from three groups, with integer periods in the intervals  $[2, 10]$ ,  $[10, 20]$  and  $[20, 40]$  time units. The number of tasks in the first two groups was bounded to 10 and for each  $J^s$  or  $\hat{J}^s$  point we generated 5000 random task sets. The simulation experiments were also carried out using Matlab on a common laptop.

### 5.1 Impact of release jitter with *flat* profile

As referred in the previous section, this profile represents a situation in which all tasks suffer an approximately similar jitter. This can be the consequence of, for example, all tasks being triggered by messages scheduled according to FCFS. Nevertheless, the purpose of this profile is solely to recreate an extreme situation to expose the impact of the release jitter parameter.

One important aspect in this profile, for the sake of the applicability of the considered schedulability tests, is that the maximum jitter value is shorter than the minimum period, or else, all considered tests would fail.

For both tests (2) and (4), this constraint also means we can define a utilization least upper bound for guaranteed schedulability by moving a majorant of the

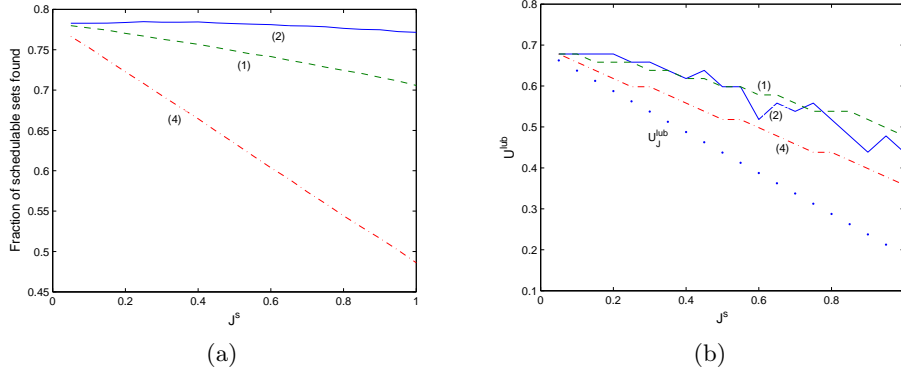


Fig. 3: Sensitivity to release jitter under a flat profile and RM scheduling

jitter term from the left to the right side of the inequalities leading to equation (7). This should, however, be rather pessimistic for tests (2) and (4) given the use of the minimum period instead of the actual period of the tasks. In particular, we expect test (2) to perform well since it takes advantage of the decreasing impact of the jitter term as the tasks periods grow. Note that among its conditions, the utilization terms become more constraining for lower priority tasks, which are compensated with smaller relative jitter terms, while the larger relative jitter terms are accommodated in the conditions of higher priority tasks in which the utilization terms are less constraining.

$$U_J^{lub}(n) = U_{RM,EDF}^{lub}(n) - \frac{\max_{i=1..n} J_i}{\min_{i=1..n} T_i} \quad (7)$$

Figure 3a shows the fraction of task sets marked as schedulable by the reference test (5) that were also deemed schedulable by the tests considered, as a function of the specified release jitter parameter  $J^s$ . The curve of each test is identified by the respective reference number. This figure shows the superiority of test (2) in this profile and the lower performance of the single condition test (4). As a curiosity, note the slight improvement in performance of test (2) when the release jitter grows to 0.4 time units (approx. 20% of the minimum period), which is explained by the fact that the utilization terms in the test conditions grow slower than the decrease of the jitter term.

Figure 3b shows the utilization least upper bound observed in the simulations for each case, which is always above the lower bound in equation (7).

When using EDF scheduling, the relative results are similar to the RM case but in absolute terms the performance is better, as expected (figure 4). In particular, note the even more dominating performance of test (2) in figure 4a, again due to the monotonic reduction of the jitter terms along its  $n$  conditions.

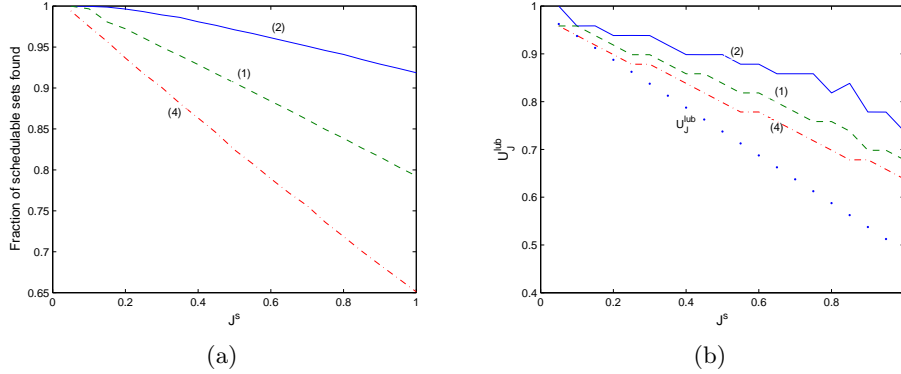


Fig. 4: Sensitivity to release jitter under a flat profile and EDF scheduling

The utilization least upper bounds observed in the simulations (figure 4b) also respect the predicted linear lower bound in equation (7).

## 5.2 Impact of release jitter with *linear* profile

Conversely to the previous case, this profile presents a situation in which all tasks suffer a release jitter that represents an equal share of their period. Thus, longer tasks suffer longer release jitter. In particular, in this profile it is possible that the jitter affecting the longer tasks is longer than the periods of some tasks in the system, a situation that rules out test (3) and is substantially penalizing to both tests (2) and (4), as we will see further on in this section.

In practical terms, this profile can be the consequence of, for example, all tasks being triggered by messages scheduled according to Rate-Monotonic. In this case, tasks with longer periods have lower priority and thus suffer more interference, leading to more release jitter. Again, the purpose of this profile is solely to recreate another extreme situation to expose the impact of the release jitter parameter.

In what concerns the utilization least upper bound, it can also be anticipated in tests (2) and (4) by considering that the jitter term is essentially the jitter fraction  $\hat{J}^s$  increased by the width of the random interval considered in the generation of  $J_i$ . Thus, we expect to observe a linear decrease in the utilization least upper bound in the simulations under this profile, at least for tests (2) and (4). This behavior was actually observed as can be seen in both figures 5b and 6b for RM and EDF scheduling, respectively.

Figure 5a shows the efficiency of the tests under RM scheduling with the linear profile. In this case, since the release jitter fraction is approximately constant for all tasks, test (2) cannot take much advantage from its  $n$  conditions and it ends up performing almost as poorly as test (4). The difference arises

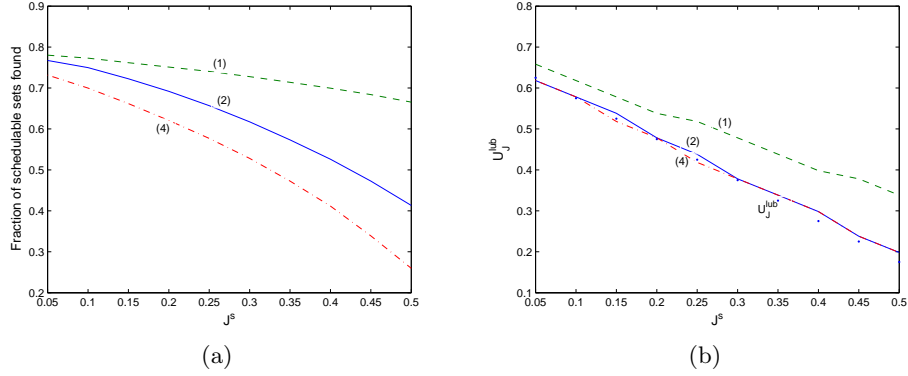


Fig. 5: Sensitivity to release jitter under a flat profile and RM scheduling

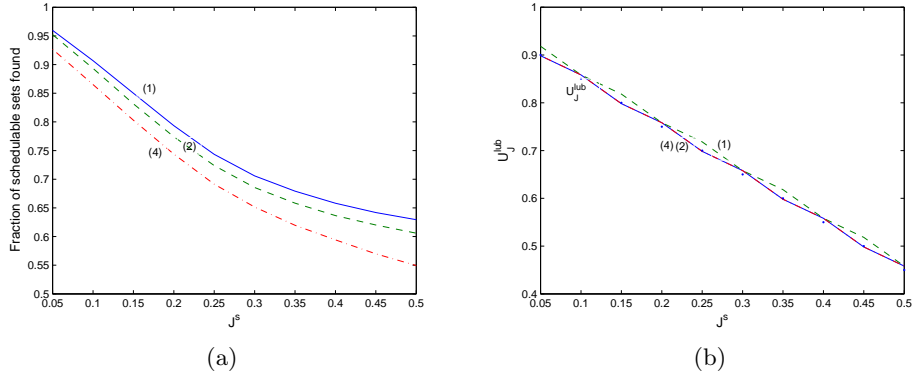


Fig. 6: Sensitivity to release jitter under a linear profile and EDF scheduling

from the small random part of the jitter term that still gives some benefit to test(2). However, in this case, test (1) is the clear dominant one.

Curiously, under EDF scheduling as shown in figure 6, all three tests perform very similarly with a minor dominance of test (2). The most interesting feature to observe in this case is the strong degradation in the tests efficiency caused by the increase in release jitter. Nevertheless, similarly to the flat jitter profile, the efficiency of all three tests is substantially higher under EDF than RM, with about 15% to 20% more schedulable sets found.

### 5.3 Comparative remarks

Overall, with the results obtained in the two profiles, it seems clear that test (1) is more resilient to wider release jitter components, particularly when the task set includes tasks with long period and long release jitter together with tasks

with short periods, shorter than the long release jitters. On the other hand, when the tasks in the set include release jitter terms that are all shorter than the minimum task period, then test (2) is clearly more efficient. Conversely, test (4) is outperformed by the other two in all cases.

However, concerning the execution time, which is a relevant aspect for on-line use as desired, test (1) is the lightest one, followed by test (4) and then (2). Note that while the former two can be executed online in constant time, the latter requires a linear time algorithm.

When considering these tests within dynamic QoS management frameworks in which slack bandwidth needs to be promptly distributed among several executing tasks, then test (4) is the most suited one, supporting one-step bandwidth assignment approaches that divide all the slack among the running tasks with a closed formula under guaranteed schedulability. Both tests (1) and (2) require more elaborate bandwidth assignment approaches to enforce continued schedulability.

Finally, real situations will entail a diversity of cases, with just some tasks suffering release jitter while others do not, some long period tasks with short release jitter terms together with others with long release jitter, etc. Thus, it would be interesting to explore a combined test, joining both tests (1) and (2).

## 6 Conclusion

Growing interest on improving the resource efficiency in embedded systems while increasing functionality and reducing the time to market is pushing towards the use of on-line adaptation and reconfiguration. For example, dynamic bandwidth management, or more generally dynamic QoS management, may allow maximizing the use of a given resource, e.g. cpu or network, by a dynamic set of entities. When these systems have real-time requirements, the on-line adaptations must be carried out with the assistance of an appropriate schedulability analyzer that assures a continued timely behavior but is light to execute.

In this paper we have analyzed several utilization-based schedulability tests that incorporate release jitter. This feature was introduced recently and allows their use in distributed systems where release jitter appears naturally. Moreover, they are particularly suited to be used on-line given their low computational requirements.

Therefore, we carried out an extensive simulation with random task sets to characterize the level of pessimism and computational overhead of several such schedulability tests. We concluded that the two most efficient tests do not dominate each other and thus we defined in which conditions one might be preferable over the other. However, these conditions are not clearly separated thus, in future work, it would be interesting to analyze a possible combination of the tests in a single one. Two other lines also remain open, improving the use of these tests in slack bandwidth distribution under guaranteed schedulability and including blocking terms.



## References

1. A. Zuhily, A. Burns : Optimality of (D-J)-monotonic Priority Assignment. *Information Processing Letters* 103(6), 247–250 (Sept 2007)
2. Almeida, L., Marau, R., Lakshmanan, K., Rajkumar, R.: On the Schedulability Analysis for Dynamic QoS Management in Distributed Embedded Systems. In: *Proc. of the 8th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems(SEUS'10)* (Oct 2010)
3. Audsley, N., Burns, A., Richardson, M., Tindell, K., Wellings, A.J.: Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling. *Software Engineering Journal* 8, 284–292 (1993)
4. Bini, E., Baruah, S.K.: Efficient computation of response time bounds under fixed-priority scheduling. In: *Proc. of 15th Int. Conf. on Real-Time and Networked Systems (RTNS'07)* (Mar 2007)
5. Buttazzo, G.C.: *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications* (Real-Time Systems Series). Springer-Verlag TELOS, Santa Clara, CA, USA (2004)
6. Buttazzo, G.C., Lipari, G., Caccamo, M., Abeni, L.: Elastic Scheduling for Flexible Workload Management. *IEEE Trans. on Computers* 51(3), 289–302 (2002)
7. Davis, R.I., Burns, A.: Response Time Upper Bounds for Fixed Priority Real-Time Systems. In: *Proc. of the 29th Real-Time Systems Symposium (RTSS'08)*. pp. 407–418. IEEE Computer Society, Washington, DC, USA (2008)
8. Fisher, N., Nguyen, T.H.C., Goossens, J., Richard, P.: Parametric Polynomial-Time Algorithms for Computing Response-Time Bounds for Static-Priority Tasks with Release Jitters. In: *Proc. of the 13th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA'07)*. pp. 377–385. IEEE Computer Society, Washington, DC, USA (2007)
9. Ghosh, S., Rajkumar, R.R., Hansen, J., Lehoczky, J.: Scalable QoS-Based Resource Allocation in Hierarchical Networked Environment. In: *Proc. of the 11th IEEE Real Time on Embedded Technology and Applications Symposium (RTAS'05)*. pp. 256–267. IEEE Computer Society, Washington, DC, USA (2005)
10. Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the Association for Computing Machinery* 20(1), 46–61 (1973)
11. Marau, R., Almeida, L., Pedreiras, P., Lakshmanan, K., Rajkumar, R.: Utilization-based Schedulability Analysis for Switched Ethernet aiming Dynamic QoS Management . In: *Proc. of the 15th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA'10)* (Sep 2010)
12. Palencia, J.C., Harbour, M.G.: Response time analysis of EDF distributed real-time systems. *J. Embedded Comput.* 1(2), 225–237 (2005)

# Timing of CAN under the Influence of Random Error Events

Philip Axer, Rolf Ernst

TU Braunschweig, Germany

## 1 Introduction

The Controller Area Network (CAN) [1] is one of the most prominent buses used in various fields (e.g. automotive, industrial, aerospace) today. Despite its age, it has been introduced in the 80's, it is still in use today due to its cost advantage, versatility and robustness against errors. Due to its simplistic nature - CAN is a priority driven serial bus - it is often used for real-time system where the worst-case timing delays of transmissions must be predictable. For example, today's cars feature a rich set of distributed control algorithms which are mapped to Electronic Control Units (ECUs) connected via CAN. Formal real-time analysis methods allow prediction of such networks.

While new high-speed bus systems such as Flexray and Ethernet emerge, it is estimated that CAN will still be used in certain subsystems either due to its price point, its real-time capabilities or simply because of legacy compatibility.

A major advantage of CAN is that it can be used in a wide range of application scenarios due to its scalable nature. For instance, CAN supports time-triggered protocols such as TTCAN as well as event-triggered communication. In addition, as specified by the CAN standard, an off-the-shelf controller already includes an error-detection mechanism based on a cyclic redundancy check and automatic retransmission of messages, so that the CAN software-stack can be kept small. Due to its error robustness, CAN is frequently used in safety critical applications such as active-steering or for controlling heavy industrial machinery.

However, the CAN protocol will most likely detect transmission errors and schedule retransmissions until data is transmitted correctly, transmission latency is then severely affected compared to an error-free transmission. Thus, for hard-real time systems which operate in environments under electromagnetic interference (EMI) such as electric cars, the transmission latency which is predicted by formal real-time analysis based on the absence of errors does not apply anymore. This does also apply to CAN flexible data rate (CAN FD) [8], the third generation protocol. Higher transmission speeds of up to 10 Mbit/s for CAN FD will most likely increase the error rate even further, leading to a strong demand for an analysis approach which also considers the error case.

Deployment in safety-critical domains makes a strongly safety oriented product life-cycle necessary to qualify a product for deployment in safety-critical missions. This is not only crucial in order to minimize the risk of casualties in case of system failure but also ensures product liability. To unify safety requirements, safety standards such as the industrial-oriented IEC-61508 [9] or the automotive domain specific ISO-26262 [10] specify a safety certification process.

IEC-61508 distinguishes between different levels of criticality, called Safety Integrity Level (SIL). Among procedural guidelines, each SIL specifies a minimum Mean Time To Failure (MTTF) as a target reliability value. Hence, if CAN is to be used in a safety critical environment, it must be proven that the MTTF goal can be achieved. The goal of this paper is to derive probabilistic scheduling guarantees for CAN communication under the effects of errors, which can be used in a certification process.

The rest of the paper is structured as follows: First, we summarize related work in Section 2, then we describe the CAN protocol and the generic response time analysis for the error-free and error case in Section 3. In Section 4 the contribution of this work is to revise an existing method to compute probabilistic bounds and prove its correctness. In Section 5, we apply the presented method to an automotive benchmark (SAE benchmark) as well as a realistic frame-set supplied by a major automotive OEM. Finally, we conclude the work in Section 6.

## 2 Related Work on CAN

Worst-case analyses are available for a broad spectrum of schedulers. One class of analysis approaches is the *busy window* analysis [11, 16]. The busy window of a task is defined as a time interval for which a resource executes only tasks of priority greater than or equal to the priority of the task under analysis and during which the resource is never idle [16]. The maximum response time for a CAN frame can then be derived from the busy window [5]. This approach can be applied to predict the worst-case in an *error-free* environment.

In order to include effects of errors (e.g. error signaling and retransmission overhead) different approaches were presented. In [14], an approach is presented to tightly bound the reliability of periodic, synchronized messages. Therefore, a reliability metric  $\mathcal{R}(t)$  is defined which denotes the probability that after time  $t$  the CAN frameset has operated without any deadline misses. Reliability is calculated based on the hyperperiod, which is the time when the activation pattern of a periodic message set repeats itself. Hence, the complexity of the algorithm depends on the amount of activations in the hyperperiod. This algorithm is suitable for automotive message sets in which periods are typically multiples of 10ms and deadlines are given implicitly (i.e. period as deadline). However, if messages are not synchronized, or the relative phasing is unknown the approach is not applicable.

In [3], the busy-window approach is used and a tree-based approach is presented, where different error scenarios are evaluated iteratively. In a second step, these scenarios are translated to probabilities and a *worst-case deadline failure probability* is calculated. The approach was extended in [4], and the tree-based was superseded by a simpler, more accurate approach. However, both methods [4, 3] allow only deadlines smaller than the periods, which is a limit for practical use since bursty CAN traffic is not supported.

In this paper, we extend the approach was in [4] to arbitrary deadlines and arbitrary activation models and apply it to an industrial use-case.

### 3 CAN Protocol and Timing Analysis

The CAN protocol is a multi-master, differential, serial bus. On the physical layer, data is Non-Return-to-Zero (NRZ) encoded. The CAN transceiver output consists of an open-collector or “wired and” circuit, thus the wire can be driven to two states: dominant (0) or recessive (1). All connected transceivers may drive the bus at the same time and the state is determined by the logical AND function of all driver inputs. Thus a CAN bus subscriber can “overwrite” the bus-line by sending a dominant bit.

Data is transmitted in frame entities which are non-preemptable, where each frame consists of the following blocks: A start of frame marker, an 11-bit frame identifier, control field, up to eight data bytes a 15-bit CRC followed by an acknowledgement field and an end-of-frame marker.

An exact protocol description can be found in the official specification [1]. In 1991, CAN 2.0 introduced extended-frames which allow a 29-bit frame identifier. The arbitration scheme uses carrier sense multiple access/bitwise arbitration (CSMA/BA) and is based on the fact that dominant bits “win” the access to the physical medium. Thus, the smaller the CAN bus identifier, the higher the priority of the frame.

The actual length of one frame for  $s$  payload data bytes is not necessarily fixed due to *bit stuffing*, where the controller inserts certain bits in order to avoid long sequences of 1’s and 0’s. In [5] it was shown, that the maximum length in bit times  $t_{bit}$  for one base frame is

$$C = (55 + 10s) t_{bit} \quad (1)$$

and for extended frames as:

$$C = (80 + 10s) t_{bit} \quad (2)$$

#### 3.1 CAN Error Handling

All receiving nodes constantly check for protocol consistency during the transmission of frames. If framing rules are violated (e.g. missing stuffing-bits, missing acknowledgement), or the CRC field does not match the payload, an error can be signaled by all bus subscribers. The CAN standard defines two error frames for this purpose: the active error frame and the passive error frame.

When an error frame is transmitted, other nodes drop the frame and a retransmission of the broken frame is triggered. The worst-case overhead for an error frame can be given as

$$F = 31 t_{bit} \quad (3)$$

Then, after an error frame has been transmitted, the re-transmission has to complete in a new arbitration phase.

The CRC-15 which is used by CAN, is able to detect the following error patterns up to 5 randomly distributed errors, burst errors of length less than 15 and errors of any odd number [13]. Additionally, it is guaranteed that the residual error probability for an undetected corrupted message is smaller than the following threshold, where BER denotes the bit error rate.

$$P_{res} = BER \cdot 4.7 \cdot 10^{-11} \quad (4)$$

Assuming an aggressive bit error rate of  $10^{-6}$ , and a frame period of 10 ms, this translates to an average time until an undetected error of  $5.91 \cdot 10^{10}$  hours. This is sufficiently high for safety critical applications thus we neglect undetected CRC errors in our analysis.

### 3.2 Event Models

Throughout the paper we use *event models* as an abstract model for the activation of CAN message frames. An event model describes the maximum and minimum amount of events  $\eta^+$ ,  $\eta^-$  which arrive during a given time window  $\Delta t$  at the CAN controller and are queued for transmission. Figure 1 shows  $\eta^+$  and  $\eta^-$  on the left. An alternative representation is to use the notion of a minimum and maximum distance which covers at least and at most  $n$  subsequent events  $\delta^-(n)$ ,  $\delta^+(n)$ . We can interpret  $\delta^-(n)$ ,  $\delta^+(n)$  as the distance from the start of the busy window until the earliest and latest arrival of the  $n$ -th event, thus it is well defined for  $n \geq 1$ . Both representations  $\eta$  and  $\delta$  are pseudoinverse and can be converted to each other:

$$\delta^-(n) = \min_{\Delta t \geq 0, \Delta t \in \mathbb{R}} \{ \Delta t \mid \eta^+(\Delta t) \geq n \} \quad (5)$$

$$\eta^+(\Delta t) = \max_{n \in \mathbb{N}, n \geq 1} \{ n \mid \delta^-(n) \leq \Delta t \} \quad (6)$$

For a compact representation, standard event models in [12] use three parameters, event model period  $\mathcal{P}$ , event model jitter  $\mathcal{J}$  and the minimum distance between two events  $d^{min}$ . The  $\eta^+$  function for a bursty input is defined as:

$$\forall \Delta t > 0 : \eta^+(\Delta t) = \min \left( \left\lceil \frac{\Delta t}{d^{min}} \right\rceil, \left\lceil \frac{\Delta t + \mathcal{J}}{\mathcal{P}} \right\rceil \right) \quad (7)$$

The standard event models are applicable to many typical real-time setups, for instance in the automotive domain where periodic systems are predominant. Later we will see, that also traces can be used to obtain accurate event models.

### 3.3 Response Time Analysis

The response time is the interval from message arrival at the CAN controller until it is fully transmitted over the bus. In hard real-time systems, the response times for all

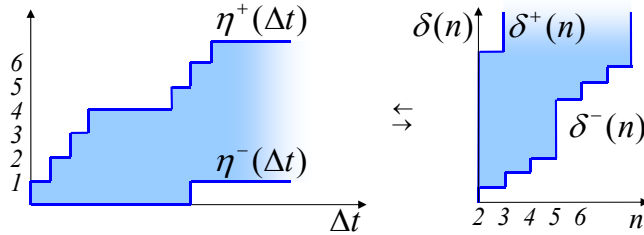


Fig. 1: Event Model Representation

activations of a task  $\tau_i$  must be smaller than a given deadline  $D_i$ . In order to show that all observed response times are actually smaller than  $D_i$ , it is necessary to derive the worst-case response time (WCRT).

For the timing analysis, the concept of the *busy window* or *busy period* [11] is used. Similar to [5], we define the level- $i$  busy window as the time the bus is busy transmitting messages of priority  $i$  or higher. The largest level- $i$  busy window can be constructed under the following conditions: No messages of priority  $i$  or higher are queued right before the beginning of the level- $i$  busy window. The largest level- $i$  busy window is initiated with the *critical instant*, that is all tasks are released simultaneously and thus create the highest load possible. All following activations are then released as early as possible according to  $\eta^+$ .

The worst-case queuing delay for message  $\tau_i$  occurs when the largest message of lower priority with transmission time  $B_i$  was released right before the start of the critical instant, so that all higher priority activations are delayed by  $B_i$  at most.

$$B_i = \max_{\forall k \in lp(\tau_i)} (C_k) \quad (8)$$

Under these assumptions, the busy window of a frame  $\tau_i$  is then given by the following recursive equation.

$$w_i = B_i + \sum_{j \in hp(\tau_i)} C_j \cdot \eta^+(w_i) \quad (9)$$

Here the busy window is the sum of the blocker and all messages of higher or equal priority of  $\tau_i$  which are released in the busy window  $w_i$ . This fixed point problem can be solved by the following recurrence relation:

$$w_i^{n+1} = B_i + \sum_{j \in hp(\tau_i)} C_j \cdot \eta^+(w_i^n) \quad (10)$$

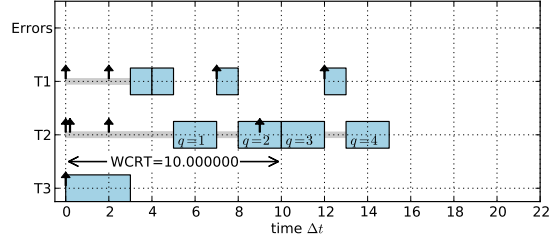
starting with  $w_i^0 = C_i$ , since at least  $C_i$  is to be transmitted. The iteration can be stopped once the smallest fixed-point  $w_i^{n+1} = w_i^n$  is found.

As shown in [5], any instance of message  $\tau_i$  released in the level- $i$  busy window can potentially lead to the worst response time. Thus it is necessary to check all  $q_{max}$  activations in the busy window for their response times,

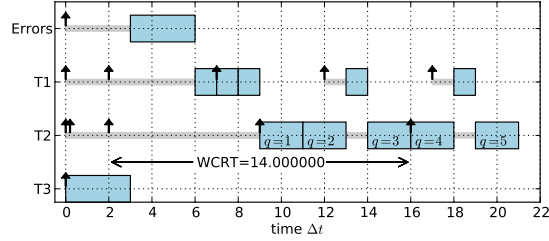
$$q_{max} = \eta_i^+(w_i) \quad (11)$$

The finishing time of the  $q$ -th activation can be calculated similarly to the busy window by another recurrence relation. The first release of a frame  $\tau_i$  in the busy window corresponds to  $q = 1$ , and the last message in the busy window is  $q = q_{max}$ . An activation of task  $\tau_i$  can start transmission when all higher priority messages  $hp(\tau_i)$  and all previously queued activations of  $\tau_i$  have been transmitted. Thus, the waiting time until the  $q$ -th activation starts transmitting is given by:

$$w_i(q) = B_i + (q-1)C_i + \sum_{j \in hp(\tau_i)} C_j \cdot \eta^+(t_{bit} + w_i(q)) \quad (12)$$



(a) Error-Free case,  $K = 0$ . WCRT is obtained for  $q = 2$ .



(b) Error condition,  $K = 1$ . WCRT is obtained for  $q = 3$ .

Fig. 2: Illustrative example for a level-i busy window for  $K = 1$  and  $K = 2$ .

The extra time  $t_{bit}$  is to account for edge effects, in case a message is to be transmitted  $\varepsilon$  time after a bit-time boundary [2]. Similarly to the level-i busy window, we can also formulate an iterative algorithm for this fixed-point problem which can be solved by starting with  $w_i^0(q) = B_i$ .

$$w_i^{n+1}(q) = B_i + (q-1)C_i + \sum_{j \in hp(\tau_i)} C_j \cdot \eta^+(t_{bit} + w_i^n(q)) \quad (13)$$

The finishing time is then the waiting time plus the transmission time  $w_i(q) + C_i$ . The response time of the  $q$ -th event is the relative time from the release of the  $q$ -th event until it arrives at the receiver.

$$r_i(q) = w_i(q) + C_i - \delta^-(q) \quad (14)$$

Therefore, the largest response time of these candidates is the worst-case response time for the message  $\tau_i$ .

$$R_i = \max_{1 \leq q \leq q_{max}} r_i(q) \quad (15)$$

### 3.4 Response Time Analysis with Errors

The analysis as presented in the previous section does not cover the effect of transmission errors. Obviously, error events trigger the transmission of an error frame as well as a retransmission which in turn increases the busy window and therefore the response time of a message. Additionally, during the time required for error handling, additional

higher priority messages could be released which further increases the busy window and the response time.

Another aspect of an increased busy window is that a larger busy window raises the likelihood that further errors fall into the same window. To account for these effects it is mandatory to model the occurrence pattern of errors. We use the Poisson model as used in previous work (e.g. in [4]). A Poisson process is memoryless, that means that the probability of the occurrence of error events during some time window which starts at  $t_0$  is independent of the history previous to  $t_0$ . Practically, that means a Poisson process models independent single bit errors (without bursts), where  $\lambda$  specifies the bit error rate. The probability for the occurrence of  $m$  error-events in the time window  $\Delta t$  is:

$$p(m, \Delta t) = \frac{e^{-\lambda \Delta t} (\lambda \Delta t)^m}{m!} \quad (16)$$

As discussed before, the error penalty from which frame  $\tau_i$  suffers in case of one error event is composed of the protocol overhead of error signaling  $F$  plus one retransmission of the largest frame of equal or higher priority:

$$E_i = F + \max_{j \in \text{hep}(\tau_i)} C_j \quad (17)$$

Consequentially, the worst-case overhead for  $K$  errors can be bounded to:

$$E_{i|K} = K \cdot E_i \quad (18)$$

Given  $K$  errors occur during the transmission of frame  $\tau_i$  with a corresponding error overhead of  $E_{i|K}$ , the formula for the level- $i$  busy window can easily be adapted by include the error as an additional blocker term. The  $K$ -error, level- $i$  busy window  $w_{i|K}$  is then defined as:

$$w_{i|K} = E_{i|K} + B_i + \sum_{j \in \text{hep}(\tau_i)} C_j \cdot \eta^+(w_i) \quad (19)$$

For each  $K$ -error scenario, it is necessary to evaluate which activation leads to the worst-case response time. Similarly to the error-free case, we add the  $E_{i|K}$  to equation 13 to obtain the waiting time for the  $q$ -th activation in the  $K$ -error case.

$$w_{i|K}^{n+1}(q) = E_{i|K} + B_i + (q-1)C_i + \sum_{j \in \text{hp}(\tau_i)} C_j \cdot \eta^+(t_{\text{bit}} + w_{i|K}^n(q)) \quad (20)$$

This is depicted in Figure 2 in an exemplary Gantt chart for a bus with three frames  $F_1, F_2, F_3$  with increasing transmission times and bursty event arrival. Figure 2a shows the level-2 busy window for the error-free case ( $K=0$ ). In this scenario, 4 activations of frame  $F_2$  have to be considered as candidates and their evaluation leads to the worst-case  $R_{i|K=0} = 10$  for  $q = 2$ . Contrary to the error-free case Figure 2b shows a scenario with one error ( $K=1$ ). Here, not activation  $q = 2$  but instead activation  $q = 3$  leads to the worst-case response time  $R_{i|K=1} = 14$ . Analogous to the error-free case, we can derive the response time for each error scenario by checking each activation in the  $K$ -error busy-window.

$$r_{i|K}(q) = w_{i|K}(q) + C_i - \delta^-(q) \quad (21)$$



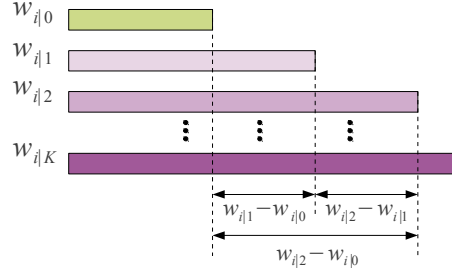


Fig. 3: Possible level-i, K-error busy windows

$$q_{\max|K} = \eta_i^+(w_{i|K}) \quad (22)$$

$$R_{i|K} = \max_{1 \leq q \leq q_{\max|K}} r_{i|K}(q) \quad (23)$$

#### 4 Probabilistic Timing Analysis Revisited

It is then possible to precompute all K-error, level-i busy windows until a threshold criterion is reached (e.g. until the deadline is exceeded,  $w_{i|K} > D_i$ ). This stopping condition will eventually be reached in a finite amount of analysis steps, since the sequence of  $w_{i|K}$  is strictly increasing function in  $K$  as shown in Figure 3.

Up to this point we have calculated the response times  $R_{i|K}$  and the level-i busy window for each K-error scenario  $w_{i|K}$ . The remaining problem is to calculate the probability that a busy window of length  $w_{i|K}$  actually occurs.

Now we revise the method to derive the probabilities for the busy-window probabilities as presented in [4]. For the following argumentation it is important to note the difference between error-events (the actual bit error) and a retransmission event. It is possible that a message of length  $C$  is hit by multiple error-events and only one retransmission occurs (e.g. after reception when the CRC is checked), but it is assumed, that in the worst-case condition, each error-event will lead to exactly one retransmission. In the following steps, we use the notation  $P(w_{i|K})$  which is an upper bound on the probability that a K-error busy window is observed. Thus we can directly use Equation 16 to obtain the probability that no error-events occur during a given time window. E.g. the probability for the error-free case is:

$$P(w_{i|0}) = p(0, w_{i|0}) = e^{-\lambda w_{i|0}} \quad (24)$$

For  $K > 0$  it is not sufficient to just calculate  $p(K, w_{i|K})$ , because error-events have to occur in certain segments of the busy window. For instance,  $w_{i|1}$  will only occur if exactly one retransmission occurs during the time interval  $[0, w_{i|0})$ . Similarly,  $w_{i|2}$  will only occur, in either of the following two scenarios: two retransmissions occur in the interval  $[0, w_{i|0})$  or, one retransmission in  $[0, w_{i|0})$  and one in  $[w_{i|0}, w_{i|1})$ . In [4] it was shown, that the amount of combinations where retransmission can occur in order to

generate a  $w_{i|K}$  busy window is given by the Catalan Series which grows rapidly with  $K$ , thus exhaustive enumeration is not possible.

Thus in [4] a more efficient technique was proposed, which can also apply for the general case in which a busy-window includes multiple queued activations which can be affected by errors. However, it was not argued how that method is pessimistic.

The approach works as follows: One error-event in the entire busy-window  $w_{i|1}$  can happen in two ways. The error may actually lead to a  $w_{i|1}$  busy window with the probability  $P(w_{i|1})$ . Or, we face a busy window of length  $w_{i|0}$  and the error event occurs in the interval  $[w_{i|0}, w_{i|1})$ . These intervals are also highlighted in Figure 3.

$$p(1, w_{i|1}) = P(w_{i|1}) + P(w_{i|0})p(1, w_{i|1} - w_{i|0}) \quad (25)$$

The value of  $P(w_{i|1})$  can then be obtained by rearranging the equation. Similarly we can apply this idea to  $K = 2$ . Two errors in the time window  $w_{i|2}$  may occur in the following mutually exclusive ways.

1. in a way that a busy window of length  $w_{i|2}$  actually occurs assuming two error-events with the probability  $P(w_{i|2})$ .
2.  $w_{i|1}$  occurred which implies exactly one error in  $w_{i|1}$  and the second error must then happen in the interval  $[w_{i|1}, w_{i|2})$ .
3.  $w_{i|0}$  occurred which implies no error in  $w_{i|0}$  and exactly two errors must be in the interval  $[w_{i|0}, w_{i|2})$ .
4. in a way that a busy window of length smaller than  $w_{i|2}$  occurs because the error events are too close to hit two different frames.

In previous work [4], the latter case was not sufficiently considered and needs further elaboration.

$$p(2, w_{i|2}) = P(w_{i|2}) + P_{res|2}(w < w_{i|2}) + P(w_{i|1})p(1, w_{i|2} - w_{i|1}) + P(w_{i|0})p(2, w_{i|2} - w_{i|0}) \quad (26)$$

In fact, the previously computed value for  $P(w_{i|1})$  and the term  $P_{res|2}(w < w_{i|2})$  are mutually exclusive. This is because  $P(w_{i|1})$  was derived under the assumption that exactly one error-event triggers one retransmission and the residual term  $P_{res|2}(w < w_{i|2})$  includes all other cases by which two error-event trigger exactly one retransmission and thus lead to  $w_{i|1}$ . We introduce  $\tilde{P}(w_{i|2})$  which is the probability that either a  $w_{i|2}$  busy window occurs or a smaller busy window occurs (e.g. two error-event hit one frame in  $w_{i|0}$ ).

$$\tilde{P}(w_{i|k}) = P(w_{i|k}) + P_{res|k}(w < w_{i|k}) \quad (27)$$

By similar reasoning we can distribute three error-events in the busy window  $w_{i|3}$ :

1. in a way that a busy window of length  $w_{i|3}$  actually occurs assuming three error-events with the probability  $P(w_{i|3})$ .

2.  $w_{i|2}$  or a busy window even smaller than  $w_{i|2}$  occurs which both imply 2 error-events in  $w_{i|2}$ , the third error-event then must happen in the interval  $[w_{i|2}, w_{i|3})$ .
3.  $w_{i|1}$  occurred, which implies exactly one error event in  $w_{i|1}$ , the second error then must happen in the interval  $[w_{i|1}, w_{i|2})$ .
4.  $w_{i|0}$  occurred, which implies that no error is in the interval  $[0, w_{i|0})$  and exactly two errors are in the interval  $[w_{i|0}, w_{i|2})$ .
5. in a way that a busy window of length smaller than  $w_{i|3}$  occurs because the three error events are too close to hit three different frames.

$$\begin{aligned}
 p(3, w_{i|3}) = & P(w_{i|3}) + P_{res|3}(w < w_{i|3}) \\
 & + \tilde{P}(w_{i|2})p(1, w_{i|3} - w_{i|1}) \\
 & + P(w_{i|1})p(2, w_{i|3} - w_{i|1}) \\
 & + P(w_{i|0})p(3, w_{i|3} - w_{i|0})
 \end{aligned} \tag{28}$$

The value for  $\tilde{P}(w_{i|3})$  can be retrieved by rearranging the equation.

$$\begin{aligned}
 \tilde{P}(w_{i|3}) = & p(3, w_{i|3}) - \tilde{P}(w_{i|2})p(1, w_{i|3} - w_{i|1}) \\
 & - P(w_{i|1})p(2, w_{i|3} - w_{i|1}) \\
 & - P(w_{i|0})p(3, w_{i|3} - w_{i|0})
 \end{aligned} \tag{29}$$

The same argument is valid for the following K-error busy windows and Equation 28 is generalized into the following form:

$$\tilde{P}(w_{i|K}) = p(K, w_{i|K}) - \sum_{j=0}^{K-1} \tilde{P}(w_{i|j})p(K-j, w_{i|K} - w_{i|j}) \tag{30}$$

Contrary to a statement in [4],  $\tilde{P}(w_{i|K})$  is not an upper bound for the probability of a busy window of length  $w_{i|K}$  since there are obviously error scenarios which also contribute to  $w_{i|K}$  with more than K error-events. This “missing” probability is implicitly considered in later iterations:  $\tilde{P}(w_{i|j}) | \forall j > K$ . And accounted as if it would lead to a larger busy-window. In that sense,  $\tilde{P}(w_{i|K})$  includes some probability terms that also smaller busy-windows than  $w_{i|K}$  occur. Thus, it does not make sense to use the values of  $\tilde{P}(w_{i|K})$  directly.

Instead, for hard real-time systems it is important to bound the probability that a response time threshold (i.e. the deadline) is exceeded. Therefore, we introduce the worst-case response time exceedance function  $P^+[R_i > r]$  which gives an upper bound that some observed response time  $R_i$  exceeds the threshold  $r$ .

**Theorem 1.** *The worst-case response time exceedance function can be calculated as:*

$$P^+[R_i > r] = 1 - \sum_{\forall K | R_{i|K} < r} \tilde{P}(w_{i|K}) \tag{31}$$

*Proof.* The probability that a response time exceeds the threshold  $r$  is the converse probability that the response time is below that threshold. Each worst-case busy window has one dedicated worst-case response time for message  $\tau_i$  and both  $w_{i|K}$  and  $R_{i|K}$  are monotonic increasing functions in  $K$ , thus it is sufficient to include the first  $K$  busy windows until the response time exceeds the threshold. In order to maximize  $P^+[R_i > D_i]$ , it must be shown that the following series is a valid *lower probability bound* to the occurrence of all possible response times less than  $r$ .

$$P^-[R_i \leq r] = \sum_{\forall K | R_{i|K} < r} \tilde{P}(w_{i|K}) \quad (32)$$

The probability  $\tilde{P}(w_{i|0})$  is the exact probability according to the model that a response time of length  $R_{i|0}$  occurs. By construction, the probability  $\tilde{P}(w_{i|1})$  is a valid lower bound for the probability that the response time  $R_{i|1}$  occurs (only 1 error-event is considered). By construction, the probability  $\tilde{P}(w_{i|K})$  is a valid lower bound for the probability that a response time less than  $R_{i|K}$  occurs: According to Equation 27, it can be decomposed into a lower bound probability that exactly  $w_{i|K}$  occurs and a probability term which covers smaller response times which were not considered in previous probability terms  $\tilde{P}(w_{i|j}) | \forall j < K$ . Since all terms in the series from equation 32 are mutually exclusive and lower bounds to  $P^-[R_i \leq r]$ , the entire series is a lower bound. Then equation 31 yields a valid upper bound  $P^+[R_i > r]$

#### 4.1 Reliability

Based on the exceedance function  $P^+[R_i > r]$ , it is possible to calculate the reliability of individual frames similarly to [14]:

$$\mathcal{R}_i(t) = P^+[R_i > r]^{\eta^+(t)} \quad (33)$$

Based on the reliability function, other common metrics such as the Mean Time To Failure (MTTF) can be computed:

$$MTTF_i = \int_{t=0}^{\infty} P^+[R_i > r]^{\eta^+(t)} \quad (34)$$

## 5 Experiments

### 5.1 SAE Benchmark

We implemented the presented algorithm in pyCPA [6], a pragmatic Python implementation of compositional performance analysis. To evaluate the approach, we use a modified version of the 17-messages SAE benchmark as presented in [15]. The benchmark includes sporadic messages, as well as periodic messages. Sporadic messages are modeled by assuming a minimum interarrival time, also we assume that the CAN bus is part of a larger distributed, automotive network and the data which ought to be transmitted has been processed on different ECUs which results in an increased released

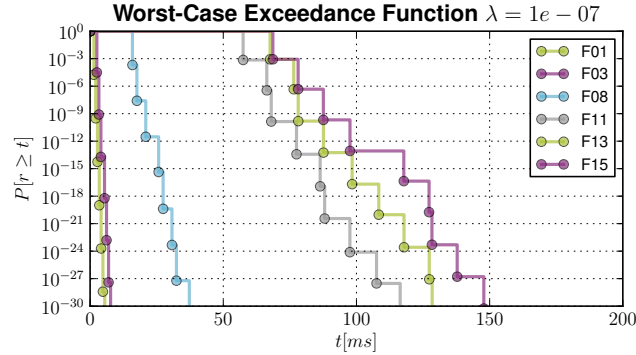


Fig. 4: Worst-case exceedance function denotes the probability that frame exceeds a given threshold response-time.

Table 1: Excerpt from the message set and corresponding deadline exceedance probability for  $\lambda = 10^{-7}$

| Name | Bytes | Prio | P [ms] | J [ms] | D [ms] | $P^+[R > D]$ |
|------|-------|------|--------|--------|--------|--------------|
| F01  | 1     | 1    | 50     | 1      | 5      | $7.4e-34$    |
| F03  | 1     | 3    | 5      | 2      | 10     | $1.9e-46$    |
| F08  | 1     | 8    | 10     | 1      | 5      | $2.1e-04$    |
| F11  | 1     | 11   | 50     | 10     | 10     | $7.2e-04$    |
| F13  | 1     | 13   | 100    | 3      | 100    | $9.5e-21$    |
| F15  | 3     | 15   | 500    | 4      | 1000   | $< 1e-50$    |

jitter (e.g. due to scheduling on upstream ECUs). Thus, the used message set covers a broader spectrum and may be more applicable to today's automotive networks. This is, for some frames we relaxed the deadline and increased the jitter to 1ms. Besides from that, the benchmark was used as it is. We carried out the following experiment using a 125 kbit/s CAN bus and a bit error rate of  $10^{-7}$ , which was measured by [7] in an aggressive environment. The results for selected frames are shown in Figure 4. The diagram shows the worst-case exceedance function which is the probability that a response-time is exceeded. For some frames (e.g. F08, F11), deadline misses due to retransmissions are quite common and may happen in average in one of 10.000 message transmissions, which means that approximately each 10 seconds for frame F08 misses its deadline. Obviously, the presented analysis is easily applicable to the SAE benchmarks as it quickly provides bounds on deadline-miss probability for each frame (cf. Table 1).

## 5.2 OEM Trace Data

Previously, we applied the approach to the SAE benchmark. As discussed, the benchmark consists of messages which are of periodic nature. However, the benchmark is old and does not resemble today's automotive communication patterns.

Today's ECUs installed in contemporary upper class or premium vehicle use various communication modes to transmit frames. This includes transmission patterns such as *cyclic+spontaneous* in which a periodic base-load is superimposed with spontaneous message transmissions. Some frames are transmitted in a *dual cyclic* pattern, where a frame is transmitted either in a "fast" mode or in a slow mode depending of the ECUs state. For a worst-case analysis, these arrival patterns can obviously conservatively approximated by either using a minimum inter-arrival time and the fast period, respectively. However, using such approximations quickly renders the configuration unschedulable.

In some configurations, safety relevant frames such as the airbag signaling are transmitted multiple times in a row (i.e. *sporadic with repetition*). Such a transmission pattern cannot be approximated easily by a periodic with jitter model.

A pragmatic way to solve this problem is to compute a *derived* worst-case behavior, which is built on previously observed behavior of the system or its components. The idea is to carry out multiple of isolated tests and fuse the derived data to carry out a formal analysis. Obviously, this only works if individual tests contain substantial and reliable testing data. It's up the designer to decide when sufficient data is gathered.

For the following analysis, we used test traces supplied by a major automotive OEM. These traces contain the activation times of all instances of individual frames as well as their ID and DLC. In the first step, we derived event models from the traces according to the following equation:

$$\delta_{derived}^-(n) = \min_{\forall m} (\sigma(m) - \sigma(m+n-1)) \quad (35)$$

Here,  $\sigma(m)$  is the absolute arrival time of the  $m$ -th event recorded by the trace tool. By applying eq. 6 we gain a derived  $\eta^+$  which resembles the worst-case as found in the traces. This  $\eta$  function was then fed into the presented analysis to derive the worst-case exceedance function.

This experiment contains 238 standard CAN frames which were captured on a 125kbit/s bus. From the trace data we can observe that the bus was heavily utilized with approximately 80%. The worst-case response times without considering errors is depicted in Figure 5. It can be seen that the frameset contains multiple low-latency

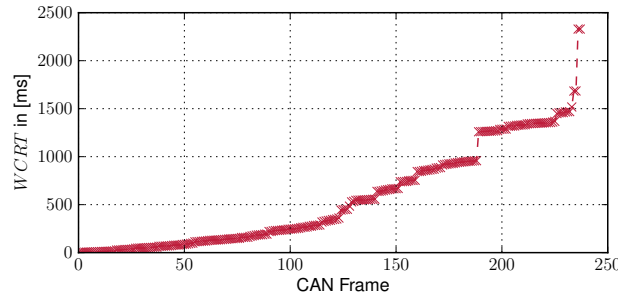


Fig. 5: Worst-case response time. Frames are sorted by priority.

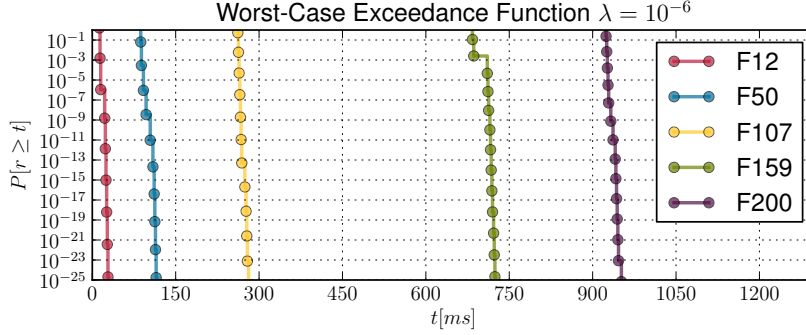


Fig. 6: Worst-case exceedance function for a typical frameset of a contemporary premium vehicle. Frames are sorted by priority.

frames with response times below 50ms. However, the majority of the frames exhibit a rather large response time up to 2 seconds in worst-case conditions.

Now, we evaluate a subset of the frames with respect to their timing under error effects. We assumed a rather harsh environment with a bit error rate of  $10^{-6}$  and investigated representative frames from the frameset (high, medium, low priorities). The result is shown in Figure 6. Interestingly, the probability for increased response times drops extremely fast. And it must be noted, that the drop is even more in reality, since a lower bit error rate can be assumed in reality.

Since no deadlines were supplied, we cannot compute a reliability value. But based on the exceedance functions, we can observe that only in rare cases (approximately with a probability of  $10^{-20}$ ) the error-free worst-case response time is exceeded by more than 50 ms.

## 6 Conclusion

Probabilistic bounds under the influence of errors are especially important for to prove the correctness of safety-critical systems. In this paper, we generalized methods to compute probabilistic bounds on hard real-time CAN messages for response times greater than deadlines and arbitrary event models. We applied the approach to the rather outdated SAE benchmark as well as to frameset supplied by major automotive OEM. We could prove the existing probability bounds to be correct also for the general case. The error analysis as provided in this paper, is with minor modifications applicable to the CAN evolution called CAN FD. Since CAN FD has an increased bitrate of 10 Mbit/s and tighter physical layer timing, it is likely that the bit error rate is higher compared to CAN. Hence, we anticipate that such an analysis becomes even more important with the advent of CAN FD.

However, there are still open questions. Currently, the presented approach cannot handle burst-errors as it relies on the memorylessness of the error model. Also, the assumption of the critical instant for each activation is utterly pessimistic and can po-

tentially be resolved by considering the amount of critical instances that can actually occur. Such an analysis technique remains future work.

## References

1. ISO 11898-1:2003 - Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling, 2003.
2. Iain John Bate. *Scheduling and timing analysis for safety critical real-time systems*. Citeseer, 1999.
3. I. Broster, A. Burns, and G. Rodríguez-Navas. Probabilistic analysis of CAN with faults. In *Proc. of Real-Time Systems Symposium*, pages 269–278. IEEE, 2002.
4. I. Broster, A. Burns, and G. Rodríguez-Navas. Comparing real-time communication under electromagnetic interference. In *Proc. 16th ECRTS*, pages 45–52, 2004.
5. R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.
6. Jonas Diemer and Philip Axer. pyCPA - a pragmatic python implementation of compositional performance analysis. <http://code.google.com/p/pycpa>.
7. J. Ferreira, A. Oliveira, P. Fonseca, and J. Fonseca. An experiment to assess bit error rate in can. *Proc. of RTN*, 2004.
8. Florian Hartwich. Can with flexible data-rate. In *Proc. of ICC*, Hambach Castle, Germany, 2012.
9. International Electrotechnical Commission (IEC). Functional safety of electrical / electronic / programmable electronic safety-related systems, 1998.
10. International Organization for Standardization (ISO). Iso/fdis 26262: Road vehicles – functional safety, 2011.
11. J. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. *Proc. 11th RTSS*, pages 201–209, Dec 1990.
12. Kai Richter. *Compositional scheduling analysis using standard event models*. PhD thesis, TU Braunschweig, 2005.
13. Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart. CAN Specification version 2.0, 1991.
14. M. Sebastian and R. Ernst. Reliability Analysis of Single Bus Communication with Real-Time Requirements. In *Proc. of PRDC*, pages 3–10, 2009.
15. K. Tindell and A. Burns. Guaranteeing message latencies on control area network (can). In *Proceedings of the 1st International CAN Conference*. Citeseer, 1994.
16. K. W. Tindell, A. Burns, and A. J. Wellings. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems*, 6(2):133–151, 1994.



# Independence - a misunderstood property of and for probabilistic real-time systems

Liliana Cucu-Grosjean

INRIA

## 1 Introduction

The arrival of more complex and modern architectures for critical embedded systems imposes the utilisation of analyses overcoming the requirement to have a certain knowledge of the worst-case execution time (WCET) of a task. The probabilistic<sup>1</sup> approach is one of the approaches offering such alternative. Since the seminal paper of Lehoczky [16], probabilistic reasoning has been the basis for response time analysis [9], worst-case execution time estimation [10] or for the proposition of optimal priority assignment policies [21]. Probabilistic approaches take into account the fact that extreme values for parameters like worst-case execution time or minimal inter-arrival time are rare. In this paper we discuss results for worst-case execution times. The idea that the worst-case execution time of a task may have a low probability of occurrence is first presented by Burns and Edgar [4] and it is later confirmed by results [14], [28], [7] heavily inspired by this first work [4].

In addition to the proposition of a new technique for estimating the WCET of tasks, the work of Burns and Edgar proposes the original definition of what is currently known as probabilistic worst-case execution time (*pWCET*) estimate (a formal definition is provided in Section 2.2). This notion is different from the probabilistic execution times (*pETs*) [8] and the *pWCET* may be used as input to the probabilistic schedulability analyses like [9] [2] [26] [11] [1]. This difference between *pWCET* and *pET* has an important consequence on the independence between the random variables describing the (WC)ET.

In this paper we show for the first time that the probabilistic real-time analyses do not have stronger requirements from the task systems than a deterministic real-time analysis (with respect to the hypothesis of independence originally introduced by Liu and Layland [17]), as long as *pWCETs* are used.

This paper is organized as follows. In Section 2 we introduce the notations, the definition of *pWCET* and *pET* and their relation. The three notions of independence are detailed and discussed in Section 3. In Section 4 we present the related work with respect to this paper and we conclude in Section 5.

---

<sup>1</sup> The notion of probabilistic is used here in a wide manner to indicate approaches for systems of tasks with at least one parameter described by a random variable. These approaches may belong to the realm of probability theory, statistics or stochastic processes.

## 2 Notations

We consider a task set of  $n$  tasks  $\{\tau_1, \tau_2, \dots, \tau_n\}$ . Each task  $\tau_i$  is characterized by four parameters  $(O_i, \mathcal{C}_i, T_i, D_i)$  where  $O_i$  is the arrival of  $\tau_i$ ,  $T_i$  is the minimal inter-arrival time (commonly known as period) and  $D_i$  the relative deadline. The variable  $\mathcal{C}_i$  is a random variable<sup>2</sup> and it describes the probabilistic worst-case execution time (pWCET) of task  $\tau_i$ . We denote by  $\mathcal{C}_i^j$  the probabilistic execution time (pET) of the  $j^{th}$  job of  $\tau_i$ .

A random variable  $\mathcal{X}$  has associated a probability function (PF)  $f_{\mathcal{X}}(\cdot)$  with  $f_{\mathcal{X}}(x) = P(\mathcal{X} = x)$ . The possible values  $X^0, X^1, \dots, X^k$  of  $\mathcal{X}$  belong to the interval  $[x^{min}, x^{max}]$ , where  $k$  is the number of possible values of  $\mathcal{X}$ . In this paper we associate the probabilities to the possible values of a random variable  $\mathcal{X}$  by using the following notation

$$\mathcal{X} = \begin{pmatrix} X^0 = X^{min} & X^1 & \dots & X^k = X^{max} \\ f_{\mathcal{X}}(X^{min}) & f_{\mathcal{X}}(X^1) & \dots & f_{\mathcal{X}}(X^{max}) \end{pmatrix}, \quad (1)$$

where  $\sum_{j=0}^k f_{\mathcal{X}}(X^j) = 1$ . A random variable may be also specified using its cumulative distribution function (CDF)  $F_{\mathcal{X}}(x) = \sum_{z=x^{min}}^x f_{\mathcal{X}}(z)$ .

### 2.1 Probabilistic Execution Time

**Definition 1.** *The probabilistic execution time (pET) of the job of a task describes the probability that the execution time of the job is equal to a given value.*

For instance the  $j^{th}$  job of a task  $\tau_i$  may have a pET

$$\mathcal{C}_i^j = \begin{pmatrix} 2 & 3 & 5 & 6 & 105 \\ 0.7 & 0.2 & 0.05 & 0.04 & 0.01 \end{pmatrix} \quad (2)$$

If  $f_{\mathcal{C}_i^j}(2) = 0.7$ , then the execution time of the  $j^{th}$  job of  $\tau_i$  has a probability of 0.7 to be equal to 2.

### 2.2 Probabilistic Worst-case Execution Time

**Definition 2.** *The probabilistic worst-case execution time (pWCET) of a task describes the probability that the worst-case execution time of that task does not exceed a given value.*

For instance, a task  $\tau_i$  may have a pWCET

$$\mathcal{C}_i = \begin{pmatrix} 2 & 3 & 105 \\ 0.8 & 0.19 & 0.01 \end{pmatrix} \quad (3)$$

If  $f_{\mathcal{C}_i}(2) = 0.8$ , then the worst-case execution time of  $\tau_i$  has a probability of 0.2 to be larger than 2.

---

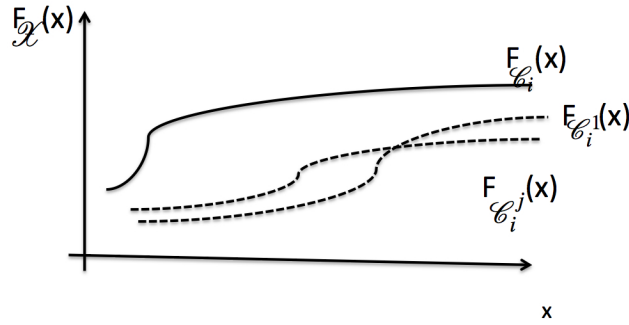
<sup>2</sup> In this paper we use a calligraphic typeface to denote random variables, e.g.,  $\mathcal{X}, \mathcal{C}$ , etc.

### 2.3 Relation between pWCET and pET

The relation between the pWCET of a task and the pETs of all jobs of a task is defined using the relation  $\succeq$  provided in Definition 3.

**Definition 3.** [18] Let  $\mathcal{X}$  and  $\mathcal{Y}$  be two random variables. We say that  $\mathcal{X}$  is worse than  $\mathcal{Y}$  if  $F_{\mathcal{X}}(x) \leq F_{\mathcal{Y}}(x)$ ,  $\forall x$ , and denote it by  $\mathcal{X} \succeq \mathcal{Y}$ .

The pWCET  $\mathcal{C}_i$  is an upper bound on the pETs  $\mathcal{C}_i^j$ ,  $\forall j$  and it may be described by the relation  $\succeq$  as  $\mathcal{C}_i \succeq \mathcal{C}_i^j$ ,  $\forall j$ . Graphically  $F_{\mathcal{C}_i}(\cdot)$  never goes above the curves  $F_{\mathcal{C}_i^j}(\cdot)$ ,  $\forall j$ . This relation is illustrated in Figure 1. One may notice that  $\mathcal{C}_i^1$  and  $\mathcal{C}_i^j$  are not comparable with respect to the relation  $\succeq$  defined by Definition 3.



**Fig. 1.** Relation between the CDF of the pWCET  $\mathcal{C}_i$  and the CDFs of the pETs  $\mathcal{C}_i^1$  and  $\mathcal{C}_i^j$

## 3 Independent tasks

Since the seminal paper of Liu and Layland [17] the independence of tasks is defined such that the "[...] requests for a certain task do not depend on the initiation or the completion of requests for other tasks.". Moreover the schedulability analysis of independent tasks may be studied under the hypothesis that the tasks do not share any resources except for the processor. This paper assumes this hypothesis true and it shows that, thanks to the definition of the pWCET provided by Burns and Edgar [10], this hypothesis is the unique requirement with the respect of the independence when probabilistic real-time analyses are considered. A probabilistic real-time system is a system with at least one parameter described by a random variable. In this paper tasks have (worst-case) execution times described by random variables.

We present now two problems for task systems with (worst-case) execution times described by random variables:

- the schedulability analysis for fixed-priority scheduling<sup>3</sup> on one processor. This problem allows us to describe the probabilistic independence hypothesis usually required by such analysis and to explain why this hypothesis is not stronger than the hypothesis of independent tasks required by deterministic analyses<sup>4</sup>.
- the estimation of the pWCET PF for a task on one processor. This problem allows us to describe the statistical independence hypothesis usually required by such analysis and to explain why this hypothesis is not stronger than the hypothesis of independent tasks required by deterministic analyses.

### 3.1 Probabilistic independence

We consider  $n$  periodic tasks  $\tau_i$  described by  $(O_i, \mathcal{C}_i, D_i, T_i)$ ,  $\forall i = 1, n$  that are scheduled by a preemptive fixed-priority scheduling algorithm on one processor. The task  $\tau_i$  has a higher priority than  $\tau_j$   $\forall i < j$ . The scheduling algorithm is known and given, and proposing one is beyond the purpose of this paper. An interested reader may find a solution to the optimal fixed-priority scheduling algorithms for tasks with worst-case execution times described by random variables in [21].

The response time PF of a job  $\tau_i^j$  is calculated using the operation between two random variables called convolution (defined in Definition 5) that exists if the random variables are (probabilistically) independent as defined in Definition 4.

**Definition 4.** *Two random variables  $\mathcal{X}$  and  $\mathcal{Y}$  are (probabilistically) independent if they describe two events such that the outcome of one event does not have any impact on the outcome of the other.*

**Definition 5.** *The sum  $\mathcal{Z}$  of two (probabilistically) independent random variables  $\mathcal{X}$  and  $\mathcal{Y}$  is the convolution  $\mathcal{X} \otimes \mathcal{Y}$  where  $P\{\mathcal{Z} = z\} = \sum_{k=-\infty}^{k=+\infty} P\{\mathcal{X} = k\}P\{\mathcal{Y} = z - k\}$ .*

The calculation of the response time PF of the  $j^{\text{th}}$  job of  $\tau_i$  is  $\mathcal{R}_i^j$  is provided in [9] as follows

$$f_{\mathcal{R}_i^j} = f_{\mathcal{R}_i^j}^{[0, \lambda_{i,j}]} + (f_{\mathcal{R}_i^j}^{(\lambda_{i,j}, \infty)} \oplus f_{\mathcal{C}_i^j}), \quad (4)$$

where  $\lambda_{i,j} = O_i + jT_i$ ,  $\forall j \geq 0$  is the release time of the job  $\tau_i^j$ . A solution for Equation (4) may be obtained recursively [9].

Equation (4) may be reformulated as follows:

$$\mathcal{R}_i^j = \mathcal{B}_i(\lambda_{i,j}) \oplus \mathcal{I}_i(\lambda_{i,j}) \oplus \mathcal{C}_i^j, \quad (5)$$

where  $\mathcal{B}_i(\lambda_{i,j})$  is the accumulated *backlog* (e.g., the sum of execution times) of higher priority tasks released before  $\lambda_{i,j}$  and still active (not completed yet) at  $\lambda_{i,j}$ .  $\mathcal{I}_i(\lambda_{i,j})$  is the sum of the execution times of higher priority tasks arriving after  $\lambda_{i,j}$ .

<sup>3</sup> The fixed-priority scheduling considers that all jobs of a task conserve the same priority during the entire schedule.

<sup>4</sup> We use deterministic analyses as opposite to probabilistic.

In Equations (5) and (4) the operation  $\oplus$  indicates that a summation is needed to take into account the execution times of all higher priority tasks active before, at or after  $\lambda_{i,j}$ . If the random variables  $\mathcal{C}_i^j, \forall i$  and  $\forall j$  are (probabilistically) independent, then this operation is the convolution as defined in Definition 5. If the random variables  $\mathcal{C}_i^j, \forall i$  and  $\forall j$  are (probabilistically) dependent, then the operation  $\oplus$  needs to take into account the dependences between these random variables. One possible solution is the utilization of copulas [3], but to our best knowledge there is no extension of the results presented by Equations (5) and (4) to the case of copulas. One may note this interesting and yet open problem of probabilistic real-time systems. This open problem may be related to the study of the impact of (missing) probabilistic independence while convolving as it has been done in [25], but the missing quantification of the number of dependent random variables within this paper does not allow to conclude. Nevertheless one may note this second open problem.

**Case of pETs.** If the random variables  $\mathcal{C}_i^j$  describe pETs of the  $j^{th}$  and  $\exists j_1 \neq j_2$  such that  $\mathcal{C}_i^{j_1} \neq \mathcal{C}_i^{j_2}$ , then the random variables are not independent and Equations (5) and (4) cannot be applied with the current knowledge of the literature.

**Case of pWCETs.** If the random variables  $\mathcal{C}_i^j$  describe pWCETs of tasks  $\tau_i$ , then the random variables are independent as they are obtained as upper bounds for pETs. Thus Equations (5) and (4) can be applied by replacing the operation  $\oplus$  by  $\otimes$  and  $\mathcal{C}_i^j$  by  $\mathcal{C}_i$ . Equation (4) becomes Equation (6) and Equation (5) becomes Equation (7) as follows (with the same notations as before):

$$f_{\mathcal{R}_i^j} = f_{\mathcal{R}_i^j}^{[0, \lambda_{i,j}]} + (f_{\mathcal{R}_i^j}^{(\lambda_{i,j}, \infty)} \otimes f_{\mathcal{C}_i}) \quad (6)$$

$$\mathcal{R}_i^j = \mathcal{B}_i(\lambda_{i,j}) \otimes \mathcal{I}_i(\lambda_{i,j}) \oplus \mathcal{C}_i \quad (7)$$

In conclusion in the case of tasks with pWCETs, the utilization of the definition of pWCET as provided in [10] implies that the (probabilistic) independence of tasks is implicit and it does not require any new hypothesis for a task system with respect to the case of independent tasks described deterministically.

We present in Section 3.2 what are the hypotheses needed to obtain a pWCET for a task executed on a given platform.

### 3.2 Statistical independence

Since the seminal paper of Edgar and Burns [10], different papers [14], [28] have been presented the calculation of pWCETs based on the utilization of extreme value theory (EVT) [12]. The statistical independence required by EVT must be properly checked as underlined by Griffin and Burns [13] and a complete application of EVT is presented in [7].

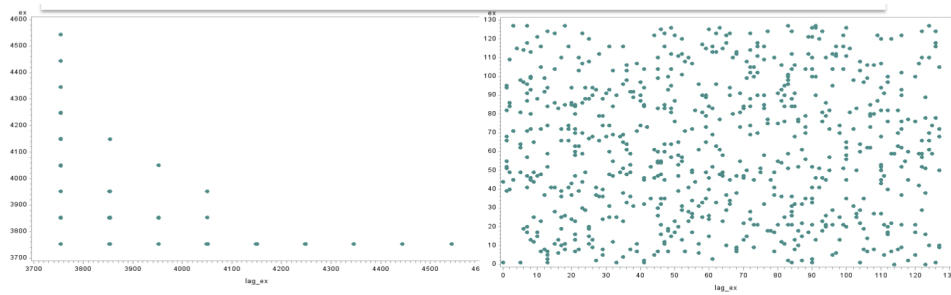
We present now the extreme value theory and the associated (statistical) independence definition. We end this section by explaining why this (statistical) independence is not a requirement on the top of independence for the tasks of a system.

**Extreme value theory** Extreme Value Theory (EVT) estimates the probability of occurrence of extreme values which are known to be rare events [10]. More precisely, EVT predicts the distribution function for the maximal values of a set of  $n$  observations, which are modelled with random variables. EVT is analogous to Central Limit Theory [27] but instead of estimating the average or the central part of a PF, EVT estimates the extremes [12].

The main result of EVT is provided in Theorem 1 where  $F$  denotes the common distribution function of  $n$  random variables describing the observations (execution times) of a task  $\tau_i$  and the pWCET  $\mathcal{C}_i = \mathcal{M}_n$ .

**Theorem 1.** [12] *Let  $\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n\}$  be a sequence of independent and identically distributed (i.i.d.) random variables. Let  $\mathcal{M}_n = \max\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n\}$ . If  $F$  is a non degenerate distribution function and there exists a sequence of pairs of real numbers  $(a_n, b_n)$  such that  $a_n \geq 0$  and  $\lim_{n \rightarrow \infty} P(\frac{\mathcal{M}_n - b_n}{a_n} \leq x) = F(x)$ , then  $F$  belongs to either the Gumbel, the Frechet or the Weibull family.*

Theorem 1 (as many statistical results) requires that the  $n$  random variables  $\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n$  are (probabilistically) independent. Here the variables  $\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n$  describe observations of the execution of task  $\tau_i$ . The independence hypothesis implies that  $\mathcal{X}_i$  is obtained from observed executions that are independent from those contained by  $\mathcal{X}_j, \forall i \neq j$ . Nevertheless in reality the user of this theory runs the task under study several times and obtains a set of data. In this case in order to fulfil the hypothesis of (probabilistic) independence required by Theorem 1, the user check the (statistical) independence by applying different independence tests on those data [27]. For instance the lag test results may indicate graphically if the data are independent (see Figure 2, right graph) or not (see Figure 2, left graph).



**Fig. 2.** Graphical results provided by lag test on two sets of observed execution times

In conclusion EVT requires that the observed executions of a task are made independently and this does not imply any extra requirement on the tasks, but on the process of producing and collecting the data. This requirement is also true for another version of extreme value theory called Peaks Over Threshold [6] that may be applied to the problem of estimating pWCETs. To our best knowledge there is no such application of

POT to the pWCET estimation and one may note this interesting and yet open problem for probabilistic real-time systems.

These independent executions of tasks may be obtained either by statistical methods [28] or by ensuring specific properties for the platform [5].

## 4 Related work

In this paper we discuss the impact of the independence property on probabilistic systems with (worst-case) execution times of tasks described by random variables. Such systems are the probabilistic systems the most studied among probabilistic real-time systems. These results may be classified in four main types: schedulability analysis, re-sampling, p(WC)ET estimation, and optimal scheduling algorithms.

**Schedulability analysis** The seminal paper of Lehoczky [16] proposes the first schedulability analysis of task systems with probabilistic execution times. This result and several improvements [29], [15] consider a specific case of PFs for the pETs. Tia et al. [26] and Gardner [11] propose probabilistic analyses for specific schedulers. Abeni et al. [1] proposes probabilistic analyses for tasks executed in isolation and a recent work consider time-evolving models [23]. The most general analysis for probabilistic systems with (worst-case) execution times of tasks described by random variables is proposed in [9]. A time-evolving model of pETs is introduced in [19] and an associated schedulability analysis on multiprocessors is presented.

**Re-sampling with respect to the p(WC)ETs** The schedulability analyses may have an important complexity directly related to the number of possible values of the random variables. This complexity may be decreased by using re-sampling techniques that ensure the safeness (the new response time PF upper bounds the result obtained without re-sampling) [24], [22].

**The p(WC)ET estimation** Since the seminal paper of Edgar and Burns [10], different papers [14], [28], [13], [7] propose statistical solutions for this problem. To our best knowledge only one paper proposes a pET estimation for a task [8].

**Optimal scheduling algorithms** To our best knowledge, there is only one paper presenting optimal fixed-priority scheduling algorithms for pWCETs of tasks described by random variables [21]. For time-evolving pETs of tasks an optimal fixed-priority algorithm is proposed for several processors [20].

## 5 Conclusion and open problems

In this paper we study the impact of defining pWCETs of tasks (concept introduced by Burns and Edgar) on the probabilistic real-time systems. This discussion is proposed from the point of view of the notion of independence. Nowadays the probabilistic real-time literature uses three distinct notions of independence: independence of tasks, probabilistic independence among random variables and statistical independence of data. We show that the two later concepts does not imply the loss of the first one. Therefore

the probabilistic real-time analyses do not have stronger requirements from the task systems than a deterministic real-time analysis as long as pWCETs are used.

Within this paper we underlined the existence of several open problems interesting for real-time systems with p(WC)ETs:

- the introduction of copulas for systems with dependent pETs;
- the impact of missing probabilistic independence while convolving pETs;
- utilization of POT theory for the pWCET estimation.

## 6 Acknowledgments

We acknowledge grant no. 249100 (PROARTIS) from the 7<sup>th</sup> Framework Programme of the European Community. The author is thankful to the members of PROARTIS and of University of York for inspiring discussions on probabilistic real-time systems.

## References

1. L. Abeni and Buttazzo. QoS guarantee using probabilistic deadlines. In *IEEE Euromicro Conference on Real-Time Systems (ECRTS99)*, 1999.
2. A.K. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *the 19th IEEE Real-Time Systems Symposium (RTSS1998)*, 1998.
3. G. Bernat and M. Newby. Probabilistic WCET analysis, an approach using copulas. *Journal of Embedded Computing*, 2007.
4. Alan Burns and Stewart Edgar. Predicting computation time for advanced processor architectures. In *the 12th Euromicro Conference on Real-time systems (ECRTS2000)*, pages 89–96, 2000.
5. Francisco J. Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. PROARTIS: Probabilistically analysable real-time systems. *ACM TECS Special Issue on Probabilistic Embedded Computing*, 2012.
6. Stuart COLES. *An introduction to statistical modeling of extreme values*. Springer, 2001.
7. L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *the 24th Euromicro Conference on Real-time Systems (ECRTS12)*, 2012.
8. Laurent David and Isabelle Puaut. Static determination of probabilistic execution times. In *ECRTS*, pages 223–230, 2004.
9. J.L. Díaz, D.F. Garcia, K. Kim, C.G. Lee, L.L. Bello, López J.M., and O. Mirabella. Stochastic analysis of periodic real-time systems. In *the 23rd IEEE Real-Time Systems Symposium (RTSS02)*, pages 289–300, 2002.
10. Stewart Edgar and Alan Burns. Statistical analysis of wcet for scheduling. In *the 22nd IEEE Real-Time Systems Symposium (RTSS01)*, pages 215–224, 2001.
11. M.K. Gardner and J.W. Lui. Analyzing stochastic fixed-priority real-time systems. In *the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 1999.
12. B.V. Gnedenko. Sur la distribution limite du terme maximum d’une serie aleatoire. *Annals of Mathematics*, 44:423–453, 1943.



13. David Griffin and Alan Burns. Realism in Statistical Analysis of Worst Case Execution Times. In *the 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, pages 44–53, 2010.
14. J. Hansen, S Hissam, and G. A. Moreno. Statistical-based wcet estimation and validation. In *the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2009.
15. Jeffery P. Hansen, John P. Lehoczky, Haifeng Zhu, and Ragunathan Rajkumar. Quantized edf scheduling in a stochastic environment. In *IPDPS*, 2002.
16. J.P. Lehoczky. Real-time queueing theory. In *the 10th IEEE Real-Time Systems Symposium (RTSS96)*, pages 186–195, 1996.
17. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
18. José María López, José Luis Díaz, Joaquín Entrialgo, and Daniel García. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems*, pages 180–207, 2008.
19. Sorin Manolache, Petru Eles, and Zebo Peng. Schedulability analysis of applications with stochastic task execution times. *ACM Trans. Embedded Comput. Syst.*, 3(4):706–735, 2004.
20. Sorin Manolache, Petru Eles, and Zebo Peng. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. *ACM Trans. Embedded Comput. Syst.*, 7(2), 2008.
21. Dorin Maxim, Olivier Buffet, Luca Santinelli, Liliana Cucu-Grosjean, and Robert I. Davis. Optimal priority assignment algorithms for probabilistic real-time systems. In *the 19th International Conference on Real-Time and Networked Systems (RTNS2011)*, pages 129–138, 2011.
22. Dorin Maxim, Mike Houston, Luca Santinelli, Guillem Bernat, Robert I. Davis, and Liliana Cucu-Grosjean. Re-sampling for statistical timing analysis of real-time systems. In *RTNS*, pages 111–120, 2012.
23. Luigi Palopoli, Luca Abeni, Daniele Fontanelli, and Nicola Manica. An analytical bound for probabilistic deadlines. In *the 24th Eumicro Conference on Real-time Systems (ECRTS2012)*, 2012.
24. Khaled S. Refaat and Pierre-Emmanuel Hladik. Efficient stochastic analysis of real-time systems via random sampling. In *the 22nd Euromicro Conference on Real-Time Systems (ECRTS 2010)*, pages 175–183, 2010.
25. Marcelo Santos, Björn Lisper, George Lima, and Veronica Lima. Sequential composition of execution time distributions by convolution. In *Proc. 4th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2011)*, pages 30–37, November 2011.
26. T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.C. Wu, and J.S Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 1995.
27. Feller W. *An introduction to Probability Theory and Its Applications*. Wiley, 1996.
28. Lu Yue, Iain Bate, Thomas Nolte, and Liliana Cucu-Grosjean. A new way about using statistical analysis of worst-case execution times. *ACM SIGBED Review*, September 2011.
29. Haifeng Zhu, Jeffery P. Hansen, John P. Lehoczky, and Ragunathan Rajkumar. Optimal partitioning for quantized edf scheduling. In *RTSS*, pages 212–222, 2002.

# Burns Standard Notation for real time scheduling

Robert I. Davis

University of York  
rob.davis@york.ac.uk

## 1 Introduction

During the last 20 years, there has been a significant growth in the number of people active in the real-time scheduling community and consequently a large increase in the number of research publications. Today, more than ever it is important that we make our latest research (whether it is a short workshop abstract like this one or a lengthy journal paper) as easy as possible for others to understand and build upon.

As readers and reviewers of papers on real-time scheduling, we have all experienced the difficulty and at times hair-pulling frustration involved in trying to decipher complex analysis embodied in numerous equations using unfamiliar, arbitrary and sometimes downright cryptic notation [1]. We have all also experienced the pleasure of reading interesting, insightful, well-structured papers with clear step-by-step analysis, that uses precise terminology, and a concise, consistent and well thought-out notation [8].

Through the volume and the quality of the research he has produced (450 publications and counting), the number of PhD students he has supervised and nurtured into independent researchers, and the number of people reading his work (approaching 15,000 citations), Alan Burns continues to have a substantial and guiding influence on the de-facto standard terminology and notation that has been adopted by many in the real-time scheduling community. We do not claim that he invented this notation, but that over the years he has been instrumental in extending and shaping it into a form suitable for continued use.

In recognition of Alan's 30th birthday<sup>1</sup> and his enduring contribution to the field of real-time scheduling, we hope that this notation, summarised below, will from hereon be referred to as ***Burns Standard Notation***. Our aim is for it to be used in future by all attending the workshop or reading this abstract, and thus become so widespread and standardised upon that in a few years it will be hard to find a new paper on real-time scheduling that does not make use of it.

## 2 Guidelines

Burns Standard Notation describes the properties of a real-time system and in particular the set  $(\tau)$  of  $n$  tasks that execute on it, where each task  $\tau_i$  is identified by a unique index  $i$  from 1 to  $n$ . This notation evolved over a number of years according to a set of informal

---

<sup>1</sup> Beyond a certain age one should think in hexadecimal and still claim to be thirty something. At the time of writing, the author had recently turned thirty.

guidelines. These guidelines provide both a rationale for the choices made and enable extension of the notation to new properties or parameters in a consistent way.

The guidelines used in creating and extending Burns Standard Notation are as follows:

- (i) Subscripts refer to a task index, for example  $D_i$ , with a second subscript (if required) referring to the index of a specific invocation or job of that task, for example  $d_{i,j}$ .
- (ii) Upper case letters are used for offline properties that are relative rather than absolute, for example  $D_i$  is the relative deadline of task  $\tau_i$ .
- (iii) Lower case letters are used for online properties that are absolute, for example  $d_{i,k}$  is the absolute deadline of the  $k$ th job of task  $\tau_i$ .
- (iv) Functions are used for properties that vary with respect to some parameter (often time or a time interval). For example  $c_i(t)$  denotes the remaining execution time of task  $\tau_i$  at time  $t$ , and  $I_i(t)$  denotes the maximum interference from task  $\tau_i$  in an interval of length  $t$ .
- (v) Superscripts are used to qualify different variants of a property, for example  $R_i^{LO}$  denotes the response time of task  $\tau_i$  when the system is in a low criticality mode.
- (vi) Sets of tasks are described as functions of the task index, for example  $lp(i)$  is the set of tasks with priorities lower than that of task  $\tau_i$ .

### 3 Burns Standard Notation

Burns Standard Notation is given in the table below, in alphabetical order. Our aim is for this notation to become the accepted standard for real-time scheduling papers.

| Notation            | Terminology                  | Meaning   |
|---------------------|------------------------------|---|
| $B_i$               | Blocking                     | The longest time for which task $\tau_i$ can be prevented from executing by tasks of lower priority.  |
| $C_i$               | Worst-case execution time    | An upper bound on the longest possible execution time of task $\tau_i$ .  |
| $c_i(t)$            |                              | Worst-case remaining execution time of task $\tau_i$ at time $t$ .  |
| $D_i$               | Relative Deadline            | The longest elapsed time that task $\tau_i$ is permitted to take from being released until it completes execution.  |
| $E(t)$              | Error recovery overhead      | The total time spent recovering from errors in a time interval of length $t$ .  |
| $F_i$               | Final non-pre-emptive region | The maximum length of the final non-pre-emptive region of task $\tau_i$ .   |
| $H$                 | Hyperperiod                  | The Least Common Multiple of all task periods.  |
| $hp(i)$<br>$hep(i)$ | Set of higher priority tasks | $hp(i)$ is the set of tasks with higher priority than task $\tau_i$ , whereas $hep(i)$ is the set of tasks with higher or equal priority to task $\tau_i$ . |

| Notation            | Terminology                 | Meaning  |
|---------------------|-----------------------------|--|
| $J_i$               | Release Jitter              | The longest possible time from the notional arrival of a job of task $\tau_i$ until it is released i.e. becomes ready to execute.  |
| $J_k$               | Job $k$                     | In the case of papers discussing systems of independent jobs, then $J_k$ is used to mean the job with index $k$ .  |
| $L_i$               | Criticality level           | In a mixed criticality system, the criticality level of task $\tau_i$ .  |
| $lp(i)$<br>$lep(i)$ | Set of lower priority tasks | $lp(i)$ is the set of tasks with lower priority than task $\tau_i$ , whereas $lep(i)$ is the set of tasks with lower or equal priority to task $\tau_i$ .  |
| $m$                 |                             | Number of processors   |
| $n$                 |                             | Number of tasks  |
| $P_i$               | Priority                    | The priority of task $\tau_i$ , used to determine which of a competing set of ready tasks should be executed. Where it is possible to do so without loss of generality, the priority of a task is often assumed to equate to its index $i$ . |
| $R_i$               | Worst-case response time    | The longest possible elapsed time from the release of any job of task $\tau_i$ until the completion of that job.   |
| $S_i$               | Slack                       | The maximum amount of additional interference that task $\tau_i$ may be subject to without missing a deadline.   |
| $s$                 | Speed                       | The speed of the processor.  |
| $T_i$               | Period                      | The minimum inter-arrival time between jobs of task $\tau_i$ .   |
| $U_i$               | Utilisation                 | The processor utilisation of task $\tau_i$ , $U_i = C_i/T_i$ . ( $U$ is the utilisation of the task set $\tau$ ).  |
| $\tau_i$            | Task                        | The task with index $i$ .  |
| $\tau$              | Task set                    |  |
| $W_i$ or $w_i$      | Window or busy period       | The length of a priority level $i$ busy period or scheduling window. Used in schedulability analysis equations where the response time is not computed directly.   |

The symbols  $C_i$ ,  $T_i$ ,  $\tau_i$  and  $U$  were used in the famous paper by Liu and Layland [6] in 1973. Many of the other symbols, including  $B_i$ ,  $D_i$ ,  $L_i$ ,  $J_i$ ,  $R_i$ , and  $hp(i)$  became established following their use in one of Alan Burns seminal contributions to the analysis of fixed priority scheduling [2]. Other symbols were introduced in subsequent papers in the mid 1990s ( $F_i$ ,  $P_i$  and superscripts [4],  $S_i$  [5],  $E(t)$  [7] ), whereas  $L_i$  [3] is a more recent addition.

## References

1. It would be unfair to pick out any one paper for criticism but I see you checked hoping it wasn't yours! A.
2. N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.
3. S.K. Baruah, A. Burns, and R.I. Davis. Response-time analysis for mixed criticality systems. In *proceedings Real-Time Systems Symposium*, pages 34–43, 29 2011-dec. 2 2011.

4. Alan Burns. Preemptive priority-based scheduling: An appropriate engineering approach. In *Advances in Real-Time Systems, chapter 10*, pages 225–248. Prentice Hall, 1994.
5. R.I. Davis, K.W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *proceedings Real-Time Systems Symposium*, pages 222 –231, 1993.
6. C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.
7. K. Tindell, A. Burns, and A. Wellings. Calculating controller area network (can) message response times. *Control Engineering Practice*, 3:1163–1169, 1995.
8. It would be unfair to pick out any one paper for praise but I see you checked this time hoping it was yours! Z.

# MAST: Bringing Response-Time Analysis into Real-Time Systems Engineering

Michael González Harbour, J. Javier Gutiérrez, Julio L. Medina, J.Carlos Palencia,  
J.María Drake, Juan M. Rivas, Patricia López Martínez, César Cuevas

Universidad de Cantabria, Spain  
{mgh, gutierjj, medinajl, palencij,  
drakejm, rivasjm, lopezpa, cuevasce}@unican.es  
<http://www.ctr.unican.es/>

**Abstract**<sup>1</sup>. This paper presents an overview of MAST, a modelling and analysis suite for real-time systems, with emphasis on the description of the underlying schedulability analysis techniques. The paper presents the MAST model for distributed real-time systems, and the tools included in the MAST suite. Response-time analysis techniques are used for schedulability analysis, and are integrated with tools that are capable of assigning scheduling parameters and performing sensitivity analysis. The MAST suite also includes design tools that allow designers to extract schedulability analysis models from their UML/MARTE design descriptions.

**Keywords:** Real-time systems, schedulability analysis, response-time analysis, tools, assignment of scheduling parameters, sensitivity analysis, design methods, MARTE

## 1 Introduction

When timing requirements must be met even in the worst case conditions, testing methods are insufficient because there is no guarantee that the worst case was tested. This is why mathematical methods, called schedulability analyses, are needed to obtain these guarantees. As inputs, the analysis needs a model of the timing behaviour of the application and its timing requirements, together with the worst-case execution times (WCETs) of the different blocks of sequential code. As a result, schedulability analysis provides the answer to the question: will a set of dynamically-scheduled concurrent tasks meet its timing requirements under all possible circumstances?

To create the timing behaviour model we take into account that real-time systems have a reactive architecture in which software reacts to timing and external workload events, executing specific tasks in response. Many of these events have a repetitive nature because tasks have to continuously react to changes in the environment. We also find that many of these events are periodic, perhaps triggered from a hardware timer, and others are aperiodic, triggered from a human operator input, a message arriving through a communications network, or a hardware interrupt generated by a sensor reading a signal from the environment. Tasks will synchronize among themselves to accomplish a coordinated result and will share resources in mutual exclusion. In

---

1. This work has been funded in part by the Spanish Government and FEDER funds under grant TIN2011-28567-C03-02 (HI-PARTES).

distributed systems, tasks will exchange messages among them using a communications network. These messages carry information and implement a control flow by which tasks in one processor may trigger the execution of tasks in other processors. All these behaviours must be captured in a model of the system that can be used as input to the schedulability analysis methods.

A simple pass/fail answer from the analysis is generally not enough for the application developer. If the system meets its timing requirements we would like to know how much space we have for growth. Similarly, if the system is not schedulable we would like to know where the timing bottlenecks are, and what parts of our system we can change to achieve schedulability. These answers can be obtained from a sensitivity analysis.

Real-time systems theory has developed a large number of scheduling policies together with their corresponding schedulability analysis techniques. Engineers trying to develop industrial real-time systems need tools that allow them to model their systems and apply these techniques. MAST (Modelling and Analysis Suite for real-Time applications) [5][17] was created at the University of Cantabria to serve both as an engineering tool and as a research platform for developing such modelling techniques and the associated analysis techniques, focusing on distributed systems. This paper describes the MAST model and the suite of tools built around it, with special emphasis on the new modelling elements introduced since the initial work published in [5]. The schedulability analysis techniques used in MAST are based on response time analysis. For distributed systems most of these techniques are based on the initial work by Tindell, Clark, Burns and Wellings [42] [43] [41], from the University of York.

The MARTE standard [25] contains the UML Profile for Modelling and Analysis of Real-Time and Embedded Systems, and can be used by real-time systems engineers to design and develop their applications. From the design model and using special-purpose tools [20] it is possible to generate the MAST model of the application and perform schedulability analysis. This process can be repeated at different stages of the development cycle in such a way that the results of the analysis can influence the design decisions. This design flow brings real-time theory directly into engineering.

The document is organized as follows. Section 2 briefly introduces the main elements of the MAST model. Section 3 lists the tools that are currently available in the MAST suite. Section 4 describes the techniques used for each of the schedulability analysis tools, while Section 5 describes the techniques used in the assignment of scheduling parameters. Finally, Section 6 gives the conclusions and discusses some future work.

## **2 The MAST Model**

MAST [5][17] defines a model to describe the timing behaviour of real-time systems designed to be analyzable via schedulability analysis techniques. MAST also provides an open-source set of tools to perform schedulability analysis and other timing analyses, including simulation, assignment of scheduling parameters, and sensitivity analysis to determine how far or close is the system from meeting its timing

requirements. The model defined in MAST is very similar to that defined in the Schedulability Analysis Modelling chapter (SAM) of the MARTE standard (The UML Profile for Modelling and Analysis of Real-Time and Embedded Systems) [25].

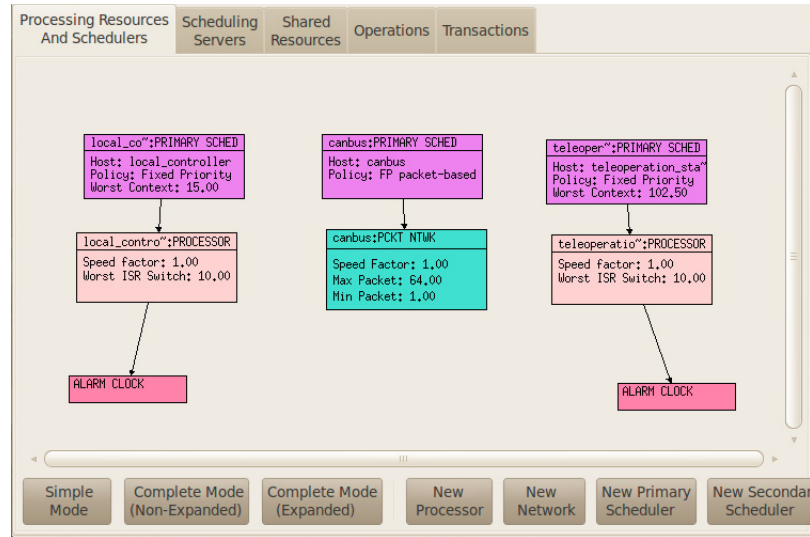
MAST is currently evolving from MAST-1 towards a new version, called MAST-2 [7], which aligns its names with those of the MARTE standard, adds new modelling elements, enhances the overhead models, and adds new scheduling policies.

We will now make a brief description of some of the main elements provided in MAST and then, as two representative examples, we will show how to use other advanced modelling elements to model effects such as tasks that self suspend and the management of maximum output jitter requirements. Finally we will list other modelling elements available in MAST.

## 2.1 Basic modelling elements

Using MAST it is easy to formulate a model of a real-time system. This process is shown here in different steps, for a simple distributed system using fixed-priority scheduling.

*Execution platform.* We first need a model of the execution platform, indicating its CPUs and networks, and the schedulers to be used for each. For instance, Figure 1 contains a simple model of distributed execution platform with a CAN bus, modelled with a packet-based network, and two processors. Both, the processors and the network, use fixed priority schedulers. This model was generated with the MAST graphical editor.



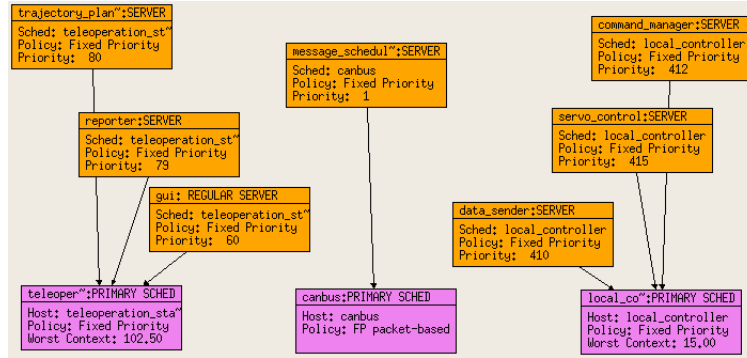
**Fig. 1** MAST model of a distributed execution platform

MAST has a rich overhead model that enables the application developer to concentrate on the modelling elements at the right abstraction level. For instance, in the execution platform we can specify the ranges of priorities being used and the



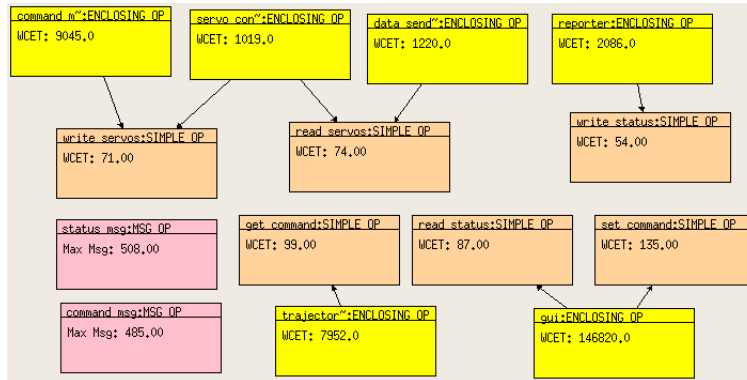
overheads of interrupt service routines, of context switches, and of protocol information being sent along with each message packet.

*Schedulable Resources*, also called *Scheduling Servers* in MAST-1. Used to model the operating system threads and the message streams, which will contain the assigned priorities (Figure 2).



**Fig. 2** Threads and message streams, modelled with MAST schedulable resources

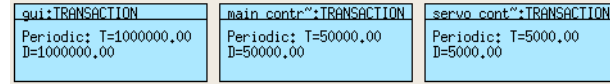
*Operations*. The next step is modelling the code blocks that will be executed by each task, and the messages sent through the network; they are both called operations in MAST, and they require specifying the WCETs or maximum message sizes, respectively (Figure 3). It is possible to model modular operations by creating composite and enclosing operations that are composed of or enclose other operations.



**Fig. 3** Operations

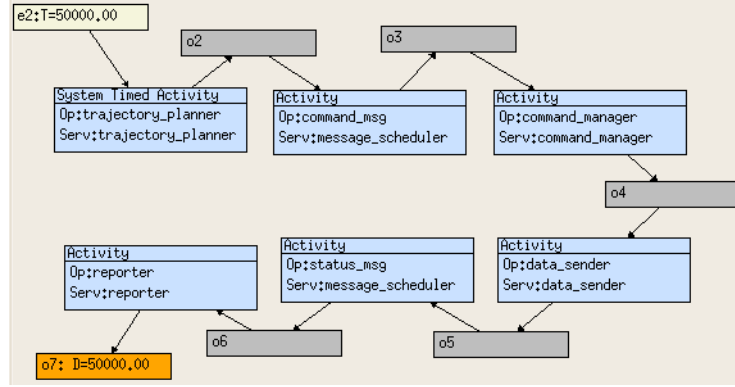
*End-to-end Flows*, abbreviated *e2e flows*, and also called *Transactions* in MAST-1. This part of the model contains the *e2e flows*, which act as the “glue” for all the other modelling elements (Figure 4). Each e2e flow has a workload external event that models the arrival pattern of the events that trigger it, and one or more *steps*, called *activities* in MAST-1. For a processor, each step specifies the thread that will be executed in response to the event instances, and the operation to be executed by that

thread at each instance. For networks, the steps describe the message stream (with scheduling information) and the particular message operation to be transmitted. We can set deadlines and other requirements at the finalization of a step.



**Fig. 4** Transactions or end-to-end flows

The model of one of the e2e flows can be seen in Figure 5 as a graph connecting a periodic workload event with the steps and the internal events.



**Fig. 5** Model of the *main\_control\_loop* end-to-end flow

*Mutual exclusion resources*, previously called *shared resources*. They model resources that are shared among different threads or tasks, and that must be used in a mutually exclusive way. They also define the synchronization protocol, which must avoid unbounded priority inversion. The supported protocols are priority inheritance [36], immediate priority ceiling [2], and the stack resource policy [2]. MAST also allows mixtures of these protocols and their use in partitioned multiprocessor systems [30][31]. For the basic priority inheritance, Rodríguez and García [35] showed that most implementations do not strictly follow the rules in [36], and that the amount of blocking may be higher than that predicted by the theory. In MAST we take this possibility into account by adding a general attribute to the platform, specifying which kind of implementation is being used.

*Timers*. They model hardware timers and define the overheads associated with their management and time resolution. Figure 1 shows two timers of the type *Alarm\_Clock*.

*Clock\_Synchronization\_Objects*: They model a clock synchronization mechanism among clocks of different processing resources.

Once the MAST model of the application is built we can use the analysis tools that will be described in Section 3. For example, by performing a sensitivity analysis on the MAST model shown above we get as an important result that the system slack is 21.88%, which means that we could increase all the execution times of the tasks and

messages by that percentage while the system would still be schedulable. We also get the individual response-time results for each task and message, shown in Figure 6.

| Global Response Times | Output Jitters | Blocking Times   | Local Response Times | Suspensions    | Local Miss Ra |
|-----------------------|----------------|------------------|----------------------|----------------|---------------|
| Transaction           | Event          | Referenced Event | Best Response        | Worst Response | Hard D        |
| servo_control         | o1             | e1               | 0.000                | 1133.00        | 5000.00       |
| main_control_loop     | o2             | e2               | 0.000                | 8442.00        |               |
| main_control_loop     | o3             | e2               | 0.000                | 18372.00       |               |
| main_control_loop     | o4             | e2               | 0.000                | 30698.00       |               |
| main_control_loop     | o5             | e2               | 0.000                | 33007.00       |               |
| main_control_loop     | o6             | e2               | 0.000                | 38087.00       |               |
| main_control_loop     | o7             | e2               | 0.000                | 40563.00       | 50000.0       |
| gui                   | o8             | e3               | 0.000                | 191408.00      | 1000000       |

Fig. 6 Results of the analysis

Once the model is built, it is trivial to get answers to questions such as: what happens if we add a new task? Would the system still be schedulable if we buy a slower cheaper CPU?

## 2.2 Self Suspension

It is common for a task to have to suspend in the middle of its execution, for example to wait for an external device to respond or modify its state. A typical example can be found when reading some information from a disk, which causes the task to suspend for some time until the data is made available to it. This time can be modelled as a wait time with a maximum and a minimum value. A model of this chain of actions is shown in Figure 7 as an end-to-end flow that includes a *delay* action.

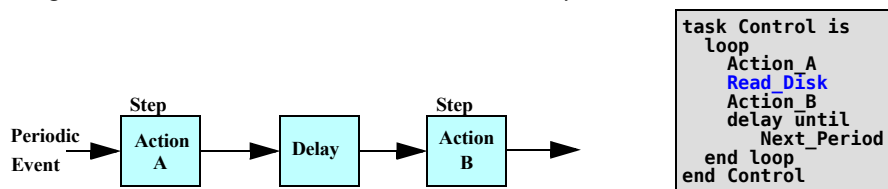


Fig. 7 Self-suspending task

This kind of end-to-end flow can be analysed with offset-based techniques by adding an offset to the second part of the task; the variability of the delay action can be modelled through additional jitter. MAST allows modelling and analysing *delay* actions, which are time intervals relative to the completion of the previous step, *Action\_A* in this example.

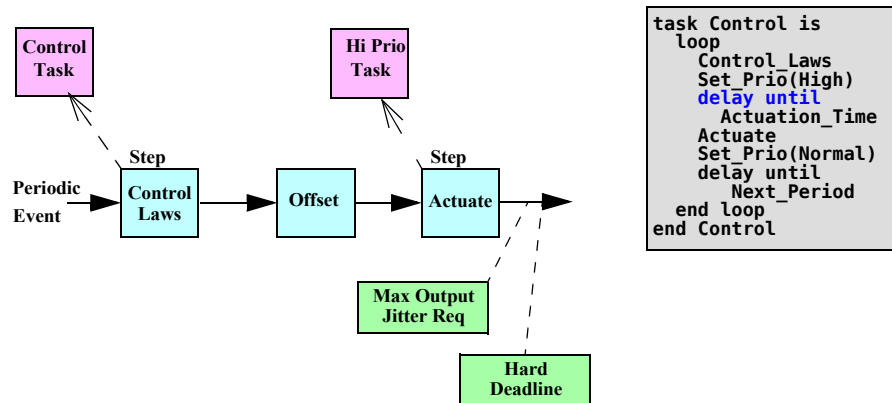
## 2.3 Handling Maximum Jitter Requirements

In some systems, especially in control applications, it is important to limit the output jitter of a particular action. For instance, in a servo controller, the application will read

the current status, execute some control law, and then perform an actuation on the servo. This actuation may need to have bounded jitter to ensure that the control laws are correct. MAST allows defining *Max\_Output\_Jitter* timing requirements to this effect.

Output jitter can be calculated as the difference between the worst- and best-case response times. But, how do we control it? We can code the application so that the action that has to be executed with bounded output jitter has, on the one hand, little input jitter, and on the other hand, limited preemption effects. To limit input jitter we can use a timer to time the action precisely. Limiting preemption requires executing the action at a high priority level.

Figure 8 shows the pseudocode of a control task in which this solution has been applied, and also shows the MAST model used for the analysis. After the first step, *Control\_Laws*, the priority of the task is set to a high level that minimizes preemption. Then, the system's timer is used with an absolute *delay* statement to time the second action precisely. The *Actuation\_Time* has to be calculated as the start of the period plus the worst-case response time of the first step, so that we ensure that when this delay expires the first step has always been completed. Once the *Actuate* step has been executed the priority is reverted to the normal level and the task waits for its next period.



**Fig. 8** Controlling output jitter with a time offset

The *offset* action included in the model is supported in MAST and is analysed using offset-based methods.

## 2.4 Other modelling elements

MAST contains a very large number of modelling elements that are defined with full details in a metamodel described with UML class diagrams [18]. These modelling elements allow describing:

- Different scheduling policies such as: preemptible and non-preemptible fixed priorities, EDF, fixed priorities with non-preemptible packets for communication networks, interrupt service routines, partitioned scheduling with time windows

repeated with a cyclic plan. For fixed priorities different aperiodic servers may be specified such as polling servers, sporadic servers [37][40], resource reservations. Hierarchical scheduling is also supported through primary and secondary schedulers.

- Different event arrival patterns such as: periodic, singular, unbounded aperiodic, sporadic, bursty.
- Different event handlers for creating multipath e2e flow graphs: step or activity, delay, offset, fork, branch, join, merge, rate divisor.
- Different requirements on the output events such as: hard and soft deadlines, local or global deadlines, maximum output jitter, maximum ratio of missed deadlines, maximum queue sizes.
- Different overhead elements with timers such as alarm clock and ticker; network drivers; context switches and interrupt switch time.

The analysis of such complex models is challenging because although there exist many published analysis techniques the interactions between the different kinds of schedulers, synchronization methods, aperiodic servers and overhead modelling elements is not readily available. In the following sections we briefly describe the structure of the MAST analysis tools. For details on some of the methods that we needed to develop to cover these interactions please see the documentation available at the MAST home page [17].

### 3 The MAST Suite of Tools

The MAST toolset includes the tools that appear in Figure 9, divided into analysis tools, data management tools, and design tools.

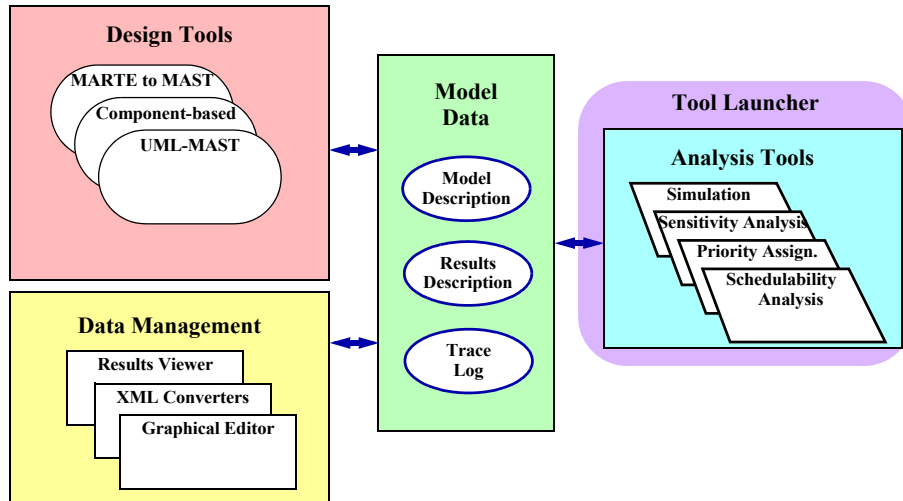
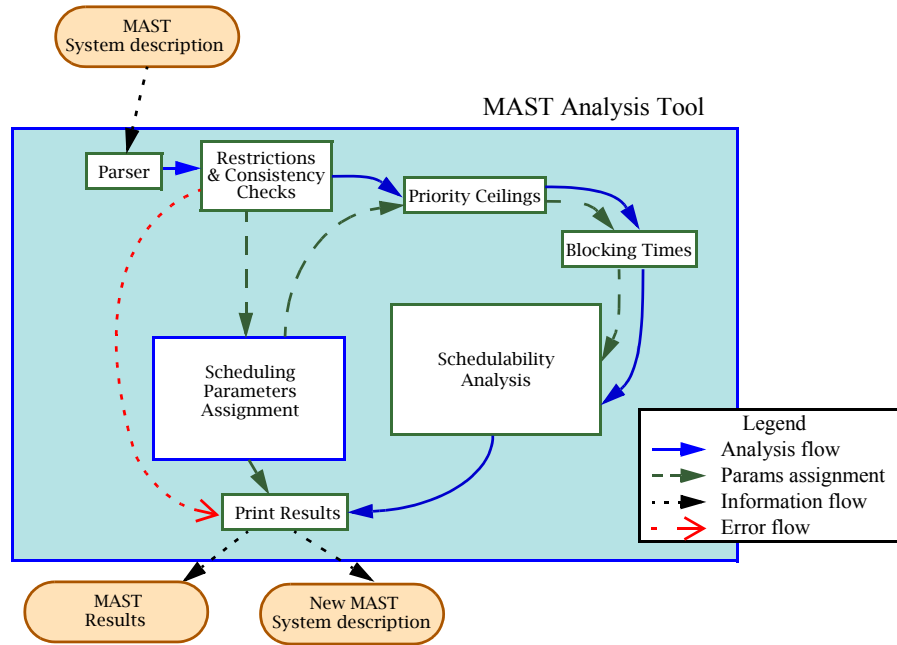


Fig. 9 MAST toolset environment



**Fig. 10** MAST Analysis tools

### 3.1 Analysis tools

*Schedulability analysis* in single-processor and distributed systems. This is the main set of tools. They check that the worst-case behaviour always meets the hard real-time requirements expressed in the model. Figure 10 represents some internal details of the MAST schedulability analysis tools. The system model is defined in a description file and a parser converts it into a data structure that is used by the tools. A module is also available to convert the data structure back into the chosen model description, to capture the results of the assignment of scheduling parameters. Different tools are provided for different kinds of systems, and also for comparison purposes.

*Automatic assignment of scheduling parameters.* These tools provide the user with capabilities to automatically calculate optimum priorities for fixed priority scheduling, scheduling deadlines for EDF scheduling. The automatic assignment uses optimum methods when available, and heuristics or optimization techniques when the optimum assignment is not available. This tool also calculates priority ceilings or preemption levels for mutual exclusion resources.

*Sensitivity analysis.* This tool can calculate slack values for the whole system, particular processing resources, end-to-end flows, or particular operations. These slack values are very useful in providing insight into which parts of the system must be modified to reach schedulability, or how much space there is for growth or for adding new actions to the system. The slack calculation tools repeat the analysis with a binary search algorithm in which execution times are successively increased or decreased.

A *discrete-event simulator* tool obtains accurate data about average results and performance features [19]. Likewise, the simulator lets us estimate observed worst and best case response parameters when the particular features of the model do not obey the restrictions of the worst-case schedulability analysis techniques. The simulation tools are able to simulate the behaviour of the system to check soft timing requirements and to generate temporal traces of the simulated execution.

### 3.2 Data management tools

A *graphical editor* can be used to generate the system description, or to view and modify a model generated elsewhere.

A *results viewer* is available to view the analysis results in a convenient way.

*XML converter.* The model and the results are specified through an ASCII description that serves as the input and output of the analysis tools. Two ASCII formats have been defined: a text special-purpose format and an XML-based format. The XML format provides the designer with capabilities to use free standard XML tools to validate, parse, analyse, and display the model files. The converter tool converts from one format to the other or back.

*Graphical editor for periodic task models.* The objective of the Periodic Task Editor is to provide an easy way of defining a MAST model for a simple task system running on a single processor. The editor allows defining periodic tasks with their execution times (WCET), periods (T), and deadlines (D). These tasks may interact by sharing mutual exclusion resources.

*Converter from MAST-1 models to MAST-2 format.* This tool is intended to ease the transition from older MAST models to the new MAST-2 specification.

### 3.3 Design tools

The MAST model of a system reflects all the necessary information to perform the verification of its timing properties by means of schedulability analysis. This model describes, for a particular analysis situation, all the independent control flows that make use of the processing capacity of the system. From a modelling perspective this approach scales up far better than a simple tasking model, thus helping in keeping in tune the conceptual paradigms used by the developers in the design with the analysis models. The distinction between platform and application fits directly in this modelling approach. MAST allows having separate models for the processing resources, the sequential functional code, and the reactive model of the application. This separation of concerns empowers reusability at many levels of detail.

The modelling capacities of MAST are suitable for being represented by means of a conceptual modelling language like the Unified Modelling Language (UML) [24]. Being UML a widely known standard for the description and design of information systems, an effort was made to enable the generation of the analysis model from the native UML design environment of the user. UML-MAST [45] was a first effort in this direction. It is a tool and a methodology [22] that provides a complete set of modelling primitives to define the analysis model in the framework of a UML tool.

The basic constructs and the way of organizing the model in this initial approach were taken into the current standard of the OMG for the Modelling and Analysis of Real-Time and Embedded Systems (MARTE) [25], in which members of the MAST team had a major participation. This UML profile provides standard extensions to the basic UML for addressing a number of concerns in the real-time and embedded systems domain. Notorious extensions include those for modelling extra-functional properties, timing constraints, generic resources and their usage, hardware and software platforms (through the modelling of operating system services), high-level real-time applications by means of tasks (called real-time units), and passive protected (shared) objects. From an analysis perspective it proposes extensions for generic quantitative analysis modelling and performance analysis and, finally, all the necessary language extensions for the construction of schedulability analysis models.

As part of a more complete model-based approach to designing real-time systems [23], a tool called `marte2mast` [20] uses UML models annotated with the schedulability analysis elements in MARTE as the input formalism to generate MAST analysis models. This tool works as a plug-in for the Eclipse integrated development environment.

Another effort was made to allow the usage of MAST in component-based approaches. A methodology [15] has been defined to provide MAST models with compositionality and reusability features. This methodology is aimed at allowing schedulability analysis of real time systems designed as assemblies of reusable modules or components. It provides capacity for building the real-time model of an application as a composition of the reusable real-time models of the modules involved in it (both software and hardware). The methodology is supported by a tool [21] that has been integrated as an Eclipse plug-in.

## 4 Schedulability Analysis Techniques in MAST

The MAST toolset contains several schedulability analysis tools capable of analysing single processor and distributed systems scheduled with fixed priority, EDF, and EDF within priorities scheduling policies. The tools are based on different scheduling analysis techniques published in the literature.

*Classic RM Analysis.* This analysis implements the classic exact response time analysis for single-processor fixed-priority systems first developed by Harter [3] and Joseph and Pandya [9], and later extended by Lehoczky to handle arbitrary deadlines [11] and by Tindell to handle jitter [42]. It corresponds to Technique 5, “Calculating response time with arbitrary deadlines and blocking”, in [10].

*Varying Priorities Analysis.* This analysis implements the response time analysis for single processor fixed priority systems in which tasks may explicitly change their priorities, developed by González, Klein and Lehoczky [4]. It corresponds to Technique 6, “Calculating response time when priorities vary”, in [10]. In terms of the MAST model, end-to-end flows for this tool are linear, i.e., composed of a sequence of steps, each representing the execution of an operation with a possibly different priority level specified through a *Permanent\_Overriden\_Priority* attribute. However, each e2e flow is limited to having a single segment. A segment is a continuous sequence of



steps executed by the same schedulable resource. Note that all the other fixed priority tools used in MAST require that the priority of a step is the same at the start and at the end, and thus they do not allow using *Permanent\_Overriden\_Priority* attributes.

*EDF Monoprocessor Analysis.* This tool implements the exact response time analysis for single-processor EDF systems first developed by Spuri [38]. In the MAST implementation we use the EDF Within Priorities tool (see below), because there may be interrupt service routines (modelled as fixed priority tasks) in addition to the EDF tasks.

*EDF Within Priorities Analysis.* This analysis is a mixture of the response time analysis for fixed priority systems [10][11][42] and for EDF [38]. It is capable of analysing systems with hierarchical schedulers, in which the underlying primary scheduler is based on fixed priorities, and there may be other EDF (secondary) schedulers scheduling tasks at a given priority level. It was developed by González and Palencia [6].

*Holistic Analysis.* This analysis extends the response time analysis to linear e2e flows in multiprocessor and distributed systems. The analysis of tasks with jitter was described by Tindell [42]. However, the problem is that in the distributed system input jitter depends on the response times, while response times depend on input jitter. There is a circular dependency that is carried over in the analysis of the different resources. An initial solution to this problem is called the holistic analysis, first developed for fixed priority systems by Tindell and Clark [42][43] and refined by Palencia et al [26]. It consists of iteratively applying the analysis in the different nodes assuming an initial estimate of jitter equal to zero, and recalculating jitter terms at the end of each analysis. By repeating this procedure iteratively the response times and jitter terms converge to the final solution, except for few pathological cases [14]. The holistic analysis is not exact because it makes the assumption that tasks of the same e2e flow are independent. For EDF systems, Spuri [39] and later Palencia [29] extended this technique for EDF systems scheduled with global deadlines (referred to a global synchronized clock), and for systems scheduled with local deadlines (referred to local clocks in each processor) [34].

*Offset Based Approximate Analysis.* This is a response time analysis for multiprocessor and distributed systems that greatly improves the pessimism of the holistic analysis by taking into account that tasks of the same end-to-end flow are not independent, through the use of offsets. Offset based analysis for fixed priorities was first introduced by Tindell [44] and then extended to distributed systems by Palencia and González [27]. It was later extended to EDF systems by Palencia and González [29]. Although it provides much better results than the holistic analysis, it is not an exact method because the exact analysis is intractable. The method approximates the interference of an e2e flow with a maximum interference function that is independent of the phase of the e2e flow. Offset-based analysis is still pessimistic but provides results that are closer to reality than the holistic analysis.

*Offset Based Approximate with Precedence Relations Analysis.* This is an enhancement of the offset based approximate analysis for fixed priority systems in which the priorities of the tasks of a given end-to-end flow are used together with the

precedence relations among those tasks to provide a tighter estimation of the response times. It was developed by Palencia et al [28], and later enhanced by Redell [32].

*Offset Based Slanted Analysis.* This is another enhancement of the offset based approximate analysis for fixed priority systems in which the maximum interference function is defined with a tighter approximation. This method provides better results than the Offset-Based Approximate Analysis, but it is uncertain whether it gets better results or not than the method with precedence relations. In general both techniques should be applied, and the best results used as an upper bound on the response times, as both methods are pessimistic. It was developed by Mäki-Turja and Nolin[16].

*Offset Based Brute Force Analysis.* This is an exact analysis of offset-based e2e flow, initially developed by Tindell [44]. It analyses all possible combinations of tasks initiating the critical instant for each e2e flow, which leads to a combinatorial problem that only offers results for very small systems. It is useful in small systems for comparison purposes.

*Analysis for heterogeneous distributed systems.* This technique [33] allows the integration of different response-time analysis techniques so that they can be applied to heterogeneous distributed systems with different scheduling policies in each resource. The Holistic and all the Offset-Based analysis techniques are now integrated in MAST into the distributed analysis for heterogeneous systems.

The capabilities of the currently implemented schedulability analysis tools are represented in Table 1. The tool in a dark-shaded cell is still under development.

**Table 1.** Schedulability analysis tools

| Technique                              | Single-Processor | Distributed /Multiple processors | Simple Transact. | Linear Transact. | Fixed priorities | EDF |
|--|------------------|----------------------------------|------------------|------------------|------------------|-----|
| Classic Rate Monotonic                 | ✓                |                                  | ✓                |                  | ✓                |     |
| Varying Priorities                     | ✓                |                                  | ✓                | ✓                | ✓                |     |
| EDF Monoprocessor                      | ✓                |                                  | ✓                |                  |                  | ✓   |
| EDF Within Priorities                  | ✓                |                                  | ✓                |                  |                  | ✓   |
| Holistic                               | ✓                | ✓                                | ✓                | ✓                | ✓                | ✓   |
| Offset Based                           | ✓                | ✓                                | ✓                | ✓                | ✓                | ✓   |
| Offset Based with Precedence relations | ✓                | ✓                                | ✓                | ✓                | ✓                |     |

## 5 Automatic assignment of scheduling parameters

The MAST toolset also contains tools to automatically assign priorities and other scheduling parameters. Priority assignment tools are provided for single-processor and

distributed systems. In single-processor systems when deadlines are within the periods the optimum deadline monotonic priority assignment developed by Leung and Whitehead is used [12]. The Liu and Layland classic rate monotonic priority assignment for the case of deadlines equal to the periods [13] is known to be a special case of the deadline monotonic assignment. When deadlines are larger than the task periods, the optimum priority assignment developed by Audsley is used [1]. This technique is based on the iterative use of the schedulability analysis tools for different priority assignment solutions, until a schedulable solution is obtained.

In multiprocessor and distributed systems the problem of assigning scheduling parameters is much harder, as there are strong interrelations between the response times in the different resources. For distributed systems scheduled with fixed priorities, we provide two heuristic solutions based on iteratively applying the schedulability analysis tools.

The first heuristic is based on the use of the simulated annealing optimization techniques, first used by Tindell, Burns, and Wellings for assigning priorities and allocating tasks to processors [41].

The second heuristic, which usually provides better and faster results is the HOPA algorithm developed by Gutiérrez and González [8]. For distributed EDF systems, the HOSDA [34] algorithm, which is an evolution of HOPA, is provided. This algorithm is capable of assigning and optimizing local and global scheduling deadlines, obtained from the end-to-end deadlines assigned to the e2e flows. In addition to HOPA and HOSDA, other simpler algorithms are provided for comparison purposes: the Proportional Deadline assignment (PD) [14], which distributes deadlines proportionally to WCETs, and the Normalized Proportional Deadline assignment (NPD) [14], which also takes into account the CPU utilization. Both algorithms can be applied for the assignment of fixed priorities or scheduling deadlines for EDF, either local or global [34]. All these techniques are integrated into HOSPA [33], the scheduling parameters assignment tool provided for heterogeneous FP/EDF distributed systems. The HOSPA algorithm needs an initial assignment of priorities or scheduling deadlines that can be provided by PD, NPD or a user-defined assignment.

The techniques for the assignment of scheduling parameters manage the concept of *preassigned scheduling parameters* (fixed priorities or local or global scheduling deadlines) to support schedulable resources whose scheduling parameters cannot be changed (for instance, legacy code that cannot be recompiled, or interrupt service routines with hardware-defined priorities). If a scheduling parameter is preassigned, it is fixed and cannot be changed by the assignment tools.

HOSPA is based on the distribution of the global deadlines of each end-to-end flow among the different steps that compose it and on the iteration over the results of the response time analysis to redistribute these deadlines. Each activity or step in the e2e flow is assigned a virtual deadline that is converted into either a priority, a local scheduling deadline, or a global scheduling deadline, depending on the scheduling policy used by the step. This conversion preserves both the order obtained by the virtual deadlines and the preassigned values of the scheduling parameters. The algorithm also takes into account that the number of steps with different virtual

deadlines could be larger than the number of available priorities for two situations: the whole priority range of a scheduler is smaller than the number of steps, or the number of steps ordered by virtual deadlines between two steps with preassigned priorities is larger than the priority range defined by the preassigned values.

In consequence, HOSPA is able to find good solutions to the problem of assigning scheduling parameters in distributed or single-processor systems, even in the presence of preassigned parameters for individual steps.

## **6 Conclusions and future work**

Real-time theory has produced a fair number of scheduling policies and their associated schedulability analysis techniques that are useful to check that a concurrent system built with state of the art operating systems and languages is able to meet its timing requirements. Bringing all this work into engineering tools that can be used to design and develop industrial real-time applications is the purpose of the MAST model and suite of tools.

Through this paper we have reviewed the main properties and elements of the MAST model for distributed real-time systems. An interesting property is the separation of concerns achieved by separating the models of the platform, the sequential functional code, and the concurrent architecture of a particular real-time analysis situation.

The MAST model enables the creation of many different tools that can be integrated into a design flow for real-time systems. The MAST tools integrate, on one end, schedulability analysis tools and design space exploration through sensitivity analysis and the automatic calculation of scheduling parameters. On the other end MAST integrates design tools using the UML and MARTE standards with the automatic generation of the analysis models.

The integration of all these tools allows the designer to perform analysis at different stages of the development life cycle, from the requirements and selection of the platform, going through the architectural design, to the detailed analysis of the final implementation.

One tool that is outside the scope of MAST but is required in this design flow is the timing analysis that can provide worst-case execution times.

As future work, we are currently working in a major enhancement called MAST-2, which will incorporate many new modelling elements including the capability to model partition-based and resource reservation scheduling policies. In addition, the implementation still has to provide support for multipath end-to-end flows and implement the missing tools, most notably support for offset-based analysis under EDF.

The integration of different scheduling policies and methods into MAST has resulted in a complex model with more than 140 modelling elements, many of them with sophisticated semantics. The different tools have lists of restrictions on the models on which they are applicable. This complexity may be a handicap for the user who tries to analyse or configure a system using a specific scheduling method. In

order to facilitate the use of MAST, we plan to develop a set profiles oriented towards specific development scenarios such as fixed-priority single-processor systems, distributed system with time partitioned scheduling, ... Users of each of these profiles could use just a subset of all the modelling elements and tools.

## References

1. N.C. Audsley, "Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times", Dept. Computer Science, University of York (1991)
2. T.P. Baker, "Stack-Based Scheduling of Realtime Processes", *Journal of Real-Time Systems*, Volume 3, Issue 1, pp. 67–99 (1991)
3. P.K. Harter, "Response times in level-structured systems". *ACM Transactions on Computer Systems*, vol. 5, no. 3, pp. 232-248 (1984)
4. M. González Harbour, M.H. Klein, and J.P. Lehoczky. "Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems". *IEEE Trans. on Software Engineering*, Vol. 20, No.1 (1994)
5. M. González Harbour, J.J. Gutiérrez, J.C. Palencia and J.M. Drake. "MAST: Modeling and Analysis Suite for Real-Time Applications". *Proceedings of the Euromicro Conference on Real-Time Systems*, Delft, The Netherlands (2001)
6. M. González Harbour and J.C. Palencia, "Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities", *Proceedings of the 24th IEEE Real-Time Systems Symposium*, Cancun, México (2003)
7. M. González Harbour, J.J. Gutiérrez, J.M. Drake, P. López Martínez, J.C. Palencia. "Modeling distributed real-time systems with MAST 2". *Journal of Systems Architecture*. In press, <http://dx.doi.org/10.1016/j.sysarc.2012.02.001>
8. J.J. Gutiérrez García and M. González Harbour, "Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems". *Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems*, Santa Barbara, California, pp. 124-132 (1995)
9. M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System". *The Computer Journal (British Computing Society)* 29, 5, pp. 390-395 (1986)
10. M.H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour. "A practitioner's Handbook for Real-Time Analysis". Kluwer Academic Pub. (1993)
11. J.P. Lehoczky. "Fixed Priority Scheduling of Periodic Tasks Sets with Arbitrary Deadlines". *IEEE Real-Time Systems Symposium* (1990)
12. J. Leung, and J. Whitehead. "On Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks". *Performance Evaluation* 2 (4), pp. 237-250 (1982)
13. C.L. Liu and J.W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". *Journal of the ACM*, Vol. 20, No. 1, pp. 46-61 (1973)
14. J. Liu, "Real-Time Systems," Prentice Hall (2000)
15. P. López Martínez, C. Cuevas and J. M. Drake. "Compositional real-time models". *Journal of Systems Architecture*. Volume 58 Issues 6–7, pp. 257-276 (2012)

16. J. Mäki-Turja and M. Nolin. "Efficient implementation of tight response-times for tasks with offsets". *Journal of Real-Time Systems*. Volume 40 Issue 1, pp. 77 - 116 (2008)
17. MAST home page: <http://mast.unican.es>
18. MAST metamodel. [http://mast.unican.es/jsimmast/Mast\\_2\\_0\\_Metamodel.pdf](http://mast.unican.es/jsimmast/Mast_2_0_Metamodel.pdf)
19. MAST simulator. <http://mast.unican.es/jsimmast/index.html>
20. MARTE to MAST. <http://mast.unican.es/umlmast/marte2mast>
21. Mod MAST. <http://mast.unican.es/modmast>
22. J.L. Medina, M. González Harbour, and J.M. Drake. "MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems". *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, London, UK, IEEE Computer Society Press, pp. 245-256 (2001)
23. J.L. Medina and A. García Cuesta. "Model-Based Analysis and Design of Real-Time Distributed Systems with Ada and the UML Profile for MARTE". *16th Int. Conf. On Reliable Software Technologies, Ada-Europe*, Edinburgh (UK), in *Lecture Notes in Computer Science*, LNCS Vol. 6652, pp. 89-102 (2011)
24. Object Management Group. *Unified Modeling Language version 2.4.1*, OMG document formal/2011-08-06 (2011)
25. Object Management Group. "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems". *MARTE specification version 1.1*, OMG Doc. formal 2011-06-02 (2011)
26. J.C. Palencia Gutiérrez, J.J. Gutiérrez García, M. González Harbour. "On the schedulability analysis for distributed hard real-time systems". *9th Euromicro Workshop on Real-Time Systems*. Toledo (1997)
27. J.C. Palencia Gutiérrez and M. González Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets". *Proceedings of the 18th. IEEE Real-Time Systems Symposium*, Madrid (1998)
28. J.C. Palencia, and M. González Harbour, "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems". *Proceedings of the 20th IEEE Real-Time Systems Symposium* (1999)
29. J.C. Palencia and M. González Harbour, "Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF". *Euromicro conference on real-time systems*, Porto, Portugal (2003)
30. R. Rajkumar, L. Sha, and J.P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors". *IEEE Real-Time Systems Symposium* (1988)
31. R. Rajkumar. "Real-Time Synchronization Protocols for Shared Memory Multiprocessors". *Proceedings of the 10th International Conference on Distributed Computing* (1990)
32. O. Redell. *Response Time Analysis for Implementation of Distributed Control Systems*, Doctoral Thesis, TRITA-MMK 2003:17, ISSN 1400-1179, ISRN KTH/MMK/R--03/17--SE (2003)

33. J.M. Rivas, J.J. Gutiérrez, J.C. Palencia, and M. González Harbour. "Schedulability Analysis and Optimization of Heterogeneous EDF and FP Distributed Real-Time Systems". ECRTS-2011, Porto (2011)
34. J.M. Rivas, J.J. Gutiérrez, J.C. Palencia, and M. González Harbour, "Optimized Deadline Assignment and Schedulability Analysis for Distributed Real-Time Systems with Local EDF Scheduling". Proceedings of the 2010 International Conference on Embedded Systems and Applications, Las Vegas, Nevada, USA, CSREA Press, pp. 150,156 (2010)
35. P. Rodríguez Hernández and J.J. García Reinoso, "Nota sobre la Implementación del mecanismo de herencia", VI Jornadas de Tiempo Real, Gijón (2003) (In Spanish)
36. L. Sha, R. Rajkumar, and J.P. Lehoczky. "Priority Inheritance Protocols: An approach to Real-Time Synchronization". IEEE Trans. on Computers (1990)
37. B. Sprunt, L. Sha, and J.P. Lehoczky. "Aperiodic Task Scheduling for Hard Real-Time Systems". The Journal of Real-Time Systems, Vol. 1, pp. 27-60 (1989)
38. M. Spuri. "Analysis of Deadline Scheduled Real-Time Systems". RR-2772, INRIA, France (1996)
39. M. Spuri. "Holistic Analysis of Deadline Scheduled Real-Time Distributed Systems". RR-2873, INRIA, France (1996)
40. M.J. Stanovich, T.P. Baker, An-I Wang, M. González Harbour. "Defects of the POSIX Sporadic Server and How to Correct Them". IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 35-45 (2010)
41. K.W. Tindell, A. Burns, and A.J. Wellings. "Allocating Real-Time Tasks. An NP-Hard Problem Made Easy". Real-Time Systems Journal, Vol. 4, No. 2, pp. 145- 166 (1992)
42. K. Tindell, "An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks". Journal of Real-Time Systems, Vol. 6, No. 2 (1994)
43. K. Tindell, and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems". Microprocessing & Microprogramming, Vol. 50, Nos.2-3, pp. 117-134 (1994)
44. K. Tindell, "Adding Time-Offsets to Schedulability Analysis", Technical Report YCS 221, Dept. of Computer Science, University of York, England (1994)
45. UML-MAST project: <http://mast.unican.es/umlmast/>

# Resource Reservation for Mixed Criticality Systems

Giuseppe Lipari<sup>1\*</sup>, Giorgio C. Buttazzo<sup>2</sup>

<sup>1</sup> LSV, ENS - Cachan, France

<sup>2</sup> Scuola Superiore Sant'Anna, Italy

**Abstract.** This paper presents a reservation-based approach to schedule mixed criticality systems in a way that guarantees the schedulability of high-criticality tasks independently of the behaviour of low-criticality tasks. Two key ideas are presented: first, to reduce the system uncertainty and advance the time at which a high-criticality task reveals its actual execution time, the initial portion of its code is handled by a dedicated server with a bandwidth reserved for the worst-case, but with a shorter deadline; second, to avoid the pessimism related to off-line budget allocation, an efficient reclaiming mechanism, namely the GRUB algorithm [6], is used to exploit the budget left by high-criticality tasks in favor of those low-criticality tasks that can still complete within their deadline.

## 1 Introduction

With the progress of computer architectures, embedded computing systems are required to execute more and more concurrent activities on the same hardware platform. In mission-critical systems, computational activities may have different levels of criticality, and therefore different guarantee requirements imposed by certification authorities. In particular, more critical tasks are required to have more conservative estimations for their computational requirements, with respect to less critical activities. Such more conservative estimations increase system predictability by over allocating computational resources to more critical tasks, but also decrease the overall efficiency.

To partially compensate for such pessimistic estimations, Vestal [16] proposed a new task model where each task can be specified with different levels of criticality, each characterised by a different computation time estimate, depending on the criticality level: the higher the criticality level, the higher the computation time estimate. Slightly different models have been also proposed by other authors.

In this paper, we consider a system with two criticality levels that must execute a task set  $\Gamma$  of  $n$  periodic or sporadic tasks. Each task  $\tau_i$  is characterised

---

\* The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 246556.



by a criticality level  $X_i$ , which can be either high ( $HI$ ) or low ( $LO$ ), a worst-case computation time (WCET)  $C_i$  (which depends on its criticality level), a period (or minimum interarrival time)  $T_i$ , and a relative deadline  $D_i$ . Tasks with low criticality, denoted as  $LO$ -tasks ( $\tau_i^{LO}$ ), have a single WCET estimate  $C_i$ , whereas tasks with high criticality, denoted as  $HI$ -tasks ( $\tau_i^{HI}$ ), have a normal WCET estimate  $C_i$  and a more conservative one,  $C_i^{ov}$ , to take overruns into account, where  $C_i^{ov} > C_i$ . Each task generates an infinite sequence of jobs,  $\tau_{i,1}, \tau_{i,2}, \dots$ , where each job  $\tau_{i,j}$  is characterised by a release time  $r_{i,j}$ , a computation time  $c_{i,j}$ , and an absolute deadline  $d_{i,j}$ . The actual computation time requested by a job  $\tau_{i,j}$  is denoted by  $e_{i,j}$ .

According to such a model, the task set  $\Gamma$  is partitioned in two subsets  $\Gamma^{LO}$  and  $\Gamma^{HI}$  and the mixed criticality (MC) feasibility problem is formulated as follows:

**Definition 1.** *A task set  $\Gamma$  is MC-feasible if and only if both the following conditions are verified:*

1. *If all  $HI$ -tasks execute for no more than their optimistic computation time  $C_i$ , then there exists a schedule where all tasks in  $\Gamma$  complete within their deadline.*
2. *If one or more  $HI$ -tasks exceed their optimistic computation time  $C_i$  (but not their conservative estimate  $C_i^{ov}$ ), then there exists a schedule where all tasks in  $\Gamma^{HI}$  complete within their deadline.*

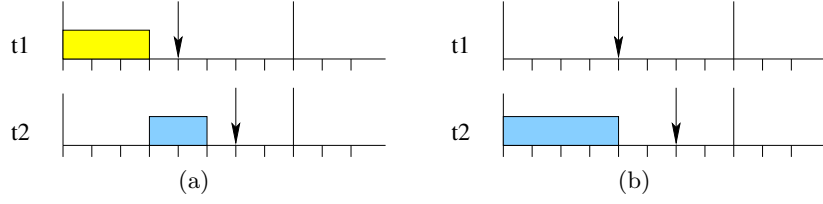
The problem of optimally scheduling such mixed-criticality systems has been shown to be highly intractable even under very simple system models [2]. The complexity comes from the fact that optimal scheduling decisions depend on the knowledge of the actual tasks execution times, which are not known off-line, but will be available only when a task completes or exceeds its optimistic estimate. Given such a dependency of scheduling decisions from future knowledge, it is clear that optimality can only be achieved by an ideal clairvoyant scheduler. To better clarify this issue, consider the task set reported in Table 1.

|          | $X_i$ | $C_i$ | $C_i^{ov}$ | $T_i$ | $D_i$ |
|----------|-------|-------|------------|-------|-------|
| $\tau_1$ | LO    | 3     | 3          | 8     | 4     |
| $\tau_2$ | HI    | 2     | 4          | 8     | 6     |

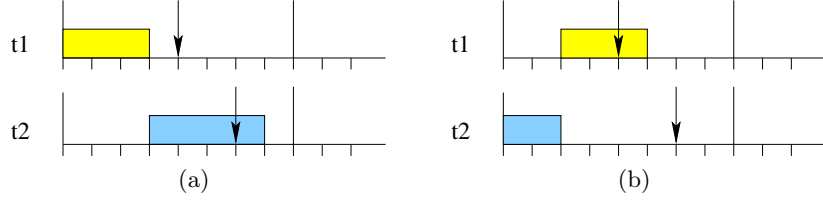
**Table 1.** Sample mixed criticality task set.

As illustrated in Figure 1, the task set is MC-feasible, since both conditions stated in Definition 1 are verified.

Nevertheless, Figure 2 illustrates that no online algorithm can guarantee the MC-schedulability of the task set, because for each decision taken at time  $t = 0$  (to schedule  $\tau_1$  or  $\tau_2$ ), there exists a situation in which a task misses its deadline. This example shows that the correct decision that produces an MC-feasible schedule can be taken only by a clairvoyant scheduler that knows (at time  $t = 0$ ) how much task  $\tau_2$  will execute.



**Fig. 1.** If  $\tau_2$  does not exceed  $C_1$ ,  $\Gamma$  is feasible (a), and if  $\tau_2$  executes for  $C_2^{ov}$ , then  $\Gamma^{HI}$  is feasible.



**Fig. 2.** If  $\tau_1$  is scheduled at  $t = 0$ ,  $\tau_2$  can miss its deadline (a); and if  $\tau_2$  is scheduled at  $t = 0$ ,  $\tau_1$  will miss its deadline (b).

*Contribution* In this paper we propose a reservation-based approach to schedule mixed criticality systems in a way that guarantees the schedulability of *HI*-tasks independently of the behavior of *LO*-tasks. Two key ideas are presented: first, to reduce the system uncertainty and advance the time at which a *HI*-task reveals its actual execution time, the initial portion of its code is handled by a dedicated server with a bandwidth reserved for the worst-case, but with a shorter deadline; second, to avoid the pessimism related to off-line budget allocation, an efficient reclaiming mechanism, namely the GRUB algorithm [6], is used to exploit the budget left by *HI*-tasks in favor of those *LO*-tasks that can still complete within their deadline.

*Paper structure* The rest of the paper is organized as follows. Section 2 formally presents the general approach. Section 3 presents the details of the reservation server. Section 4 illustrates the simulation results obtained with the proposed approach. Section 5 presents some related work. Section 6 concludes the paper and presents some future work.

## 2 General approach

We consider a reservation-based real-time system where each task (or group of tasks) can be assigned a reservation server that allocates a budget  $Q_i$  every period  $P_i$ . To better exploit the available computational resources, we assume that all the servers are scheduled by the Earliest Deadline First (EDF) scheduling algorithm [11].

Since *HI*-tasks must be guaranteed under all operating conditions and we cannot know in advance how much they will execute, each *HI*-task  $\tau_i^{HI}$  is assigned a dedicated reservation server (*HI*-server) with bandwidth  $\alpha_i^{HI}$  sufficient to satisfy its more conservative execution time  $C_i^{ov}$ . Therefore, each *HI*-task is handled by a periodic reservation  $(Q_i, P_i)$ , where  $Q_i = C_i^{ov}$  and  $P_i = T_i$ , thus having a bandwidth

$$\alpha_i^{HI} = \frac{C_i^{ov}}{T_i}. \quad (1)$$

The bandwidth remaining for serving all the *LO*-tasks is then:

$$\alpha^{LO} = 1 - \sum_{\tau_i \in \Gamma^{HI}} \alpha_i^{HI}. \quad (2)$$

Let  $U^{LO}$  be the total bandwidth required by the *LO*-tasks, that is,

$$U^{LO} = \sum_{\tau_i \in \Gamma^{LO}} \frac{C_i}{T_i}. \quad (3)$$

Note that, if  $U^{LO} \leq \alpha^{LO}$ , then there is enough bandwidth to complete all *LO*-tasks within their deadline, even when the system runs in high-criticality mode, hence the problem is trivially solved. In this paper, we consider the more interesting case where  $U^{LO} > \alpha^{LO}$ , so that not all *LO*-tasks can be guaranteed to complete before their deadlines in all modes. Nevertheless, when the system runs in low-criticality mode, that is, when all the *HI*-tasks do not exceed their optimistic estimate  $C_i$ , we propose to use a reclaiming mechanism for distributing the spare bandwidth saved by *HI*-task to *LO*-tasks. In particular, whenever a *HI*-job  $\tau_{i,j}^{HI}$  completes before its optimistic estimate ( $e_{i,j} < C_i$ ), the following bandwidth can be reclaimed:

$$U_{i,j}^{rec} = \frac{C_i^{ov} - e_{i,j}}{T_i}. \quad (4)$$

When the system switches to high-criticality mode, there are two ways of dealing with *LO*-tasks: they may be dropped, as proposed in most of the previous research on mixed-criticality scheduling, or they can continue executing as soft real-time tasks, without interfering with the *HI*-tasks. In this paper, we adopt this second approach.

To schedule *LO*-tasks, we can follow two alternative approaches:

1. All *LO*-tasks are assigned a single reservation server (*LO*-server) with bandwidth  $\alpha^{LO}$ .
2. Each *LO*-task is assigned a different server such that the sum of the bandwidths of these servers does not exceed  $\alpha^{LO}$ .

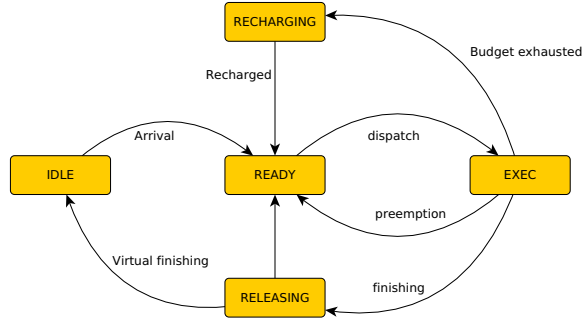
In this paper, these two approaches are compared to see whether any one dominates the other.

### 3 Server mechanism

As discussed in the previous section, we assign each *HI*-task a dedicated server with bandwidth  $\alpha_i^{HI}$  sufficient to cover the largest computational requirements  $C_i^{ov}$  of the task. At the same time, as soon as each job of a *HI*-task  $\tau_i$  completes, the remaining budget is reclaimed and assigned to the *LO*-tasks. To be able to reclaim such an extra budget in advance, we are interested in advancing the completion time of *HI*-tasks as soon as possible. To achieve this goal, we use a technique similar to the EDF-VD algorithm, proposed in [4,3]. This technique consists in using two different deadlines and budgets for a *HI*-task, depending on whether it completes before or after its normal worst-case execution time  $C_i$ . The following section describes how to modify the GRUB server to achieve this objective.

#### 3.1 High-criticality servers

A *HI*-server for a *HI*-task  $\tau_i$  is defined as a tuple  $(Q_i, Q_i^{ov}, P_i)$ , where  $Q_i = C_i$ ,  $Q_i^{ov} = C_i^{ov}$ , and  $P_i = T_i$ . The server is assigned a bandwidth  $\alpha_i^{HI} = Q_i^{ov}/P_i$  sufficient to guarantee the more conservative execution time. At run time, the server manages a capacity  $q_i$ , a *virtual time*  $v_i$ , a scheduling deadline  $d_i$ , and a criticality mode  $\gamma_i$ . Moreover, the server has an internal *state* which can be IDLE, READY, EXECUTING, RECHARGING and RELEASING. The server is initially in the IDLE state and its criticality mode is  $\gamma_i = LO$ . The state diagram for the server is shown in Figure 3.



**Fig. 3.** State diagram for the *HI*-Server.

A server that is not IDLE is said to be *active*. Let  $\mathcal{A}$  be the set of active servers. The algorithm maintains a global variable

$$U^{act} = \sum_{S_i \in \mathcal{A}} \alpha_i^{HI}. \quad (5)$$

The server uses the following rules:

1. When a *HI*-task is activated at time  $t$ , the server moves from the IDLE to the READY state. Correspondingly, its budget is replenished at  $q_i = Q_i$  and its deadline is set at:

$$d_i = t + \frac{Q_i}{Q_i^{ov}} P_i = t + \frac{Q_i}{\alpha_i^{HI}} = t + \frac{q_i}{\alpha_i^{HI}}.$$

Note that such a deadline is shorter than the usual server deadline ( $t + P_i$ ). Moreover, the capacity ( $Q_i = C_i$ ) is also less than the maximum one ( $Q_i^{ov} = C_i^{ov}$ ). However, the server bandwidth is still  $\alpha_i^{HI}$ .

Also, the server moves to the active set  $\mathcal{A}$ , therefore the value of  $U^{act}$  is updated to  $U^{act} + \alpha_i^{HI}$ .

2. When in READY, the *HI*-server with the earliest scheduling deadline is executed and moves to EXECUTING.
3. When in EXECUTING, the server capacity is decreased as:

$$dq_i = -U^{act} dt$$

If the system is fully utilised and all servers are active ( $U^{act} = 1$ ), this translates in reducing the capacity of the server at unit rate. When the system is not fully utilized, or when some server is IDLE, the capacity is decreased as a lower rate, so that the server can actually *reclaim* the free system bandwidth.

4. If, while in EXECUTING, the server is preempted by another server with earlier deadline, it moves back to READY, and its capacity is not decremented anymore.
5. If, while in EXECUTING, the capacity is exhausted, then the server behaves according to the values of the criticality mode:
  - (a) If the criticality mode  $\gamma_i = LO$ , then the capacity is immediately recharged to  $q_i = Q_i^{ov} - Q_i$ , and the deadline is postponed to

$$d_i \leftarrow d_i + \frac{Q_i^{ov} - Q_i}{\alpha_i^{HI}} = d_i + \frac{q_i}{\alpha_i^{HI}}.$$

(Notice that this corresponds to the deadline of the original *HI*-task).

Also, the criticality level is raised to  $\gamma_i \leftarrow HI$ .

- (b) If the criticality mode is  $\gamma_i = HI$ , a system exception is raised, as a *HI*-task should never exceed its budget  $Q_i^{ov}$ .
6. If, while in EXECUTING, the task completes the execution of the current job, the server moves to the RELEASING state. Correspondingly, the *virtual time* is computed for the server as follows:

$$v_i \leftarrow d_i - \frac{q_i}{\alpha_i^{HI}}$$

7. The server remains in state RELEASING until  $t \geq v_i$ . At that point, the server moves to IDLE state and the criticality level is set to  $\gamma_i \leftarrow LO$ . If  $t > v_i$ , then an extra capacity of  $\alpha_i^{HI}(v_i - t)$  is *donated* to the first server in the ready queue. Also, the server is removed from the set of active servers  $\mathcal{A}$ , and the overall utilization is decreased to  $U^{act} \leftarrow U^{act} - \alpha_i^{HI}$ .

### 3.2 Low-criticality servers

A similar algorithm is used for implementing the *LO*-server for serving *LO*-tasks, with a few modifications to the rules. A *LO*-Server is defined by only two parameters, the budget  $Q_i$  and the period  $P_i$ . As stated in Section 2, the *LO*-server is assigned a bandwidth  $\alpha_i^{LO}$  given by Equation (2). At run-time, the server manages a capacity  $q_i$ , a scheduling deadline  $d_i$ , and has a state. The state diagram for a *LO*-server is the same as a *HI*-server, and is shown in Figure 3. The following rules change:

- 1 If the server is IDLE, when one of the *LO*-tasks served by the server is activated at time  $t$ , the server moves from IDLE to the READY state. Correspondingly, its budget is replenished at the value  $q_i = Q_i$  and the server deadline is set at:

$$d_i = t + P_i = t + \frac{Q_i}{\alpha_i^{LO}} = t + \frac{q_i}{\alpha_i^{LO}}.$$

Notice that this is the same equation as the *HI*-server. Also, the server moves to the active set  $\mathcal{A}$ , therefore the value of  $U^{act}$  is updated to  $U^{act} \leftarrow U^{act} + \alpha_i^{LO}$ .

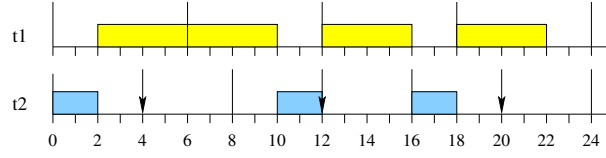
- 5 If, while in EXECUTING, the capacity is exhausted, then the server moves to the RECHARGING state and a recharging time is set at  $d_i$ . The server is suspended from execution until the capacity replenishment.
- 6 If, while in EXECUTING, the task completes the execution of the current job, and there are no more tasks in the server local queue, the server moves to the RELEASING state. Correspondingly, the *virtual time* is computed for the server as follows:

$$v_i \leftarrow d_i - \frac{q_i}{\alpha_i^{LO}}.$$

- 8 If, while the server is in RECHARGING state,  $t = d_i$ , then the server budget is replenished at  $q_i \leftarrow Q_i$ , the server deadline is postponed at  $d_i \leftarrow d_i + P_i$ , and the server is moved to the READY state.
- 9 If, while the server is in RELEASING state, a new *LO*-task is activated locally in the server, then the server moves to the READY state, with the same capacity and deadline.

### 3.3 An example

Consider a simple MC task set consisting of only two tasks: a *LO*-task  $\tau_1 = (C_1 = 4, T_1 = 6)$  and a *HI*-task  $\tau_2^{HI} = (C_2 = 2, C_2^{ov} = 4, T_2 = 8)$ . To schedule this task set, we start by defining a *HI*-server  $S_2 = (Q_2 = 2, Q_2^{ov} = 4, P_2 = 8)$ ; the server bandwidth is  $\alpha_2^{HI} = 0.5$ . The first question is: how much budget we can reserve for  $\tau_1$ ? Clearly, we cannot reserve  $(Q_1 = 4, P_1 = 6)$ , as there is not enough bandwidth left. Therefore, the *LO*-server is defined with a bandwidth  $\alpha_1^{LO} = 1 - \alpha_2^{HI} = 0.5$ , a period  $P_1 = T_1 = 6$ , and a budget  $Q_1 = \alpha_1^{LO} P_1 = 3$ . The schedule produced by the proposed algorithm is shown in Figure 4.



**Fig. 4.** Example of reclamation.

- At time 0, task  $\tau_2$  starts executing inside its server. Both servers are active (because they both arrive at time  $t = 0$ ), so the capacity is updated at the following rate while it executes:

$$\frac{dq_2}{dt} = -U^{act} = -1$$

Therefore, when task  $\tau_2$  completes, the server has budget  $q_2 = 0$ . The server moves to state **RELEASING** and virtual time is computed as  $v_2 = 4$ . Therefore, the server remains active until time  $t = 4$ , and from then it becomes inactive until time  $t = 8$ .

- At time  $t = 2$ , task  $\tau_1$  starts executing. Its budget is initially  $q_1 = 3$ . Again, all servers are active (remember that Server  $S_2$  will become inactive at time 4), hence capacity is decreased at unit rate.
- At time  $t = 4$ , the server  $S_2$  becomes **IDLE**. Therefore, the capacity rate for task  $\tau_1$  changes. First of all, let us observe that at time  $t = 4$  its value is  $q_1 = 1$ . The new rate is:

$$\frac{dq_1}{dt} = -U^{act} = -0.5.$$

This means that, at the current rate, the task can still execute for 2 units of time. That is exactly what we need to complete the task at time  $t = 6$ .

- At time  $t = 6$ , the first instance of task  $\tau_1$  completes, but the second one is activated. Therefore, the server recharges its budget at  $q_1 = 3$ , and continues to execute with deadline at  $d_1 = 12$ . Since the other server is still inactive, the rates for the virtual time and the deadline do not change.
- At time  $t = 8$ , task  $\tau_2$  is activated again and  $S_2$  becomes active. The current value of the capacity is  $q_1 = 2$ . Suppose the scheduler decides to continue executing  $\tau_1$ . Therefore, the new rates is now:

$$\frac{dq_1}{dt} = -1$$

And this means that  $\tau_1$  can execute for two more units of time, completing its executing at time  $t = 10$ .

In this example we have seen that  $\tau_1$  is able to complete execution of all its jobs within its deadline, even if the assigned budget is less than its execution requirements, thanks to the reclaiming mechanism.

However, it is still to be understood how much capacity can be reclaimed by *LO*-tasks. In fact, it is easy to build an example in which the *LO*-server does not receive enough extra capacity to complete all its tasks before their deadlines. For example, if the *LO*-tasks have very short relative deadlines compared to the *HI*-tasks, they will be executed before them most of the times, and hence they will not be able to reclaim any capacity.

Computing the amount of capacity reclaimed by *LO*-tasks is a difficult and open problem that will be the subject of our future research. In this paper we just compare two methods for scheduling *LO*-tasks inside a *LO*-server: using a single server for serving all the *LO*-tasks; or using a dedicated *LO*-server for each *LO*-task.

## 4 Experimental results

To evaluate the performance of the reclamation algorithm presented in Section 3, we performed a set of simulation experiments. The server algorithms have been implemented in RTSim [12], a scheduling simulation tool for modeling real-time systems.

In each simulation run, we generated 4 *HI*-tasks and 4 *LO*-tasks. Computation times and the periods of the *HI*-tasks have been randomly selected to get a cumulative utilization equal to 50%:

$$\sum_{\tau_i \in \Gamma^{HI}} \frac{C_i^{ov}}{T_i} = 0.5.$$

In the first set of experiments, the periods of the *HI*-tasks have been chosen according to a uniform distribution in the range [1000, 5000], in multiples of 100. In the second set of experiments, they were chosen in the interval [6000, 10000].

Computation times in low-criticality mode were computed as a fixed fraction of the computation times in high-criticality mode:

$$\forall \tau_i \in \Gamma^{HI} \quad C_i = r \cdot C_i^{ov}$$

where  $r$  was varied between [0.2, 0.75].

Computation times and periods of the *LO*-tasks were chosen to achieve a cumulative utilization equal to  $\frac{0.5}{0.75}$ . In this way, for all values of parameter  $r$ , we have:

$$\sum_{\tau_i^{HI}} \frac{C_i}{T_i} + \sum_{\tau_i^{LO}} \frac{C_i}{T_i} \leq 1.$$

In other words, we made sure that the system is never overloaded in low-criticality mode. In the first set of experiments, the periods of the *LO*-tasks were chosen in the range [6000, 10000], whereas in the second set of experiments periods were chosen in [1000, 5000].

For each *HI*-task we prepared a *HI*-server with bandwidth  $\alpha_i^{HI} = C_i^{ov}/T_i$ . For the *LO*-tasks, we made two different choices: assigning all *LO*-tasks to a



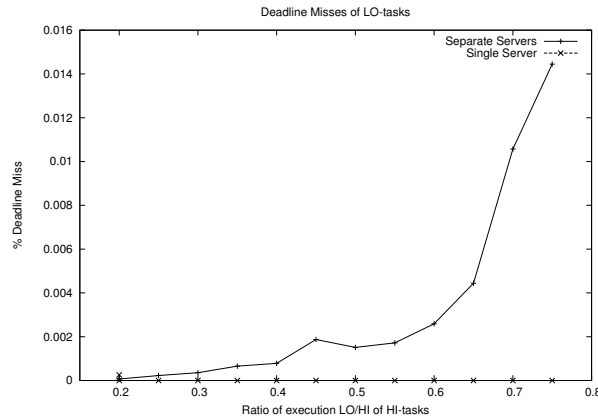
single *LO*-server with utilization equal to 50% and period  $P = 100$ ; or assign each *LO*-task to a different *LO*-server with bandwidth proportional to its computational requirements, scaled down so that the sum of the bandwidth assigned to the *LO*-server was 50%. In the single server case, *LO*-tasks inside the server were scheduled by the EDF local scheduler.

For each combination of parameters, we generated 30 different task sets with random periods and computation times. Each simulation was run for 10,000,000 units of simulation time. We only analysed the low-criticality mode, to test the effectiveness of the reclaiming algorithm in that condition. Therefore, *HI*-tasks never execute more than their optimistic computation time  $C_i$ . We measured the average number of deadlines missed and the tardiness of the *LO*-tasks. Simulation results are reported in the following sections.

#### 4.1 First set of experiments

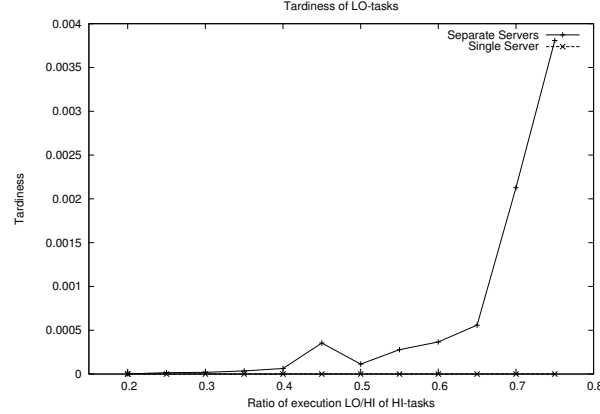
In this first set of experiments, the periods of the *HI*-tasks have been chosen to be all lower than the periods of the *LO*-tasks. Therefore, at least at the beginning of the schedule, *LO*-tasks execute after *HI*-tasks have been completed, and so they can immediately take advantage of the reclaimed bandwidth.

The deadline miss percentage of the *LO*-tasks is shown in Figure 5 for the case of separate servers, and single server. It is evident that the deadline miss percentage is very low in all cases. Even for the case of  $r = 0.75$  (where the total system utilization is very close to 100%), it never goes above 5% of the total number of deadlines. Also notice that putting all *LO*-tasks in a single server is very effective, as we observed 0 deadlines missed in all the experiments.



**Fig. 5.** Deadline miss percentage of *LO*-tasks, using dedicated servers, or a single cumulative server. The *LO*-tasks have larger periods than the *HI*-tasks.

A very similar trend can be observed for the tardiness reported in Figure 6. Such small values of the tardiness indicate that the reclaiming mechanism is very effective, even with very highly loaded systems.



**Fig. 6.** Tardiness of *LO*-tasks, using dedicated servers or a single cumulative server. The *LO*-tasks have larger periods than the *HI*-tasks.

## 4.2 Second set of experiments

In this second set of experiments, the periods of the *HI*-tasks have been chosen to be all higher than the periods of the *LO*-tasks. Therefore, at least at the beginning of the schedule, *LO*-tasks execute before *HI*-tasks, so they cannot immediately take advantage of the reclaimed bandwidth.

The deadline miss percentage of the *LO*-tasks is shown in Figure 7 for the case of separate servers and single server.

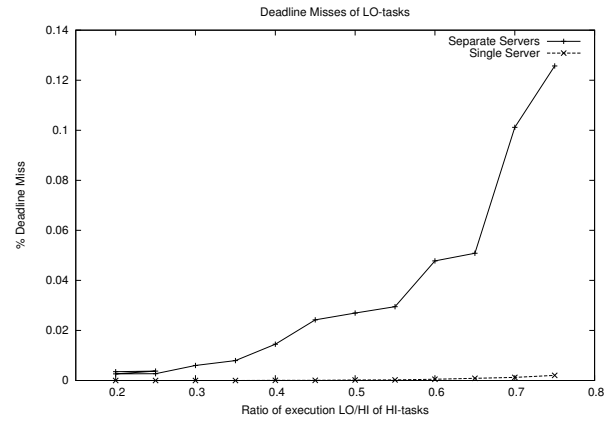
Once again, a very similar trend can be observed for the tardiness in Figure 8. Notice that in this case we detected very small values of the tardiness for  $r \geq 0.6$  due to a very small number of deadline misses (not visible in the graphs).

While these simulations cannot conclusively establish the theoretical performance guarantees for *LO*-tasks, they indicate that it is indeed worthwhile to perform further investigation on reclamation techniques for mixed criticality systems.

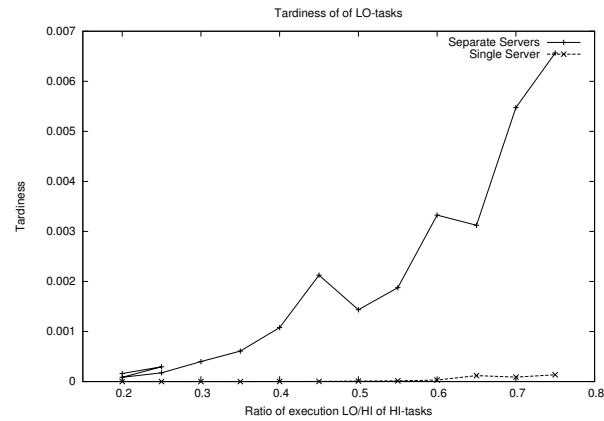
## 5 Related work

The problem of scheduling mixed criticality systems has been addressed by several authors under slightly different models and assumptions.

Lakshmanan et al. [5,8,9] proposed a slack-aware approach on top of fixed priority scheduling, providing a schedulability test that guarantees that all deadlines



**Fig. 7.** Deadline miss percentage of *LO*-tasks, using dedicated servers, or a single cumulative server. The *LO*-tasks have smaller periods than the *HI*-tasks.



**Fig. 8.** Tardiness of *LO*-tasks, using dedicated servers or a single cumulative server. The *LO*-tasks have smaller periods than the *HI*-tasks.

of *HI*-tasks are met regardless of the runtime behavior of *LO*-tasks, provided the execution of at most one *HI*-task overruns its lower WCET estimate.

Baruah et. al. [1] proposed an effective algorithm, called OCBP (Own Criticality Based Priority), to schedule a set of non-recurrent jobs. Although OCBP is able to achieve the highest speedup factor among all the fixed-job-priority algorithms, it cannot be applied to recurrent tasks.

Li and Baruah [10] proposed an algorithm, referred to as LB, to extend OCBP to sporadic tasks, but it relies on very pessimistic schedulability tests based on load bound conditions. Moreover, it introduces large run-time overhead as it needs on-line pseudo-polynomial priority assignment recomputation. To overcome the limitations of LB, Guan et al. [7] presented a new algorithm, referred to as PLRS (Priority List Reuse Scheduling) to schedule certifiable mixed-criticality sporadic task systems.

Pellizzoni et al. [13] proposed a reservations-based approach to ensure strong isolation among subsystems of different criticality. Petters et. al. [14] also considered the use of temporal isolation of subsystems for mixed-criticality systems, and addressed many practical issues in building such systems in reality. In general, the drawback of the resource/temporal isolation approach is that it relies on severely over-provisioning computing resources, which may result in significant cost and energy waste.

Baruah et al. proposed the EDF-VD algorithm [3,4], which is similar to the server mechanism proposed in this paper. In particular, they propose to anticipate the deadlines of *HI*-tasks so that when executing in *LO*-criticality mode their completion is anticipated. Their formula for computing the anticipated deadline is global, i.e. it relies on the global utilisation of *HI*-tasks, whereas in this paper we propose a different formula that accounts for the bandwidth of each different *HI*-task separately and independently. In [3], the authors also propose a test for checking the schedulability of *LO*-tasks in *LO*-criticality mode, and compute the speed-up factor of their algorithm to be  $4/3$ .

Santy et al. [15] propose an algorithm for letting some of the *LO*-criticality task execute even after the system has switched to *HI*-criticality mode, as long as their execution does not compromise the schedulability of *HI*-tasks. Also, they propose a method to reset the system criticality level at certain specified idle intervals. In this paper, we also propose to continue executing *LO*-criticality tasks as soft real-time tasks even after the system switches to *HI*-criticality: the temporal isolation mechanism guarantees that the *HI*-tasks will not be influenced.

## 6 Conclusions

We presented a reservation-based approach to schedule mixed criticality systems in a way that guarantees the schedulability of high-criticality tasks independently of the behavior of low-criticality tasks. Pessimism related to off-line budget allocation is avoided by an efficient reclaiming mechanism that exploits the budget left by high-criticality tasks for those active low-criticality tasks that can still complete within their deadlines.

We are currently investigating a test for guaranteeing the schedulability of LO-tasks in LO-criticality mode, by computing a lower bound on the amount of reclaiming available in any time interval. We will then compare the schedulability test with the one proposed in [3] to evaluate the performance of the two approaches.

## References

1. S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2010)*, 2010.
2. Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D’Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 61(8):1140 – 1152, 2012.
3. Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D’Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne van der Ster, and Leen Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS’12*, pages 145–154, 2012.
4. Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D’Angelo, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proceedings of the 19th European conference on Algorithms, ESA’11*, pages 555–566, Berlin, Heidelberg, 2011. Springer-Verlag.
5. D. de Niz, K. Lakshmanan, and R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS 2009)*, December 2009.
6. G.Lipari and S.K. Baruah. Greedy reclamation of unused bandwidth in constant bandwidth servers. In *IEEE Proceedings of the 12th Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, June 2000.
7. Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS 2011)*, 2011.
8. K. Lakshmanan, D. de Niz, and R. Rajkumar. Resource allocation in distributed mixed-criticality cyber-physical systems. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS 2010)*, 2010.
9. K. Lakshmanan, D. de Niz, and R. Rajkumar. Mixed-criticality task synchronization in zero-slack scheduling. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2011)*, 2011.
10. H. Li and S. Baruah. An algorithm for scheduling certifiable mixedcriticality sporadic task systems. In *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS 2010)*, December 2010.
11. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.
12. Luigi Palopoli, Giuseppe Lipari, Luca Abeni, Marco Di Natale, Paolo Ancilotti, and Fabio Conticelli. A tool for simulation and fast prototyping of embedded control systems. In Seongsoo Hong and Santosh Pande, editors, *LCTES/OM*, pages 73–81. ACM, 2001.
13. R. Pellizzoni, P. Meredith, M.Y. Nam, M. Sun, M. Caccamo, and L. Sha. Handling mixed criticality in soc-based real-time embedded systems. In *Proceedings of the ACM International Conference on Embedded Software (EMSOFT 2009)*, 2009.

14. S. Petters, M. Lawitzky, R. Heffernan, and K. Elphinstone. Towards real multi-criticality scheduling. In *15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2009)*, 2009.
15. François Santy, Laurent George, Ph. Thierry, and Joël Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In IEEE Computer Society, editor, *ECRTS*, pages 155–165, July 2012.
16. Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007)*, pages 239–243, December 2007.

# On the Average Complexity of the Processor Demand Analysis for Earliest Deadline Scheduling

Giuseppe Lipari<sup>1</sup>, Laurent George<sup>2</sup>, Enrico Bini<sup>3</sup>, Marko Bertogna<sup>4</sup>

<sup>1</sup> LSV, ENS - Cachan, France <sup>‡</sup>

<sup>2</sup> University of Paris-Est, LIGM, France

<sup>3</sup> Lund University, Sweden

<sup>4</sup> University of Modena, Italy

**Abstract.** Schedulability analysis of a set of sporadic tasks scheduled by EDF on a single processor system is a well known and solved problem: the Processor Demand Analysis is a necessary and sufficient test for EDF with pseudo-polynomial complexity. Over the years, many researchers have tried to find efficient methods for reducing the average-case running time of this test. The problem becomes relevant when doing sensitivity analysis of the worst-case execution times of the tasks: the number of constraints to check is directly linked to the complexity of the analysis. In this paper we describe the problem and present some known facts, with the aim of summarising the state of the art and stimulate research in this direction.

## 1 Introduction

The *Processor Demand Analysis* is a necessary and sufficient algorithm for testing the schedulability of a set of real-time synchronous periodic or sporadic tasks to be scheduled by the Earliest Deadline First on a single processor. It was first proposed by Baruah et al. [2], and it was later extended to account for more complex task models, shared resources, etc.

The core of the analysis is the computation of the *Demand Bound Function*: the analysis consists in checking that in each interval of time the function does not exceed the length of the interval (more details in Section 2). The problem has been proven to be NP-Hard [2], in the sense that in the worst case it is necessary to analyse the demand bound function over a number of intervals that is exponential in the number of tasks.

Nevertheless, very efficient algorithms have been proposed. A notable example is the QPA algorithm [4], which iterates over the values of the demand bound function in intervals of decreasing length. In average it requires a very small number of steps to assess the schedulability.

---

<sup>‡</sup> The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 246556.

George and Hermant [3] proposed a characterisation of the space of computation times of the tasks (named C-space) that makes the set schedulable. In practice, the constraint that the demand bound function in a certain interval must be less than the length of the interval is interpreted as an inequality where the computation times are unknown. The schedulability test is a set of inequalities that defines a convex polyhedron in the space of the computation times. In general, each different interval produces a different inequality, hence the number of inequalities is exponential. George and Hermant observed that, given a task set, many of the inequalities are redundant, in the sense that they can be safely eliminated without introducing any new solution. In particular, as we will see in Section 3, the number of necessary inequalities is much lower than the total number of intervals that are necessary in theory.

In this paper we further investigate the method proposed by George and Hermant, with the goal of trying to gain additional insight into the complexity of the problem of testing the schedulability of EDF-scheduled task systems.

## 2 Background

A real-time task  $\tau_i$  is an infinite sequence of jobs,  $J_{i,k}(a_{i,k}, c_{i,k}, d_{i,k})$ , where  $a_{i,k}$  is the job's arrival time,  $c_{i,k}$  is its computation time and  $d_{i,k}$  is its absolute deadline. The goal of a real-time scheduling algorithm is to execute the sequence of incoming jobs on the hardware machine (in our case a single processors) so that each job  $J_{i,k}$  executes exactly  $c_{i,k}$  units of execution time in its execution windows  $[a_{i,k}, d_{i,k}]$ . The Earliest Deadline First scheduling algorithm selects the active job with the earliest absolute deadline.

A sporadic task  $\tau_i$  is characterised by a triplet  $(C_i, D_i, T_i)$ , where  $C_i$  is the worst-case computation time,  $D_i$  is the relative deadline and  $T_i$  is the period, or minimum inter-arrival time. For a sporadic task, the distance between the arrival times of two consecutive jobs is greater than or equal to  $T_i$ ,  $\forall k, a_{i,k+1} - a_{i,k} = T_i$ . Moreover, the deadline is computed as  $d_{i,k} = a_{i,k} + D_i$ , and the computation time never exceeds the worst case computation time,  $\forall k, c_{i,k} \leq C_i$ . In this paper we restrict our attention to sporadic tasks with constrained or implicit deadline, i.e., having  $D_i \leq T_i$ , or  $D_i = T_i$ , respectively.

The *hyperperiod* is computed as the least common multiple of the task periods:  $H = \text{lcm}(T_1, T_2, \dots, T_n)$ . The utilisation of a task  $\tau_i$  is defined as  $U_i = C_i/T_i$ . The total utilisation of a task set  $\tau$  composed of  $n$  sporadic tasks is denoted as  $U = \sum_{i=1}^n U_i$ .

The demand bound function  $\text{dbf}(t)$  is defined as the total amount of computation time of the tasks that have arrival time and deadline in  $[0, t]$ . For a set of real-time sporadic tasks, the  $\text{dbf}$  function can be computed as

$$\text{dbf}(t) = \sum_{i=1}^n \left( \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i. \quad (1)$$

Notice that the  $\text{dbf}$  is a step-wise function which changes values at the absolute deadlines of the jobs. We define as  $\text{dSet}(L)$  the set of all absolute deadlines of



all jobs in interval  $[0, L]$ :

$$\text{dSet}(L) = \{d_{i,k} | d_{i,k} \leq L\}. \quad (2)$$

The following theorem gives a necessary and sufficient condition for schedulability.

**Theorem 1.** *A set of sporadic real-time tasks  $\mathcal{T}$  is schedulable on a single processor by the earliest deadline first scheduling algorithm if and only if:*

$$\forall t \in \text{dSet}(H) \quad \text{dbf}(t) \leq t \quad (3)$$

For a constrained deadline task set with  $U < 1$ , when the computation times of all tasks are known, it is possible to reduce the amount of deadlines to be checked to the first busy period, or to interval  $[0, L^*]$ , as shown in [1], where

$$L^* = \frac{\sum_{i=1}^n U_i(T_i - D_i)}{1 - \sum_{i=1}^n U_i}.$$

### 3 Problem statement

Suppose that we have a task set where we know the periods and the relative deadlines, whereas the worst-case computation times are unknown. Our goal is to compute the *C-space* [3], i.e., all the possible values of the worst-case computation times that make the system schedulable.

To do this, we set-up a system of inequalities using Theorem 1. In particular, we define

$$n_i(t) = \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1,$$

i.e. the number of instances of task  $\tau_i$  entirely contained in interval  $[0, t]$ . Then the inequality can be written as:

$$\begin{cases} \forall t \in \text{dSet}(\mathcal{T}) & \sum_{i=1}^n n_i(t)C_i \leq t \\ \forall i = 1, \dots, n, & C_i \geq 0 \end{cases} \quad (4)$$

This set of inequality defines a convex polyhedron in the space of the variables  $C_i$ : all solutions to this set of inequalities are the vectors that we are looking for.

Since computation times are unknown, we have to analyse all deadlines in  $\text{dSet}(H)$ . However, the number of points in  $\text{dSet}(H)$  may be very large, due to a large hyperperiod. In turns, every point in  $\text{dSet}(H)$  corresponds to a linear inequality. Therefore, it is natural to ask if all such inequalities are strictly necessary.

An inequality can be safely removed if, by doing so, no new solution is introduced. George et al. [3] have shown that indeed it is possible to eliminate many such inequalities. In their paper, they use the Simplex algorithm, to identify the inequalities to remove. Their approach is presented in section 4.

Another equivalent method is to use a geometric characterisation of the polyhedron and computing the smallest convex enveloping polyhedron. We will describe this method in more details in the Section 6.

How many inequalities are removed by applying the elimination procedure? How many are left? Is there any special pattern to identify the minimum set of inequalities, to help us reduce the complexity of the problem *in the average case*?

This paper reports some early investigation on this problem. While we have no definitive answer to these questions, we believe that writing the known facts that we discovered by performing extensive experiments may convince other researchers to help us solve the problem.

## 4 Linear Programming approach

In [3], George et al. show how to characterise the space of feasible WCETs. Considering the WCETs in  $X = (x_1, \dots, x_n)$  as variables and  $D, T$  constants, the C-space is defined by a set of  $s+1$  constraints, where the first  $s$  constraints are derived from the inequalities in Equation 4 corresponding to absolute deadlines in  $\text{dSet}(H)$  and the  $(s+1)^{th}$  constraint is derived from the load utilisation ( $U \leq 1$ ).

They show how to prune the set  $\text{dSet}(H)$  to extract the subset of absolute deadlines representing the most constrained times characterising the C-space. For any time  $t_i \in \text{dSet}(H)$ , they formalise as a linear programming problem in which the constraint corresponding to  $t_i$  is removed, and the objective function is to maximise  $\text{dbf}(t_i)$ . More formally:

### Linear Programming Problem: LP 1

**Maximise**  $\text{dbf}(t_i)$

With  $x_1 \geq 0, \dots, x_n \geq 0$  positive real variables

Under the constraints:

$$\bigcap_{k=1, k \neq i}^s \{\text{dbf}(t_k) \leq t_k\}$$

In [3], the author show that the space of feasible WCETs is convex. Hence, LP1 can be solved by using the Simplex Algorithm. If the solution of the problem is  $\text{dbf}(t_i) \leq t_i$ , then the corresponding inequality (that was just removed to obtain LP1) is redundant, and can safely be eliminated. In fact, the maximum value that the  $\text{dbf}(t_i)$  can assume is however inferior to  $t_i$  even without imposing the constraint. On the other hand, if  $\text{dbf}(t_i) > t_i$ , the corresponding constraint cannot be eliminated without enlarging the C-space.

### 4.1 Numerical Example

We consider a sporadic task set  $\tau = \{\tau_1, \tau_2, \tau_3\}$ , composed of three sporadic tasks  $\tau_i$ , where, for any task  $\tau_i$ ,  $T_i$  and  $D_i$  are fixed, and  $x_i \in \mathbb{R}^+$ , the WCET of task  $\tau_i$ , is variable.

- $\tau_1 : (x_1, T_1, D_1) = (x_1, 7, 5)$ ;
- $\tau_2 : (x_2, T_2, D_2) = (x_2, 11, 7)$ ;
- $\tau_3 : (x_3, T_3, D_3) = (x_3, 13, 10)$ .

In this example, we have:  $D_{min} = 5$  and  $H = 1001$ . To characterise the C-space, we have to consider all absolute deadlines in  $\mathbf{dSet}(H)$  in time interval  $[5, 1001)$ . The cardinality  $m$  of  $\mathbf{dSet}(H)$  is 281.

Therefore, we apply the simplex algorithm on the Linear Programming problem  $LP_i$ , for any time  $t_i \in \mathbf{dSet}(H)$ , starting from time  $t_m$  down to time  $t_1$  (to optimise the computation). We obtain the following subset  $\mathcal{S}_1$  of times in  $\mathbf{dSet}(H)$ :

$$\mathcal{S}_1 = \{5, 7, 10, 12, 19, 40\} \subseteq \mathcal{M}.$$

Since:

$$\begin{cases} x_1 + x_2 \leq 7 \\ 2x_1 + x_2 + x_3 \leq 12 \end{cases} \Rightarrow 3x_1 + 2x_2 + x_3 \leq 19$$

the set can be further reduced to:

$$\mathcal{S}_2 = \{5, 7, 10, 12, 40\} \subseteq \mathcal{S}_1.$$

The exact deadline set characterising the C-space is therefore  $\{5, 7, 10, 12, 40\}$ .

## 4.2 Experiments

We now study the performance of the simplex for pruning the elements in  $\mathbf{S}$ , in the case of constrained deadlines ( $\forall i \in [1, n], D_i \leq T_i$ ).

The first question we ask is how large is the number of points in  $\mathbf{dSet}$  before the reduction. This number strongly depends on the hyperperiod: if periods are co-prime, the hyperperiod can be very large. Suppose that all periods are prime with respect to each other. Then the total number of deadline points generated by task  $i$  is  $\prod_{j=1, j \neq i}^n T_j$ , and the total number of points is:

$$|\mathbf{dSet}| = \sum_{i=1}^n \prod_{j=1, j \neq i}^n T_j.$$

Of course, if the periods are multiples of each other, the hyperperiod and the number of points become much lower.

Notice that the number of constraints in  $\mathbf{dSet}(H)$  depends more on the value of the periods, than on the number of tasks (the number of constraints can be small even for a high number of tasks).

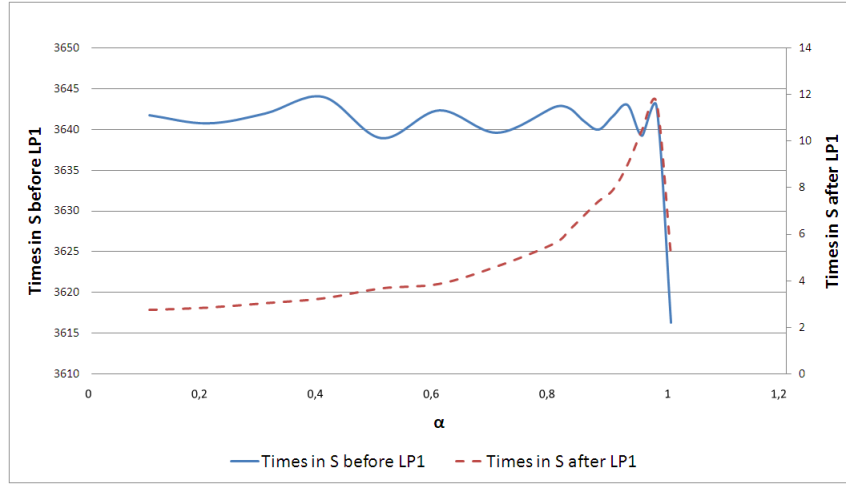
We generated 100,000 task systems, each one containing 3 tasks. For each system, we proceed as follows:

- The period of each task is uniformly chosen from  $[1, 100]$

- The deadline of any task  $\tau_i(C_i, T_i, D_i)$  is  $D_i = \alpha T_i$ .  $\alpha$  is chosen in the intervals  $[0, 0.8]$  and  $[0.8, 1]$  with a granularity of respectively 0.1 and 0.025.

We focus on the influence of  $\alpha$  on the size of the  $\text{dSet}(H)$  after pruning the constraints.

Figure 1 shows the results of our analysis. The average number of elements in  $\mathcal{S}$  over all generated systems is represented by the solid line and associated to the left axis as a function of  $\alpha$ . The dotted line refers to the average number of elements obtained after the simplex is applied to the linear programming problem LP1 and must be read according to the right axis, as a function of  $\alpha$ .



**Fig. 1.** Reduction of elements in  $\mathcal{S}$  with LP1

We notice that the number of elements which curb the C-Space inch-up in  $[0.1, 0.6]$  then descend when  $\alpha$  tends toward 1. If  $\alpha = 1$ , we are in the special case of implicit deadlines where the only constraint is the processor utilisation constraint:  $U = \sum_{i=1}^n \frac{x_i}{T_i} \leq 1$ .

In all generated systems with  $\alpha < 1$ , we have found that the number of constraints before and after pruning the set  $\mathcal{S}$  is respectively higher than 3570 and less than 12. For a load less than 0.6, the average number of constraints in  $\mathcal{S}$  after pruning the elements in  $\mathcal{S}$  is less than or equal to 4. Similar trends are confirmed for larger task sets.

This confirms that the average complexity of the feasibility problem is much smaller than the worst-case. The Simplex is very effective in this reduction. However, the Simplex is itself an algorithm with high computational complexity in the worst-case: and the Simplex must be run for every point in  $\text{dSet}(H)$ . Therefore, even if the final number of points is very low, to obtain such a reduction we need to apply a complex algorithm an exponential number of times.

We then need to investigate if there is some other property of the problem that can be used to reduce the number of points.

## 5 The definitive idle time approach

One explanation for the increasing trend in Figure 1 can be given by the presence of definitely idle times.

**Definition 1.** A time  $t_d$  is said to be a **definitive idle time (DIT)** if at time  $t_d$  there is no task released before  $t_d$  having an absolute deadline after  $t_d$ .

A DIT must remain idle for any value of the computation times that makes the task set schedulable. In facts, a DIT is a time in which no schedulable job can execute.

A DIT exists only if all tasks in the task set have constrained or implicit deadlines (i.e.  $\forall i, D_i \leq T_i$ ). In this case, one DIT is point  $H$ , because at the hyperperiod all jobs must have terminated, and no new job has arrived yet.

Consider, as an example, the task set consisting of two tasks,  $\tau_1 = (D_1 = 5, T_1 = 8)$  and  $\tau_2 = (D_2 = 9, T_2 = 15)$ . All times in  $[13, 15]$  are definitely idle because  $\tau_1$  has jobs  $J_{11} = (0, 5)$ ,  $J_{12} = (8, 13)$ ,  $J_{13} = (16, 21), \dots$ , and  $\tau_2$  has jobs  $J_{21} = (0, 9)$ ,  $J_{22} = (15, 24), \dots$ , and there is no job active in interval  $[13, 15]$ . Therefore, for any value of  $C_1$  and  $C_2$  for which the task set is schedulable, interval  $[13, 15]$  will remain idle, and hence any initial busy period has length less than 13.

It is easy to show that all constraints corresponding to deadlines after  $t = 13$  are redundant.

**Lemma 1.** Let  $t_1$  be a definitely idle time. Then for any  $t > t_1$ ,  $\text{dbf}(t) \leq t$  is a redundant inequality.

*Proof.* Since  $t_1$  is a DIT, for any  $t > t_1$  we can rewrite:

$$\text{dbf}(t) = \text{dbf}(t_1) + \text{dbf}(t - t_1) = \text{dbf}(d_1) + \text{dbf}(d_2)$$

where  $d_1$  is the latest deadline no later than  $t_1$ , and  $d_2$  is the latest deadline no later than  $(t - t_1)$  after a critical instant, i.e., the synchronous periodic arrival of all task instances.

Then

$$\begin{cases} \text{dbf}(d_1) \leq d_1 \\ \text{dbf}(d_2) \leq d_2 \end{cases} \Rightarrow \text{dbf}(t) \leq d_1 + d_2 \leq t$$

□

Clearly, the smaller the ratio between relative deadlines and periods, the highest is the probability that the first DIT happens quite soon in the schedule. Hence, one possible reason for the trend in Figure 1.

Consider again the previous example: time 13 is the first DIT, hence only deadlines  $\text{dSet}(13) = \{5, 9, 13\}$  have to be considered, and in fact the corresponding three inequalities are all non redundant. Therefore, thanks to the concept of DIT, in this particular example we defined precisely the minimum set of inequalities. However, this is not true in general. Consider the example task set in Table 1.

| <i>Tasks</i> | D  | T  | Jobs Windows                          |
|--------------|----|----|---------------------------------------|
| $\tau_1$     | 7  | 9  | $[0, 7], [9, 16], [18, 25], [27, 34]$ |
| $\tau_2$     | 12 | 15 | $[0, 12], [15, 27], [30, 42]$         |

**Table 1.** Example 2

The first DIT is at time 27, and all deadlines in  $[0, 27]$  are  $\text{dSet}(27) = \{7, 12, 16, 25, 27\}$ : of these, the inequalities corresponding to  $\{7, 12, 16, 27\}$  are necessary, while the inequality corresponding to deadline 25 is redundant. In fact,

$$\begin{cases} 2C_1 + C_2 \leq 16 \\ C_1 \leq 9 \end{cases} \Rightarrow 3C_1 + C_2 \leq 25.$$

Therefore, not all deadlines before the first DIT correspond to necessary inequalities. As the ratio between deadlines and periods approaches 1, and as the number of tasks  $n$  increases, the first DIT happens quite late in the schedule; nevertheless, many inequalities are still redundant, and the minimum number of necessary inequalities is small.

### 5.1 First definitive idle time computation

Notice that  $t_d = H$  is a DIT in the constrained deadlines case. Indeed,  $W(H) = U \cdot H$  and

$$n_i(H) = \left\lfloor \frac{H + T_i - D_i}{T_i} \right\rfloor = \frac{H}{T_i} + \left\lfloor \frac{T_i - D_i}{T_i} \right\rfloor = \frac{H}{T_i}$$

as  $D_i \leq T_i$ . Thus  $\text{dbf}(H) = U \cdot H = W(H)$ : for constrained deadlines,  $H$  is not necessarily the first DIT.

Our goal now is to propose an algorithm to compute the first DIT before  $H$ , if it exists. For a task  $\tau_i$ , a first DIT  $t_d$  corresponds to a time between a deadline of  $\tau_i$  and the next release time of  $\tau_i$  after this deadline. This corresponds modulo the task period to a time between  $[D_i, T_i]$ .

Thus, finding the first DIT is equivalent to find the first time  $t_d$  satisfying:

$$\begin{aligned} \forall i, t_d &= a_i \text{ Mod } T_i \\ a_i &\in [D_i, T_i] \end{aligned} \tag{5}$$

This problem can be solved by using the general Chinese Remainder Theorem.

**Theorem 2 (General Chinese Remainder Theorem).** *Suppose  $T_1, T_2, \dots, T_n$  are positive integers which are pairwise coprime. Then, for any given sequence of integers  $a_1, a_2, \dots, a_n$ , there exists an integer  $x$  solving the following system of simultaneous congruences.*

$$\begin{aligned} x &\equiv a_1 \pmod{T_1} \\ x &\equiv a_2 \pmod{T_2} \\ &\vdots \\ x &\equiv a_n \pmod{T_n} \end{aligned}$$

Furthermore, all solutions  $x$  of this system are congruent modulo the product,  $H = T_1 T_2 \dots T_n$

Notice that the theorem requires all integers to be pairwise coprime. In this particular case, one DIT is the solution of:

$$t_{idle} = \sum_{i=1}^n a_i \frac{H}{T_i} \left[ \left( \frac{H}{T_i} \right) \right]_{T_i}^{-1} \pmod{T_1 T_2 \dots T_n} \quad (6)$$

for  $a_i \in [D_i, T_i]$

where  $[a^{-1}]_b$  stands for the multiplicative inverse of  $a \pmod{b}$  i.e. the number  $y$  such that  $ay = 1 \pmod{b}$ .  $t_d$  is thus the minimum value satisfying equation 5 i.e:

$$t_d = \min_{a_i \in [D_i, T_i]} t_{idle} \quad (7)$$

If we compute the first definitive idle time in the example used for the linear programming approach where periods are all pairwise coprimes and  $H = T_1 T_2 T_3 = 1001$ :

- $\tau_1 : (x_1, T_1, D_1) = (x_1, 7, 5)$ ;
- $\tau_2 : (x_2, T_2, D_2) = (x_2, 11, 7)$ ;
- $\tau_3 : (x_3, T_3, D_3) = (x_3, 13, 10)$ .

We find the minimum value of  $t_{idle}$  for  $a_1 = 6, a_2 = 7$  and  $a_3 = 10$ :

$$\begin{aligned} t_d &= a_1 T_2 T_3 [(T_2 T_3)^{-1}]_{T_1} + a_2 T_1 T_3 [(T_1 T_3)^{-1}]_{T_2} + a_3 T_1 T_2 [(T_1 T_2)^{-1}]_{T_3} \\ &= 6 \cdot 11 \cdot 13 \cdot 5 + 7 \cdot 7 \cdot 13 \cdot 4 + 10 \cdot 7 \cdot 11 \cdot 12 = 16078 \end{aligned}$$

and  $16078 \pmod{T_1 T_2 T_3} = 62$ . Thus  $t_d = 62$ . Hence, according to this approach, we only need to consider the deadlines in  $[0, 62]$ . To these, we can apply the Simplex algorithm, thus reducing considerably the time to derive the minimum set of inequalities.

In the general case where periods are not coprime, it is possible to use one of the algorithms for solving systems of simultaneous congruences, for example the method of successive substitutions.

Thus, applying the definitive idle time approach can help reducing the initial time interval to consider before applying the linear programming approach.

## 6 Geometric interpretation

We now look at the problem from a slightly different point of view. As discussed in Section 3, for each deadline  $d \in \text{dSet}(L)$ , we have an inequality of the form

$$\sum_{i=1}^n n_i(d) C_i \leq d$$

where  $n_i(d)$  is the number of jobs of  $\tau_i$  entirely included in interval  $[0, d]$ . Using some simple algebra, we can rewrite the inequality as:

$$\sum_{i=1}^n \frac{n_i(d) T_i}{d} U_i \leq 1$$

In vectorial form:

$$\mathbf{V}(d) \cdot \mathbf{U} \leq 1 \quad (8)$$

where  $\mathbf{V}(d)$  is the vector whose  $i$ -th component is  $\frac{n_i(d) T_i}{d}$ ,  $\mathbf{U}$  is the vector of task utilisations, and the dot represents scalar product. Hence, the set of deadlines defines a set of vectors  $\mathbf{V}$  in the space of the task utilisations.

A vector is redundant if it can be expressed as a linear combination of other vectors with positive coefficient whose sum is less than 1.

**Theorem 3.** *A vector  $\mathbf{V}(d)$  is redundant if there exists at least two other vectors  $\{\mathbf{V}(d_1), \dots, \mathbf{V}(d_h)\}$ ,  $h \geq 2$  such that*

$$\mathbf{V}(d) = k_1 \mathbf{V}(d_1) + \dots + k_h \mathbf{V}(d_h), \quad \sum_{j=1}^h k_j \leq 1$$

*Proof.* Equation (8) must be true for all vectors. We show that under the hypothesis, the inequality for  $d$  can be eliminated.

$$\mathbf{V}(d) \cdot \mathbf{U} = \sum_{j=1}^h k_j \mathbf{V}(d_j) \cdot \mathbf{U} \leq \sum_{j=1}^h k_j \leq 1$$

□

The implication of the Theorem is that the set of necessary points deadlines is identified by the convex hull of all the points  $\mathbf{V}(d)$ , i.e. the most external points that define a convex polyhedron in the utilisation space. To clarify the issue, let us make an example.

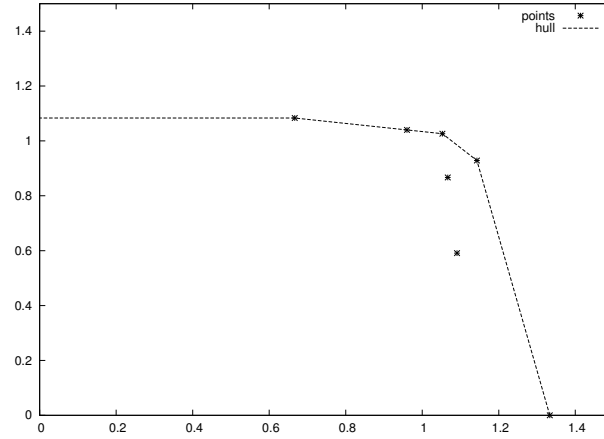
Consider the task set of Table 2. The first DIT is at 38, hence the set of deadlines to consider is:  $\text{dSet}(38) = \{6, 12, 14, 22, 25, 30, 38\}$ . Each of these deadlines corresponds to a vector in the space of utilisations  $< U_1, U_2 >$ . The resulting space is shown in Figure 2.

As a result, the minimal set of deadlines to consider is  $\text{dSet}^* = \{6, 12, 14, 38\}$ . Unfortunately, computing the convex hull is in general rather complex. Specialised algorithms exist in 2 and 3 dimensions that have complexity  $O(s \log(s))$



| Tasks    | D  | T  | Jobs  |
|----------|----|----|---|
| $\tau_1$ | 6  | 8  | [0, 6], [8, 14], [16, 22], [24, 30], [32, 38] |
| $\tau_2$ | 12 | 13 | [0, 12], [13, 25], [26, 38], [39, 51]         |

Table 2. Example 3



**Fig. 2.** Representation of  $\mathbf{V}(d)$  in the utilisation space for the tasks in Table 2, together with the convex hull.

and  $O(s^2)$ , respectively, where  $s$  is the number of points in  $\mathbf{dSet}$ . However, in the  $n$ -dimension case the complexity is  $O(s^{\lfloor \frac{n}{2} \rfloor + 1})$ , therefore it grows exponentially in the number of constraints. The DIT is even more important in the case, as it can help reduce  $s$  before applying the convex-hull algorithm.

## 7 Conclusion

In this paper we have described the problem of computing the C-Space of a set of sporadic tasks scheduled by EDF on a single processor. The method amounts to using the Processor Demand Analysis for generating a set of inequalities that define the space. We have seen that in practical situations, many of the inequalities are redundant and can hence be eliminated, reducing the complexity of the C-Space representation. However, methods for reducing the number of inequalities are themselves complex and require a certain amount of computation.

We have observed that the concept of Definitely Idle Time can help us reducing the number of constraints to analyse in the case of constrained deadline systems. The method is particularly effective when the differences between the relative deadlines and the periods are large. Then, we have proposed a different point of view on the problem by using a geometric interpretation.

The observations reported in this paper are important for better understanding the complexity of feasibility analysis for single processor systems. While the

problem has been proved to be NP-hard, the facts reported in this paper show that in practical cases the complexity is rather low. These facts are in accordance with other results on similar problems, as reported by Zhang and Burns [4].

We believe, however, that the investigation is not concluded. In addition to the Definitely Idle Time, other properties could be used to further reduce the set of non-redundant deadlines quickly and more effectively. The results could shed light on more fundamental properties of the feasibility problem.

## References

1. Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. pages 182–190, 1993.
2. S.K. Baruah, L.E. Rosier, and R.R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *The Journal of Real-Time Systems*, 2, 1990.
3. L. George and J.F. Hermant. Characterization of the space of feasible worst-case execution times for earliest-deadline-first scheduling. *Journal of Aerospace Computing, Information, and Communication*, 6(11):604–623, 2009.
4. F. Zhang and A. Burns. Schedulability analysis for real-time systems with edf scheduling. *Computers, IEEE Transactions on*, 58(9):1250–1258, 2009.

# Perspectives on the Analysis and Design of Real-Time Distributed Systems with UML-based standard modelling languages

Julio L. Medina,

Departamento de Electrónica y Computadores, Universidad de Cantabria,  
39005-Santander, SPAIN  
julio.medina@unican.es

**Abstract.** This paper considers the design of hard real-time distributed systems. It uses a model-based approach whose specification is made using UML, a high level standard modelling language. This work points out the need for modelling standards in this domain and the role that those other standards closely related may play in an integrated model-based development process. It also describes a tool-aided methodology that uses the UML profile for Modelling and Analysis of Real-Time and Embedded systems (MARTE), a recent modelling standard of the OMG to enable the assembly and transformation of such design intended models into schedulability analysis models. These analysis models are suitable for the verification of the timing properties of a fully described system in a real-time situation. The description of a real-time situation includes also the knowledge of the load the system is expected to support. The methodology proposed uses also a new Ada code generator that helps to get in tune the structure of the applications described by the high level application modelling concepts provided by MARTE, with its corresponding schedulability analysis models. The tool associated to this methodology generates as an output the concrete analysis models used by the MAST set of tools; it invokes MAST, and also recovers the output results back into the UML models.

**Keywords:** Ada, embedded systems, MARTE, MDA, MDE, modelling, OMG standards, real-time, schedulability analysis, UML.

## 1 Introduction<sup>1</sup>

The continuous study and increasing progress in the development of schedulability analysis techniques for the timing verification of hard real-time systems is an effort for having better, safer, and more efficient automated products. This is not an effort for making them cheap. As any other scientists, having a philosophical view of our role in society, we keep the hope that our results, as small as that they can seem to

---

<sup>1</sup> This work has been funded in part by the Spanish Government and FEDER funds under grant TIN2011-28567-C03-02 (HI-PARTES). It reflects only the author's views; the funding agents are not liable for any use that may be made of the information contained herein.

others, would be part of human development someday. This is one of our particular ways of dreaming of eternity. For good or for sad, the monetary capitalist system we face or enjoy puts this labour in the hands of profit based enterprises. The quality of the final software products is usually planned and measured according to the profit they may generate. Hence strategies like designing for reusability, maintainability, and alike are now part of most software engineering procedures and policies in industry. For those application domains in which timing verification is advisable or needed, a more or less strong effort is made in doing a lot of testing. Only domains in which the potential profit lost is huge or a strong external regulation exists have aspects like timing verification, schedulability analysis techniques, or any other sort of real-time aware design strategy included in their production cycle. The aerospace, the aeronautics, and recently the automotive domains are examples of the later.

Among the factors that contribute to these situations we may find that: mastering these techniques requires some extra learning effort, they use some additional math, which is not really very complex but enlarge the collection of skill developers need to have, and most importantly, there is not a smooth match between the hardware platforms available at reasonable price, the software development environments widely known by practitioners, the programming paradigms, languages, compilers, and the operating systems that are mostly available. Moreover, the large number of research solutions that we provide and are narrowly applicable to very specific cases, the low level of publicly known experience results of these techniques in industry, and the growing gap between the state of the practice and the state of the art are making the real industrial exploitation, the digestion and the maturation process of these techniques chaotic and in summary slow.

All these said, there is still some space for hope. Bit a bit some of the industries that are considering using these techniques identify the challenges and look for solutions to integrate them into their production procedures. The usual mechanism is to find a specialized company that provides a commercial tool, has the expertise, and takes some how part of the responsibility. Whichever it is the technique that is going to be used (static WCET evaluation, RMA offset-based, or even synchronous languages for example), from a purely practical point of view it requires at least yet another formalism to integrate the design/analysis/verification model creation and maintenance into the chain of artefacts that support the process. Model-based development strategies may help to get over these caveats and get at least some automation in the generation, extraction, and management of analysis models. But the jump into a full model-based development process is a strategic decision for an enterprise. It may resemble the even today not fully adopted change from structural to object-oriented software decomposition in the general purpose information systems domain. Reached this point it is important to observe that threatens like the poor tools interoperability and the potential dependency on one or very few tool providers may make the bet for this approach risky or very expensive.

## **1.1 The need for standards**

These situations lead us to easily observe the benefits of having sound and widely accepted industry modelling standards. They may help large corporate end users,

academia, big and small tool providers, and practitioners in general to deposit in each modelling effort the added value of reusability and further comprehensibility. Creating standards is a collective effort, in which each agent gives as much as possible in the aim of easing the subsequent implementation of the standard. But there are several magic things about standards. One is that their preparation makes each agent learn from each other through deep technical discussions in which the humility and generosity required by the scientific method is put on trial. When the effort is made with real scientific objectives in mind and stronger arguments prevail, the resulting standard has many possibilities of being a real asset for the research community. Besides, it will be suitable for its technical purposes whichever the business patterns for its exploitation.

## **1.2 The Unified Modeling Language**

The Unified Modeling Language (UML) [12] was brought as a syncretism of other at the time well known formalisms for the representation of object-oriented models. Since its appearance in scene it has been experienced as a modelling panacea and disappointment at the same time for almost anyone that has used it intensively. It lacks strong semantics but provides flexibility. It enables communication but almost every tool vendor makes its own model storage format incompatible to the others. It enjoys the beauty and expressive power of a graphical representation for almost any sensible kind of model needed in this domain but the amount of data that any serious application needs to really specify a system does not fit in a single screen shot.

The success of UML is precisely this ambiguity. For a conceptual modelling language the key to cope with complexity is the mastering of abstraction. In its vocabulary the initial UML provided a number of tokens that give the object oriented modeller some moderate power for managing complexity. But soon the standards proposed also the use of a bare basic structural UML as a meta-language to define others, including the very same full UML. These meta-object facilities plus to the abundance of UML tools brought the illusion of super abstraction powers, and have promoted UML far ahead from its initial object orientation.

Then each (complexity domain) community has seen in the formalization of extensions to UML a means to get their own vocabulary into the mainstream of modelling tools. As examples consider BPMN, SOA, UPDM, Healthcare, Security, Systems Engineering, and a large etc. Ours is a relatively small community in this context, but significant effort has been made in the creation of a standard that complements UML for the modelling and analysis of real-time and embedded systems: MARTE [3].

## **1.3 The UML Profile for MARTE**

This standard is made in the form of a UML Profile, a way of extending UML that is included in the UML standard itself. It collects the modelling elements required for addressing a number of concerns in the real-time and embedded systems domain that were not in the original UML. Notorious extensions include those for annotating

numerical values and units; modelling extra-functional properties; timing constraints; diverse configurations of operating modes for a system model; a full taxonomy of generic resources and their static and dynamic usage; hardware and software platforms (through the modelling of hardware resources and OS services); high level real-time applications by means of tasks (called real-time units), and passive protected (shared) objects; and a concrete semantics for the communication between components through ports and interfaces. From its analysis dimension it proposes generic extensions for quantitative analysis modelling and specific extensions for performance and schedulability analysis. It includes all the necessary language extensions for the construction of schedulability analysis models.

The concrete effort that this paper shows for the generation of schedulability analysis models uses MARTE as an input formalism but in the context of the full model-based design methodology there are several other standardization efforts that result useful to take into account.

#### **1.4 Standardization efforts related to real-time and embedded systems.**

The closer in time and scope is the OMG Systems Modeling Language (OMG SysML™) [13]. This standard re-uses only a part of UML (constraining the rest) and extends it with a UML Profile, a library of modelling elements and new diagrams that fit the purpose of creating visual and interchangeable system engineering models. Many of the concerns of this standard are shared by MARTE, particularly the way of expressing real values and the semantics of the communication between components (blocks). SysML plays a role particularly in the initial definition of the system and the partitioning of responsibilities between hardware and software elements.

In the chain of tools and models that interact along a model-based development process they appear also some issues that have been addressed by other standards of the OMG. To express constraints it is used The Object Constraint Language (OCL), Version 2.3 (<http://www.omg.org/spec/OCL/2.3/>). To grant interoperability between tools the OMG proposes The Meta Object Facility MOF 2 XMI Mapping, Version 2.4 (<http://www.omg.org/spec/XMI/2.4/>). Others are related to the way models are treated in the approach, for which the OMG promotes the Model Driven Architecture initiative (MDA). This architecting view proposes to have Platform Independent Models (PIM) as abstract/relocate-able representation of the desired system, and Platform Specific Models (PSM) for the different concrete usages of the PIM. The realization of the PSM is obtained by using model transformations of the PIM. The full exploitation of these possibilities include the writing of the transformations in a standardize language like those in Query Views and Transformations (QvT) [14].

Several other standards or initiatives are under maturation process but may require our attention. The Foundational UML (fUML) is an effort to fix the semantics of a reduced subset of UML, so that their use particularly in the dynamic (behavioural) models may lead to the same results disregarding the implementation platform. Textual annotations (much like code programming) may be placed also in behavioural models with the Action Language for fUML (Alf). Finally a new standard, the Precise Semantics of UML Composite Structures [10] is coming to adjust the use of fUML and Alf to describe the internal description and the relationships between components.

### 1.5 Focus on schedulability analysis.

Model-based software development is called to be one of the most promising software engineering approaches. Having reusable, configurable, and composable models may help in the separation of concerns, the increase of development efficiency, and even enhancing the quality of the software at large.

For the applications with real-time requirements, a model-based methodology can additionally help to ease the process of building their timing behaviour analysis models. These models usually constitute the basis of the real-time design and the schedulability analysis validation processes. With that purpose, the designer of a software (or even hardware) real-time component must generate, in synchrony with the models used to generate the component's code (or the hardware specification), an additional parameterisable model. This other model must be suitable for the timing validation of the system resulting out of the usage of that component in the composition as a whole. The analysis model for each component abstracts away implementation details but retains the timing behaviour of all the actions it performs. In particular it needs to include all the scheduling, synchronization and resources information that is necessary to predict the real-time qualities of the applications in which it might be integrated. In the approach that we present here, these analysis models are to be automatically derived from high level software design models annotated with a minimum set of real-time features. The input data are taken from the requirements of the application in which they are to be used. In analogy to the generation of the application's code as a composition of the code of its constituent components, the analyst, or application designer, can also compose the set of real-time sub-models, and build the complete real-time analysis model of the application. This strategy helps the designer to get rid of the tedious and error prone task of building in one piece the complete reactive model of the application.

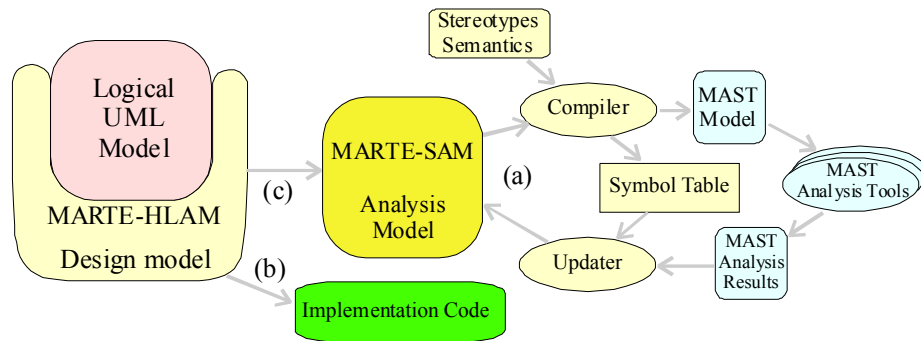
A discussion of the process followed for the design of the real-time characteristics of an application in a strict component-based development methodology may be found in [1]. Here we explore the semantics of the modelling elements provided by MARTE in both sides of its principal concerns: analysis and design. It also exploits them with the aim of enabling the automatic extraction of analysis models from high level design models. This task is feasible provided the design space is restricted to those constructs that are safely implementable by specific analysable patterns described in Ada. From a model based engineering perspective, those Ada patterns serve the purpose of conducting the code generation consistently with the schedulability analysis capabilities associated to the design modelling constructs used. Those basic constructs are in essence implementable as tasks and shared objects [11]

The rest of the paper is organized as follows: Section 2 presents the rationale for the modelling approach and visits some related work. Section 3 describes the modelling and extraction tools used for the generation of the output MAST analysis models [2]. Section 4 gives guidelines for the construction of high level application design UML models that can be transformed in an automated way in the respective schedulability analysis models. Section 5 presents our initial experiences with the analysis models for an example application. Finally we draw some conclusions and outline future work.

## 2 The approach

The UML Profile for MARTE [3] brings a large number of modelling constructs and concepts that may be used for realizing schedulability analysis in a variety of ways. This effort has been driven by the goal to enable, thanks to those modelling constructs, (1) early V&V and (2) the iterative use of the models created. These requirements are key elements of any development process that aims to reduce integration costs while assuring predictability.

In order to cope with complexity, to manage the risks associated to the research and the development of tools efforts, and also to make better use of the modelling resources offered by MARTE, the complete problem has been divided in two challenging but achievable research steps. Fig. 1 illustrates these two challenges in the context of the model processing paradigm. The picture shows a re-visited version of the approach followed in a previous work [4]. Now we consider separate representations of both: the design and the analysis models in UML, and use the MARTE standard for them.



**Fig. 1.** The Model processing paradigm in the design and analysis approach.

The first step addressed by our research effort (a) comprises the definition and manipulation of what we will denominate the "analysis models". This effort is described in Section 3 and includes two modules; one (Compiler) that extracts the MAST models, and another (Updater) that recovers analysis results back into the UML models.

A second part (b) is the generation of the implementation code in Ada from structural and behavioural diagrams in UML. Finally this effort (c) proposes the specification and automation of those analysis modelling constructs related to what we call the "design models". These models are usually entered by hand in the UML tool. Instead, the analysis models can be obtained either through model transformation from the design models, or if needed entered by hand.

The proposed methodology extends the regular UML description of a system with a design model which includes some particular MARTE modelling elements describing specific real-time features. These constructs must be sufficient to generate the analysis model. An analysis model defines an additional view of a particular situation of the system that is subject to real-time requirements and expresses:



- the computational capacity of the hardware and software resources that constitute the platform,
- the processing requirements and synchronization artefacts that are relevant for evaluating the timing behaviour of the execution of the logical operations, and
- the real-time situations to be evaluated, which include the workload and the timing requirements to be met.

From another point of view, this method also helps to support the design of applications in terms of composable parts. As far as their granularity is concern, these parts are closer to the concept of real-time objects than to the CBSE (Component-based Software Engineering) interpretation of components. In a fully component-based approach, the creation of the analysis models would have to be made as a combination of structural elements plus the recursive inclusion of their behaviour invocations following their precise deployment. In a model-driven approach, this later strong form of composability is in a higher level of abstraction, but still may benefit from the approach described here in order to assess a variety of non-functional properties, in our case its timing properties by means of schedulability analysis.

In order to constrain the design space to the patterns that may be analysed by the currently available schedulability analysis techniques, the models of computation implicit in the high level application modelling constructs offered by MARTE are restricted to those that have been studied in our previous work [4]. This implies the use of Ada platforms with both the Real-Time System as well as the Distributed Systems annexes of the Ada standard.

Considering some related work it is relevant to mention a similar effort that has been proposed as a result of the ASSERT project [5]. Though the solutions provided there disregard the semantics framework of the design models in the MARTE standard (by using an ad-hoc profile), and commit for single processor systems, it is a very relevant effort that shows the main formalisms and the technical and industrial trends in place. Our approach differs in several aspects, first (1) the analysis models are also expressed using UML plus the MARTE standard so that they may be used for additional transformations not only to MAST but to other tools if necessary. Then (2) the constraints over the design model are not restricted specifically to the Ravenscar profile but to those of the finally used techniques, which in our case are the offset based holistic analysis techniques used for distributed systems. Finally, (3) as already mentioned the formalism to express the model of computation with the real-time features is the corresponding high level application modelling chapter of the MARTE standard itself.

Considering the conversion from MARTE to schedulability analysis tools in particular, there are some other efforts to mention. The closest in style and modelling capabilities is the RSA plugin to perform schedulability analysis with RapidRMA [8]. The version of this tool that is available shows some limitations: it supports scheduling analysis for mono-processors, with periodic and sporadic events (through sporadic servers). It does not provide support for multi-processors and distributed systems. RapidRMA and IBM RSA are not integrated through the GUI. Moreover, there is no automatic launch of RapidRMA after the input files are generated. It requires a manual operation. The current implementation does not offer any feedback capabilities from RapidRMA to the UML modelling tool. All the analysis results can

be exploited within the tool only. Similar limitations plus a lack in modelling guidance is provided by the tool in [9], which aims to represent Cheddar models with MARTE. That document shows how MARTE concepts can be matched to those used in Cheddar in order to do analysis on models and proposes model transformation solutions using ATL.

### 3 Analysis models

The first, and so far sufficiently solved problem, has been the definition/selection of which and how elements in MARTE are to be used in the creation of schedulability analysis models. These elements are the basis for the tool that has been developed for the generation of MAST analysis models taken from UML+MARTE annotated analysis models. Following previous research efforts [6], MARTE provides concepts to structure the analysis models using three main categories: The platform resources (a), the elements describing the logical behaviour of the system constituent parts (b), and finally the real-time situations to be analysed (c). The precise mapping from MARTE to MAST elements may be easier to see by inspecting the code, nevertheless here we summarize a condensed view of the MARTE elements proposed for their use in each of these three main categories.

**Table 1.** Modelling elements in MARTE used for the creation of MAST schedulability analysis models.

| <i>Platform Resources</i>  | <i>Behavioural Models</i>                                  | <i>Real-Time Situations</i>   |
|--|--|---|
| GaResourcesPlatform<br>SaExecHost *<br>SaCommHost *<br>SaSharedResource *<br>SchedulableResource * | GaWorkloadBehavior<br>GaScenario<br>SaStep *<br>SaCommStep | SaAnalysisContext *<br>GaWorkloadEvent *<br>Allocate<br>Allocated<br>Assign<br>SaEndToEndFlow *<br>SaSchedObs<br>GaLatencyObs * |
| * Elements used in the marte2mast extraction tool in its current version.                          |  |   |

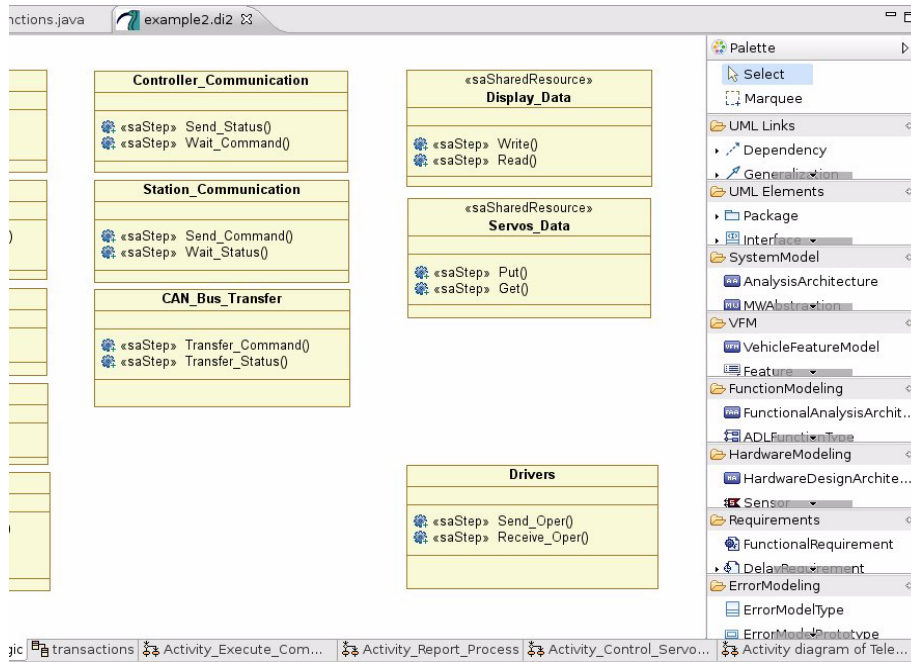
In the tool that has been provided for the generation of MAST models from UML+MARTE models, the platform elements and the logical (software) components are modelled as a set of structural elements with stereotypes annotated on them. Fig. 2 shows an example of the usage of these elements in the modelling of a tele-operated robot distributed application.

The stereotype annotations shown there represent the timing models of their respective operation software entities. The timing properties (i.e. the worst case execution time) for each of them appear in the properties tab of the tool corresponding to the SaStep stereotype used. The class “Drivers” models the overheads due to the sending and reception of messages; it is used to represent in SAM the modelling elements called “drivers” in MAST. The elements used for modelling the platform

include nodes, networks, tasks, channels, and operations that will be invoked as part of the internal platform behaviours.

The end-to-end flows that described execution scenarios are modelled using sequence charts or activity diagrams. Fig. 3 shows an example of the usage of these elements in the modelling of a tele-operated robot distributed end-to-end flow. Some steps in the flow are shown directly in the activity as stereotyped actions, others are statically defined somewhere else in the model but are modelled in the flow as invoked subUsages of the shown actions. These internal steps are not visible in the diagram but are stored in the model and retrieved by the tool recursively.

The full example is available in the web page of the tool: <http://mast.unican.es/umlmast/marte2mast>. There the reader can find the specification of the example application as well as the models used for its analysis. These are delivered in the form of an eclipse workspace containing all the models.



**Fig. 2.** View of the tool showing structural modelling elements for schedulability analysis.

In summary, in the current version of this tool, the elements taken from MARTE to generate the MAST analysis models are:

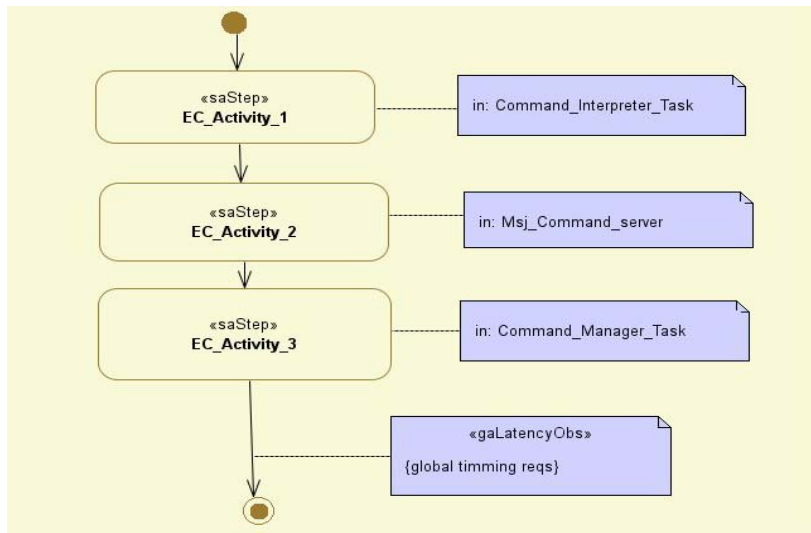
```
> Processing_Resource <= SaExecHost, SaCommHost
> Scheduler <= SaExecHost, SaCommHost
> Scheduling Servers <= SchedulableResource
> Shared_Resource <= SaSharedResource
```

```

> Operations <= SaStep <= Sequence/activity Diagram
    plus subUsages (ordered list of called operations)
> Transactions <= Sequence/Activity Diagram +
    GaWorkloadEvent + GaLatencyObs

```

This effort has been implemented using the Eclipse technologies provided by PapyrusUML as graphical tool, the UML2 plugin as model repository, and the Acceleo plugin for the extraction of text from the UML2 models plus a significant number of Java functions. The code used as well as the scripts created are shared as open source. The current version supports the modelling of end-to-end-flows by means of activity diagrams and the composition of independently characterized timed behaviours. It also includes the invocation of the MAST tools from the eclipse environment and the recovery of the results back into the UML model. It may be downloaded from <http://mast.unican.es/umlmast/marte2mast>.



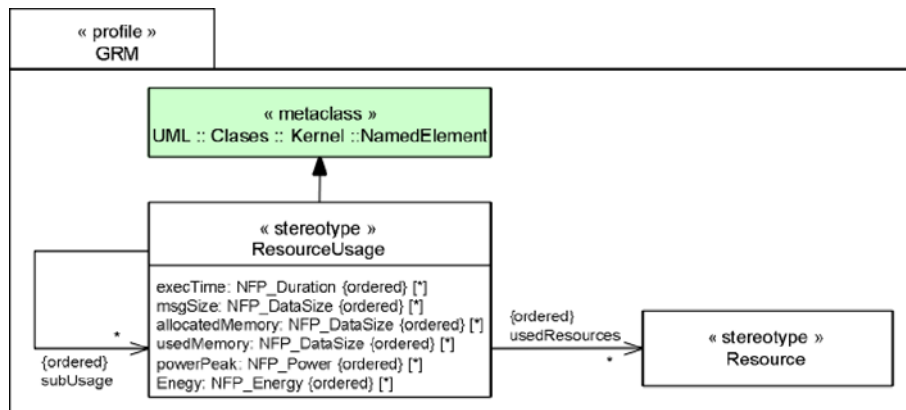
**Fig. 3.** View of the tool showing behavioural modelling elements for schedulability analysis.

The high level algorithm to do the extraction of the MAST model is easy to identify following the script in `marte2mast.mtl`. Here you may see a summary of it:

- 1) Processing Resources and Schedulers take their attributes from:  
Classes with stereotype 'SaExecHost' or 'SaCommHost'
- 2) Scheduling Servers in processors and networks take their attributes from:  
Classes with stereotype 'SchedulableResource' and 'GaCommChannel'
- 3) Shared Resources (critical sections) take their attributes from:  
Classes with stereotype 'SaSharedResource'
- 4) The real time situation is taken from:

- A Class with the stereotype 'SaAnalysisContext'
- 5) Finally the Transactions in it are extracted from:  
Activities with the stereotype 'SaEndtoEndFlow'

A recursive strategy is used for retrieving composed operations and assigning them to the appropriate MAST activity-handlers. There are two ways of expressing these composed operations. One is static and is based in the use of the recursive capabilities of the subUsage association of the SaStep modelling element. This association is inherited from ResourceUsage. Fig. 4 shows the implementation model of this modelling construct in an extract from the MARTE GRM UML view



**Fig. 4.** The list of subUsages, a recursive mechanism to model composite operations.  
Extract of the model of the GRM::ResourceUsage, ancestor of SAM::SaStep

The other way of modelling composed operations is based on the activity diagrams that are internal to the actions (as well as any internally called operation) that are placed in the activities used for representing the flows. The corresponding extracting code is implemented in java and may be seen in the source code file ActivityFunctions.java

## 4 Design models

The second challenge is the definition of the elements in UML+MARTE to be used in early V&V design models in such a way that they can be used for the double purpose of constructing development (implementation) oriented models or even code directly while, at the same time, their respective analysis models may be generated through simple and as much as possible automated model transformation mechanisms

For this purpose the natural candidates in MARTE are the fundamental modelling constructs described in the HLAM (High Level Application Modelling) chapter: RtUnit and PpUnit. Which actually hold the semantics of the basic elements for programming concurrent real-time systems tasks and shared protected objects [11].

The RtUnit modelling element is the basic building block for handling concurrency in the design and analysis of real-time applications. The PpUnit is the modelling element used for specifying mutual exclusion between concurrent units and the adequate protection protocols in the access to passive shared resources, for avoiding unbounded priority inversion.

The key for the usage of these elements is the enabling of simple mechanisms to keep in synch the two specialized views that are elaborated as transformations from the design models built with them: the code generation and the platform configuration on the one side, and the corresponding schedulability or even performance analysis models on the other side. In order to get that semantic alignment we propose a methodology founded in a small number of modelling rules for the usage of RtUnits and PpUnits, and some directions for the generation of the subsequent implementation and analysis models.

In order to accomplish the objective of setting up the basis for an iterative development process, the driving forces for the definition of the methodology have been the easiness to iterate over modelling intents, and a design space exploration strategy to introduce analysis results back in the design constraints.

For the purpose of this methodology we will consider all the requirements as applicable to a generic unit of design called module. A module in this sense represents a fraction of the system. It is to be mapped to the equivalent abstraction/encapsulation entities defined for coping with complexity on the concrete design methodology used by the industrial practitioners in the field targeted. This results natural when considering them as independent subsystems, but it is applicable also to other composition mechanisms like loosely coupled software/hardware components, or physical concurrent units.

The modelling rules to be applied are the basis for the combined purpose of a design & analysis methodology and are complemented with guidelines for specific phases and concrete concerns.

The description of the RtUnits and PpUnits and their precise semantics are made in the domain view of HLAM chapter and the Appendix F of MARTE [3] respectively. The set of rules is worded considering the semantics described in the domain description contained in Appendix F but using the nomenclature of the attributes available in the stereotypes of the corresponding UML representation.

Early V&V assumes that at the time of analysis there are still a number of decisions about aspects like the platforms or specific interface technologies that have not been taken yet. To be able to assess the viability of the system without this information, some default values will be filled in the analysis & design models.

The set of rules for the use of UML with the HLAM modelling elements of MARTE needs to restrict the design space to get models that may be analysed by schedulability analysis with the available techniques. This way it formulates the basis for modelling at any stage of the development process. This initial set of rules is the following:

- 1) Real concurrency is handled by RtUnits at processing resource level, each node by them represented may in turn handle several schedulable resources by means of a regular scheduler.
- 2) Each RtUnit may have up to one schedulable resource on it, and all its behaviours, which may be called from other RtUnits, run under the scheduling

parameters associated to that schedulable resource. In case the RtUnit has no schedulable resource, its behaviours run under the scheduling parameters of the calling RtUnit.

3) All the RtUnits deployed in a processing resource are handled by the same scheduler and use the same (or fully compatible) scheduling policy.

4) Each RtUnit whose isMain attribute is set to true, implies the presence of an execution host where the main service of the RtUnit is deployed.

5) The attribute srPoolPolicy holds the value infiniteWait.

6) ExecKind of PpUnits is ImmediateRemote.

7) All services use the same priority scheme: ImmediateCeiling or PriorityInheritance.

8) The ConcurrencyPolicy of PpUnit is “Guarded”. [The concurrency policy of the kind Concurrent might be enabled in order to have the writer/reader ConcurrencyKind available but this behaviour requires additional capabilities from the analysis techniques so in principle it is discouraged].

9) Behaviours of RtUnits stereotyped as RtServices are those that may be called from others.

Additional rules that apply in specific phases of the development process are:

10) The platform models of the execution hosts are derived from the RtFeatures of RtUnits with the attribute isMain set to true. The basic assumption is that a main is the starting of the full piece of software running on a concrete node. The scheduling policy of the scheduler is derived from the one used for this main. Consistently the range of priorities (in the case in which this is the policy chosen) will be set to be greater than the number of RtUnits (with their isMain attribute to false) with which the main RtUnit has any sort of interaction.

11) The rules for analysis platform models will be refined after practising with the MAST default values (using initially no context switch time for example).

12) The links between services define the steps in the end-to-end-flows.

13) The parameters of the Analysis Context modelling element will be used to define the variations in the analysis due to refinements in the design.

14) Results will be placed back in design models by means of RtFeatures Specifications and the parameters of AnalysisContexts.

15) The iterative nature of the models used for design space exploration will be handled by specializing/using the configuration stereotype, described in the Modal Behaviour section of the CoreElements chapter of MARTE.

The tooling support for enforcing and helping to assess the usage of these rules is part of our ongoing work.

Considering the transformation of design to analysis models the procedure most difficult to automate is the definition of the real-time situations and the extraction of the precise scenarios to take into account. The key aspect to consider in order to extract the necessary execution scenarios is the manner in which the behaviours are expressed.

For passive RtUnits or for those whose behaviours are expressed as activities (by means of activity diagrams), the end-to-end flows may be composed by creating and combining the resource usage (GRM part of MARTE) that represents the operations/services of the objects involved. These are expressed in our design model as classes with the SaStep stereotype (which inherits from ResourceUsage). The

precedence or control flow dependencies between them are expressed as transitions between actions inside the activities. These transitions correspond to the simple precedence relationships among the steps (as described in GQAM model of MARTE) and are the place to express also the latency or jitter constraints, see the `gaLatencyObs` stereotype in Fig. 3. These transitions follow those in the activities used for the automated generation of the Ada code using the tool that assist this methodology [15].

For the `RtUnits` whose behaviours are expressed as state machines, the scenarios may be extracted by assuming the order in which the independent external events that trigger the transitions of the state machine are expected to occur. An `analysisContext` represents a particular real-time situation. For each end-to-end-flow in the `analysisContext` the composition of such ordered list of event-occurrences would need in any case the specification of at least one additional diagram per end-to-end flow. Then it might be worth asking the designer for the composition and description of the high level services conforming the end-to-end flows instead of the ordered list of event-occurrences that triggers the state machine. Additional efforts would have to be required from the designer in case the state machines have concurrent states or involve several active objects.

The easier mechanism to automate the extraction of elements from the design models to be included in the analysis context of the real-time situation to analyse is the use of a collaboration whose parts are the `RtUnits` and `PpUnits` to include.

#### 4.1 Real-Time Design Model of the Basic Ada Structures

Even though the modelling and schedulability analysis methodology presented is language independent and is useful for modelling a wide range of real-time applications, the semantics of the high-level modelling constructs defined is particularly suitable to represent systems conceived and coded in Ada [7]. Similar topics have been described in detailed in our previous work [4] but we now wish to re-visit some of the aspects that are relevant for the discussion herein in the light of the HLAM-MARTE modelling constructs.

**The RT-design-model has the structure of the Ada application:** The `RTUnit` instances may model the real-time behaviour of packages and tagged types, which are the basic structural elements of an Ada architecture:

- Each object describes the real-time model of all the procedures and functions included in a package or Ada class.
- Each object declares all other inner objects (package, protected object, task, etc.) that are relevant to model its real-time behaviour. It also preserves in the model declarations the same visibility and scope of the original Ada structures.

An object only models the code included in the logical structure that it describes. It does not include the models of other packages or modules on which it is dependent.

**The RT-design-model includes the concurrency introduced by Ada tasks:** An active `RTUnit` model an Ada task. Each task component instance has implicitly an aggregated `SchedulableResource`, which is associated with the processor where the component instance is allocated. Synchronization between tasks is only allowed in the invocation of `RTServices` belonging to active `RTUnits`. The model implicitly handles the overhead due to the context switching between tasks.



**The RT-design-model includes the contention in the access to protected objects:**

A PpUnit models the real-time behaviour of an Ada protected object. It implicitly models the mutual exclusion in the execution of the operations declared in its interface, the evaluation of the guarding conditions of its entries, the priority changes implied by the execution of its operations under the priority ceiling locking policy, and also the possible delay while waiting for the guard to become true. Even though the methodology that we propose is not able to model all the possible synchronization schemes that can be coded using protected entries with guarding conditions in Ada, it does allow to describe the usual synchronization patterns that are used in real-time applications. Therefore, protected object-based synchronization mechanisms like handling of hardware interrupts, periodic and asynchronous task activation, waiting for multiple events, or message queues, can be modelled in an accurate and quantitative way. Please refer to [4] for a detailed description of the patterns that may be supported.

The RtService operations involved in the declaration of a PpUnit are implicitly modelled with mutual exclusion between them, by attaching an implicit shared resource to them. Each operation in this component implicitly locks and unlocks the shared resource before and after the operation activities.

**The RT-design-model shall include the real-time communication between Ada distributed partitions:** MARTE does not support explicitly in a particular construct all the data that is necessary to analyze the remote access to the APC (Asynchronous Procedure Call) and RPC (Remote Procedure Call) procedures of a Remote Call Interface (RCI), as described in Annex E of the Ada standard. It is possible to determine whether an invocation is local or remote but additional modelling constructs will be necessary to handle information for the marshalling of messages, their transmission through the network, their management by the local and remote dispatchers and the un-marshalling of messages, in order to be able to manage it automatically by the tools.

## 5 Practical experience

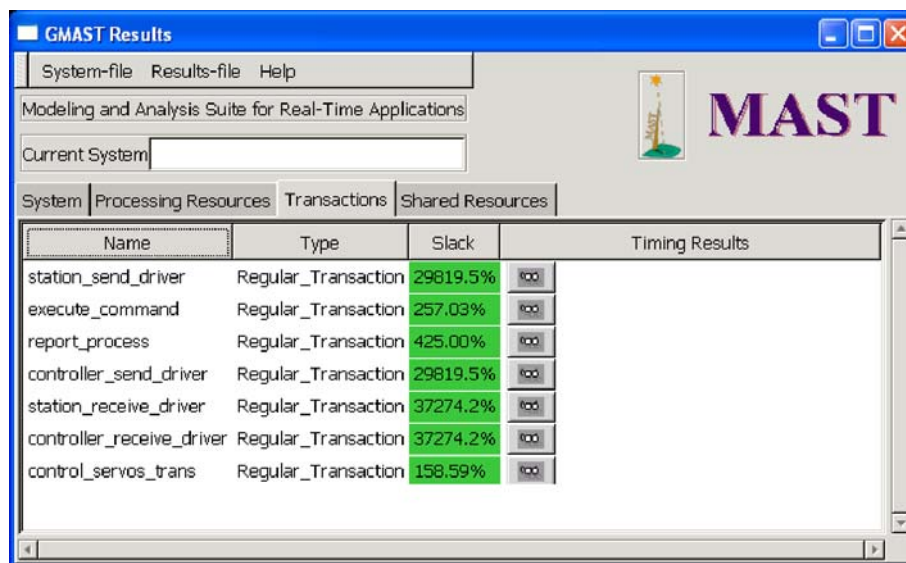
In order to try in practical terms the modelling methodology and particularly the analysis capabilities in it, the implemented tool has been used over an example application that has been already used in previous works. So the analysis model corresponding to the Teleoperated\_Robot example [6] has been introduced in UML using PapyrusUML, and the stereotypes provided by its UML MARTE Profile.

To help the reader to reproduce the modelling and analysis experience, the installation procedure to follow, and a video with usage hints, as well as a workspace for eclipse with all the files used may be downloaded from <http://mast.unican.es/umlmast/marte2mast>.

For the habitual users of MAST the most relevant issues found in the modelling of distributed applications like the already traditional Teleoperated\_robot are related to the lack of the “driver” concept in MARTE. This is solved by inserting the corresponding overhead end-to-end flows (formerly called transactions in MAST). These transactions model the overhead due to the insertion and recovering of

messages into and from the network respectively. For this reason the results in the MAST graphical results viewer show 7 instead of 3 transactions. Fig. 5 shows a snapshot of these results.

The Eclipse plugin includes an option to run the MAST set of analysis tools directly from Eclipse and finally recover the results back into the UML model. These results are annotated in the attributes of the corresponding stereotypes in the UML model. For example the system slack is recovered in the slack attribute of the analysisContext stereotype, and for the response time of each step the attribute respT of the SaStep stereotypes is used. The response time of each End-to-end-Flow is the one stored in the last SaStep of the flow.



The screenshot shows the 'GMAST Results' window with a menu bar (System-file, Results-file, Help) and a title bar. Below the menu is a text field for 'Current System' and a logo for 'MAST'. The main content area has a tabbed interface with 'Transactions' selected. It displays a table with the following data:

| Name                      | Type                | Slack    | Timing Results |
|---------------------------|---------------------|----------|----------------|
| station_send_driver       | Regular_Transaction | 29819.5% | 100            |
| execute_command           | Regular_Transaction | 257.03%  | 100            |
| report_process            | Regular_Transaction | 425.00%  | 100            |
| controller_send_driver    | Regular_Transaction | 29819.5% | 100            |
| station_receive_driver    | Regular_Transaction | 37274.2% | 100            |
| controller_receive_driver | Regular_Transaction | 37274.2% | 100            |
| control_servos_trans      | Regular_Transaction | 158.59%  | 100            |

**Fig. 5.** Results for the schedulability analysis of the Teleoperated\_Robot example

The UML model having all this values may be stored in the same file. In this case each new analysis result value is appended to the ordered list of values of the attribute. To identify the analysis to which the results correspond, the field “mode” of each new value is set to the “mode” field defined in the AnalisisContext at the time of launching the analysis. Alternatively the “updated” version on the UML model may be stored in a new file on disk. This way the original model may be either used again in a different analysis essay, or used to iterate searching in an optimization loop.

## 6 Conclusions and future work

Considering the prospects of the OMG’s UML Profile for MARTE as a modelling standard for analysis tools interoperability, it seems reasonable to look for model based strategies that link it with modelling intensive activities. And a clear semantics

for the High level application modelling is the basis for automating the process of having timing analysis results quickly in the development life cycle.

The extraction of MAST analysis models from the UML+MARTE schedulability analysis specific models is another step in the direction pointed out by this effort and comprises the construction of analysis models from separated composable modelling descriptions using the specific constructs brought by the SAM chapter of MARTE, which is consistent with MAST and the previous efforts in this direction [4].

From the real-time and embedded systems research community perspective, this effort constitutes a step to get the effective exploitation of the capabilities of the available analysis and verification techniques, which despite the efforts in dissemination, have not yet reached an audience large enough to reward the many years of work in the field.

The modelling strategy and tools proposed in this work are just a first step in this direction; a significant work remains to be done in order to have a fully automated process. The validation of the rules and their automation by means of a model validator and the necessary transformations are part of our ongoing.

The re-visit to former work made for the consistent modelling and analysis of Ada applications, in the light of the previous UML formalizations of the MAST model, has been a driving force in the adjustment of the high level application modelling of MARTE for this purpose. Nevertheless some lacks in the standard have been identified whose resolution requires either formal extensions to it or the definition of additional methodological guidelines, in particular for the case of remote procedures invocation.

## References

1. López P., Drake J.M., and Medina J.L., Enabling Model-Driven Schedulability Analysis in the Development of Distributed Component-Based Real-Time Applications. In Proceedings of 35th Euromicro Conference on Software Engineering and Advanced Applications, Component-based Software Engineering Track, Patras, Greece, August 2009, IEEE, ISBN 978-0-7695-3784-9, pp. 109-112.
2. M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake, MAST: Modeling and Analysis Suite for Real-Time Applications, in Proc. of the Euromicro Conference on Real-Time Systems, June 2001.
3. Object Management Group, UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) version 1.0, OMG doc. formal/2009-11-02, 2009
4. J.J.Gutiérrez, J.M.Drake, M.González Harbour, and J.L.Medina. Modeling and Schedulability Analysis in the Development of Real-Time and Distributed Ada Systems. ACM Ada Letters, Vol. XXII, Num. 4. 2002.
5. S. Mazzini, S. Puri and T. Vardanega. An MDE Methodology for the Development of High-Integrity Real-Time Systems. DATE'09 April 20-24, 2009 Nice, France. pp.1154
6. J.L.Medina, M.González Harbour and J.M. Drake, Mast Real-Time: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems, in Proc of the 22nd IEEE Real-Time System Symposium (RTSS 2001),pp 245-256, 2001.
7. T. Taft et al. editors: Ada 2005 Reference Manual. Int. Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1. LNCS 43-48, Springer-Verlag 2006.
8. S.Demathieu, and L. Rioux. MARTE to RapidRMA. Thales Report/Technical Document number 61565273 305 6. <http://www.omgwiki.org/marte/node/31>.

- 9 Eric Maes and Nicolas Vienne. MARTE to Cheddar Transformation using ATL. Thales Report/Technical Documents number 61565546-179 and 61565546 108. [http://beru.univ-brest.fr/~singhoff/cheddar/contribs/examples\\_of\\_use/thales\\_rt/MARTE2CheddarTransformationRules.pdf](http://beru.univ-brest.fr/~singhoff/cheddar/contribs/examples_of_use/thales_rt/MARTE2CheddarTransformationRules.pdf)
- 10 Object Management Group. Precise Semantics of UML Composite Structures. Response to Request for Proposal ad/11-12-07. Initial submission. OMG Document Number: ad/2012-11-05
- 11 Alan Burns and Andy Wellings. Concurrent and Real-Time Programming in Ada. ISBN 978-0-521-86697-2. Published by Cambridge University Press , 2007
- 12 Object Management Group. Unified Modeling Language version 2.4.1, OMG document formal/2011-08-06, 2011
- 13 Object Management Group. OMG Systems Modeling Language (OMG SysML™) Version 1.3. OMG document formal/2012-06-01
- 14 Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1 - January 2011. OMG Document Number: formal/2011-01-01.
- 15 <http://mast.unican.es/umlmast/uml2ada>

# Operating system support for Ada timers

Mario Aldea Rivas and Michael González Harbour

University of Cantabria

**Abstract.** <sup>1</sup> The Ada language, in its revision of the year 2005, added support for three different kind of timers very useful for real-time applications: the timing events, the execution time timers and the group budgets. In most of the platforms where Ada applications are executed, the implementation of these timers in the Ada run-time library requires support in the underlying operating system. Most of the real-time operating systems do not provide primitives to allow an efficient implementation of Ada timers. This is also the case of the operating systems that follow the POSIX standard. In this paper we present the “timed handlers”, a new operating system primitive that allows to implement the Ada timers in a efficient way. A POSIX-like interface is provided in order to make accessible the timed handlers to the Ada run-time library and to C/C++ applications.

## 1 Introduction

Real-time systems control and/or monitor dynamic environments that evolves with the passage of time. A usual requirement of any real-time system is that the interactions with the environment have to be performed at very specific moments in time. A typical example of this requirement would be a control system where the input to the actuators must be updated at a specific rate with the least jitter as possible.

The requirement formerly stated can be fulfilled, up to some extent, using high priority tasks and suspension operations. However, for some applications the overhead and complexity of this method is excessive and a simpler mechanism would be desirable. With this purpose in mind the Ada language [3], in its revision of the year 2005 [11], defined the “timing events”, a mechanism that allows the execution of user’s code at a specific point in time without the necessity of using an extra high priority task.

Another different requirement of the real-time systems is the necessity of measuring and monitoring the execution time of the tasks in the system. In hard real-time systems it is essential to detect when the estimated worst-case execution time (WCET) of a task is exceeded and perform a corrective action as soon as that happens.

The control of the execution time of the tasks is also a key requirement for implementing some real-time scheduling policies which need to perform scheduling actions when a certain amount of execution time has been consumed. Examples of this kind of policies are aperiodic servers such as the sporadic server policy defined in the POSIX standard [2] or the constant bandwidth server (CBS) [5].

The construction of complex real-time systems could require the integration of independently developed multi-task components. The execution-time of each component

---

<sup>1</sup> This work has been funded in part by the Spanish Government and FEDER funds under grant number TIN2011-28567-C03-02 (HI-PARTES).

must be bounded so that it cannot interfere with other components by using up more resources than it should. In this context, the aforementioned aperiodic servers are used to bound the consumed execution-time of the group of tasks been part of each component. The implementation of these multi-task aperiodic servers requires from the underlying implementation to support the measurement of the execution times of groups of tasks, and the handling of potential budget overruns, in a way similar to what is usually done for individual tasks.

In summary, there are a number of real-time situations that require the execution of a piece of user's code at a very specific point in time. This point in time can be referred to the "real time clock" or to the execution time clock of some task or some group of tasks. In the Ada language, these situations are handled using three different kinds of timers: "Timing events" ([3].D.15), "Execution time timers" ([3].D.14.1) and "Group execution time timers" ([3].D.14.2).

In previous papers [7] [6] we describe the implementation of these Ada timers from the Ada language point of view. In this paper we focus on the description of the operating system services, the "Timed Handlers" and the "Thread Group Clocks", that provide support for these three kinds of timers. Both services, Timed Handlers and Thread Group Clocks, have been implemented in our operating system MaRTE OS [10] [4], and a POSIX-like interface has been defined in order to make these services accessible to the Ada run-time system and to the C/C++ applications that want to make direct use of these functionalities.

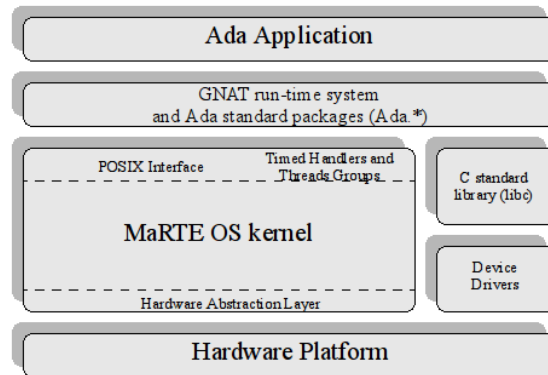
The paper is organized as follows. Section 2 provides an overview of the MaRTE OS architecture. Section 3 makes a short description of the Ada timers. Section 4 describes the implementation and interface of the timed handlers while Section 5 does the same for the thread group clocks. Finally, Section 6 gives our conclusions.

## 2 MaRTE OS overview

MaRTE OS is a real-time operating system (RTOS) that follows the POSIX.13 [1] minimum real-time system profile. This profile includes the basic support for small real-time multi-threaded applications: threads, fixed priority scheduling policies, mutexes, condition variables, timers, etc. MaRTE OS is available for the ix86 architecture as a bare machine, and it can also be configured as a POSIX-thread library for GNU/Linux. It is mostly written in Ada but it provides a standard POSIX-C interface that can be used to build concurrent C/C++ applications.

MaRTE OS is integrated with the GNAT compiler and run-time library to provide support for Ada tasking. The Linux version of the GNAT run-time library has been adapted to run on top of the POSIX interface provided by MaRTE OS.

Figure 1 shows the simplified layered architecture of MaRTE OS. Advanced Ada constructions, as the concurrency or the timers, are provided to Ada applications by the run-time system and the standard Ada packages which, in turn, relies on the services implemented in the kernel of MaRTE OS. These services are accessible through the POSIX interface and, in the case of the Timed Handlers and the Thread Group Clocks, through extensions of that interface presented respectively in sections 4.1, 5.1 and 5.2.



**Fig. 1.** Layered architecture of MaRTE OS.

### 3 Ada timers

#### 3.1 Timing Events

Timing Events are a solution for real-time applications that require executing actions at very specific points in time. Those actions are typically very simple, usually a few lines of code, but deviations from the ideal execution time will lead to a serious degradation of the quality of service of the application or even to an error with disastrous consequences for the system under control.

Burns and Wellings proposed in [8] what they called the “Timing events”. This new service was included in the Ada 2005 revision of the language as the new language-defined package `Ada.Real_Time.Timing_Events`.

```
package Ada.Real_Time.Timing_Events is
  type Timing_Event is tagged limited private;
  type Timing_Event_Handler
    is access protected procedure (Event : in out Timing_Event);
  procedure Set_Handler (Event : in out Timing_Event;
                        At_Time : in Time;
                        Handler : in Timing_Event_Handler);
  procedure Set_Handler (Event : in out Timing_Event;
                        In_Time : in Time_Span;
                        Handler : in Timing_Event_Handler);
  function Current_Handler (Event : Timing_Event)
    return Timing_Event_Handler;
  procedure Cancel_Handler (Event : in out Timing_Event;
                           Cancelled : out Boolean);
  function Time_Of_Event (Event : Timing_Event) return Time;
private
  ... -- not specified by the language
end Ada.Real_Time.Timing_Events;
```

A Timing Event is a software object that has an expiration time and an associated handler (a protected procedure) that will be executed by the system when the timing event expires. The handler is executed directly by the hardware timer interrupt service routine (ISR), consequently, it is executed as soon as possible after the expiration time and no extra server task is required.

It is important to notice that, since the handler is executed by the ISR, the protected object of the handler should include some mechanism to achieve mutual exclusion between user's tasks and ISRs.

### 3.2 Execution Time Timers

Schedulability analysis is based on the assumption that the execution time of each task can be accurately estimated. This measurement is always difficult, because, with effects like cache misses, pipelined and superscalar processor architectures, the execution time is highly unpredictable. Run-time monitoring of processor usage permits detecting and responding to wrong estimations in a controlled manner.

Ada defines a timing mechanism which allows creating timers that are triggered when the execution time of a task reaches a given value, providing the means whereby action can be taken when this budget expires. Execution time timers have been included in the standard as a new language-defined package (`Ada.Execution_Time.Timers`).

```
with System;
package Ada.Execution_Time.Timers is
  type Timer (T : not null access constant
              Ada.Task_Identification.Task_Id) is
    tagged limited private;
  type Timer_Handler is
    access protected procedure (TM : in out Timer);
  Min_Handler_Ceiling : constant System.Any_Priority :=
    implementation-defined;
  procedure Set_Handler (TM      : in out Timer;
                        In_Time  : in Time_Span;
                        Handler  : in Timer_Handler);
  procedure Set_Handler (TM      : in out Timer;
                        At_Time  : in CPU_Time;
                        Handler  : in Timer_Handler);
  function Current_Handler (TM : Timer) return Timer_Handler;
  procedure Cancel_Handler (TM      : in out Timer;
                           Cancelled : out Boolean);
  function Time_Remaining (TM : Timer) return Time_Span;
  Timer_Resource_Error : exception;
private
  ... -- not specified by the language
end Ada.Execution_Time.Timers;
```

Execution Time Timers are very similar to Timing Events, both have an expiration time and a protected handler procedure. The main difference is that the expiration time



of an execution time timer is based on the execution clock of a task. This task is specified by the discriminant, T, of the Timer type.

Another difference between Execution Time Timers and Timing Events is that, in the case of the timers, the Ada Reference Manual is silent about the context where the timer handler must be executed. As it will be discussed in Section 4 we have chosen to use the same mechanism for all the Ada timers and execute all their handlers directly by the hardware timer ISR.

### 3.3 Execution Time Budgets for Groups of Tasks

In real-time applications is often required to bound the execution time of a group of tasks, so that they cannot interfere with other activities in other protection boundaries by using up more resources than they should. These protection boundaries are usually implemented using the so called “execution-time servers”. Some examples of these servers are the deferrable server, the sporadic server or the constant bandwidth server. Implementing these servers for a group of tasks requires the joint measurement of the execution times of the tasks in the group and the handling of potential budget overruns, in a way similar to what is done for individual tasks.

Following this general requirement, the Ada standard defines the package `Ada.Execution_Time.Group_Budgets` with all the functionality required to implement execution-time servers [9]. In this aspect, the Ada standard is now a step forward in relation to the real-time extensions to POSIX, which still has no such service.

```
with System;
with System.Multiprocessors;
package Ada.Execution_Time.Group_Budgets is
  type Group_Budget(CPU : System.Multiprocessors.CPU :=
                    System.Multiprocessors.CPU'First)
    is tagged limited private;
  type Group_Budget_Handler is access
    protected procedure (GB : in out Group_Budget);
  type Task_Array is array (Positive range <>) of
    Ada.Task_Identification.Task_Id;
  Min_Handler_Ceiling : constant System.Any_Priority :=
    implementation-defined;
  procedure Add_Task (GB : in out Group_Budget;
                     T : in Ada.Task_Identification.Task_Id);
  procedure Remove_Task (GB: in out Group_Budget;
                        T : in Ada.Task_Identification.Task_Id);
  function Is_Member (GB : Group_Budget;
                     T : Ada.Task_Identification.Task_Id) return Boolean;
  function Is_A_Group_Member
    (T : Ada.Task_Identification.Task_Id) return Boolean;
  function Members (GB : Group_Budget) return Task_Array;
  procedure Replenish (GB : in out Group_Budget; To : in Time_Span);
  procedure Add (GB : in out Group_Budget; Interval : in Time_Span);
```

```

function Budget_Has_Expired (GB : Group_Budget) return Boolean;
function Budget_Remaining (GB : Group_Budget) return Time_Span;
procedure Set_Handler (GB      : in out Group_Budget;
                      Handler : in Group_Budget_Handler);
function Current_Handler (GB : Group_Budget)
    return Group_Budget_Handler;
procedure Cancel_Handler (GB      : in out Group_Budget;
                          Cancelled : out Boolean);

Group_Budget_Error : exception;
private
    -- not specified by the language
end Ada.Execution_Time.Group_Budgets;

```

## 4 Timed Handlers

POSIX [2] standard does not provides any service equivalent to the Ada timers. At the first glance it could be considered that a valid alternative would be the use of timers and signal handlers: a POSIX timer generates a signal that leads to the execution of a signal handler that executes the user's code for the action.

However, this approach is not appropriate for real-time applications. Firstly, the generation and delivery of a signal requires quite complex operations in the operating system kernel, and secondly, POSIX standard is silent about the context nor priority of the entity in charge of executing the signal handlers. Consequently, the signal handler mechanism is not appropriate in order to execute a particular piece of code at a very precise point in time since its execution latency can be very long or even unbounded depending on the particular operating system implementation.

Another alternative to execute actions at a specific time could consist in the use of a high priority server task that executes a loop in which it blocks waiting for the time to execute the action.

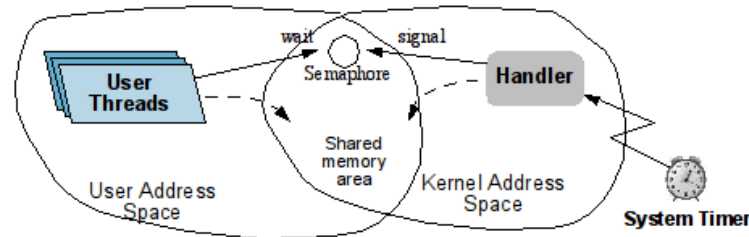
This approach also leads to overheads from both the performance and the memory usage points of view. An extra task requires quite large structures in memory as its “task control block” and its stack. From the performance point of view, the execution of the simple action involves the expiration of the hardware timer, the subsequent invocation of the ISR (that activates the high priority server task), a context switch, the execution of the action, the suspension of the server task and another context switch.

In this paper we present a new operating system service, the “Timed Handlers”, that provide the base support to implement the Ada timers described in Section 3. Timed Handlers allows executing user code in the hardware timer ISR context at the desired time and they are simpler and much more efficient than the aforementioned alternatives.

The expiration time of a timed handler can be based on any of the clocks defined by the POSIX standard, including thread execution-time clocks. Timed handlers can also be based on the “Thread Group Clocks” used to implement the Ada execution-time group budgets that are described in Section 5.

Our Timed Handlers proposal is targeted to be implemented in any kind of RTOS, from the smallest one with a simple and plain memory map to a complex RTOS with

separate address spaces. When used in a complex RTOS, a shared memory area is provided to share data between the handler and the user threads (Figure 2).



**Fig. 2.** Timed Handlers in a RTOS with separate address spaces

#### 4.1 Timed Handlers API

A POSIX-like interface has been defined to allow C/C++ applications (and the GNAT run-time library) to access the timed handlers functionality implemented in the MaRTE OS kernel. In this proposed interface, a timed handler is represented by an object of type `mar_te_timed_handler_t`. When a timed handler expires, the operating system executes its associated handler function. In our interface, a handler function has the following prototype:

```
void handler(void *area, mar_te_timed_handler_t *th)
```

The `area` argument points to a memory area shared by the handler and the rest of the application. The `th` argument points to the timed handler that fires the execution of the handler. This last argument is useful when the same handler function is associated with several timed handlers.

A timed handler can be initialized or destroyed with functions:

```
int mar_te_timed_handler_init(mar_te_timed_handler_t *th,
    clockid_t clock_id,
    void (*handler) (void *area,
        mar_te_timed_handler_t *th),
    const void * area,
    size_t size);

int mar_te_timed_handler_destroy(mar_te_timed_handler_t *th);
```

When a timed handler is initialized (using `mar_te_timed_handler_init()`) it is specified the clock to be used as the timing base. The clock can be any of the clocks defined by the POSIX standard (including thread execution-time clocks) or the “Thread

Group Clocks” described in Section 5. POSIX requires the type `clockid_t` to be defined as an arithmetic type, and therefore clock ids are implemented using a unsigned number of 32 bits. The value stored in a clock id can have different interpretations:

- Special values for the regular calendar-time clock `CLOCK_REALTIME`, the execution time clock of the current thread `CLOCK_THREAD_CPUTIME_ID`, and the monotonic clock `CLOCK_MONOTONIC`.
- A pointer to a thread control block when the clock is a thread execution-time clock of a particular thread.
- A pointer to a thread set when it is a thread group clock (Section 5).

The other parameters of the function `marte_timed_handler_init()` are the memory area of `size` bytes to allow information sharing between the handler function and the rest of the application and the handler function to be executed every time the timed handler expires.

The expiration time (`ts`) of a timed handler (`th`) is set using the function:

```
int marte_timed_handler_set(marte_timed_handler_t *th,
    int options,
    const struct timespec *ts);
```

Flags `TIMER_ABSTIME` or `PERIODIC_HANDLER` can be set in the options argument. If `TIMER_ABSTIME` is set, the `ts` argument shall be interpreted as an absolute time. If `PERIODIC_HANDLER` is set, the timed handler will expire periodically with a period equal to the value of `ts` argument. If none of these flags are set, the `ts` argument shall be interpret as a relative time and the timed event will be a “one shot” event.

Any particular timed handler can be enabled or disabled using functions:

```
int marte_timed_handler_disable(marte_timed_handler_t *th);
int marte_timed_handler_enable(marte_timed_handler_t *th);
```

If a timed handler expires while it is disabled, execution of its associated handler function will be delayed until the timed handler is re-enabled. Using this pair of functions, an application thread can create a non-preemptive section with a timed handler function in order to access shared data. This is the mechanism used by the Ada run-time library to implement the protected objects of the timer’s handlers.

It is possible to enable or disable all timed handlers in a process using functions:

```
int marte_timed_handler_global_enable ();
int marte_timed_handler_global_disable ();
```

These functions do not affect the individual enabled/disabled status of every timed handler in the process but only change a global flag, this implies that using `marte_timed_handler_global_enable()` does not enable any timed handler that was individually disabled using `marte_timed_handler_disable()`. If an enabled timed handler expires while handlers are globally disabled, execution of its associated handler will be delayed until timed handlers are globally re-enabled using `marte_timed_handler_global_enable()`. These functions are intended to be a very efficient way of disabling all the timed events in a process for situation in which several timed events cooperate and share data between them and with the application.

## 4.2 Implementation in MaRTE OS

In MaRTE OS, all the timed events are represented as extensions of the base type `Timed_Event` from which they inherits the expiration time field. There are many different kinds of timed events: timed task activation, time-out of a timed blocking operation, replenishment of a sporadic server, expiration of a POSIX timer, etc. Following this approach, timed handlers are also implemented as an extension of the `Timed_Event` type. The most relevant fields added to the base type by the Timed Handlers are `Enabled`, a flag to indicate if the execution of the handler is currently enabled or not, and `Handler`, an access to the user's function to be executed when the timed handler expires.

All the pending timed events based on the real-time clock are queued in the “Timed Events Queue” ordered according to their expiration time (most urgent first). Besides that global queue, every task has associated an “Execution-Time Events Queue” where the timed events based on its execution-time clock are queued in order. Every time the hardware timer has to be programmed the events at the heads of the Timed Events Queue and of the Execution-Time Events Queue of the running task are compared. The hardware timer is programmed to generate an interrupt at the smallest expiration time of the two events.

When the hardware timer generates an interrupt, the MaRTE OS timer ISR extracts the most urgent timed event from its queue. In the case it represents an enabled timed handler the associated handler function is executed immediately in ISR's context.

Disabling a particular timed handler only involves changing the `Enabled` flag in the timed handler structure. If a timed handler expires while disabled, the timer ISR marks it as pending. When a pending timed handler is re-enabled, the associated handler function is executed immediately with the hardware interrupts disabled.

In the case an enabled timed handler expires while timed handlers are globally disabled, the timer ISR stores the timed handler in the “Pending Handlers List”. Later, when the global enable operation is called, the associated handler function of every timed handler in the list is executed with interrupts disabled and the list is emptied.

In conclusion, implementing timed handlers in MaRTE OS has been relatively easy. Just to give a coarse idea of the implementation difficulty, it can be noticed that the timed handlers implementation uses around 240 semicolons, while timers implementation uses 410 and signals around 700.

## 4.3 Performance Metrics

Table 1 shows some performance metrics for the implementation of the timed handlers in our operating system MaRTE OS. We have chosen a typical scheduling decision: raising the priority of a thread at a particular point in time. As a consequence of this priority change the promoted thread preempts a medium priority thread that was executing at that moment. Two different alternatives have been compared:

- The thread priority is raised from a timed handler.
- The thread priority is raised from an auxiliary high priority thread. This auxiliary thread is waked up by a signal generated by a timer at the desired time.

Third row in Table 1 shows the total time taken by the context switch between the medium priority thread and the just promoted thread with both alternatives. As it can be observed using the timed handler saves 0.8ms out of 1.9ms (the 42%).

| Timed handler                     |                | Timer and auxiliary thread              |                |
|-----------------------------------|----------------|---|----------------|
| Description                       | Time( $\mu$ s) | Description                             | Time( $\mu$ s) |
| Medium priority thread to handler | 0.4            | Medium prio. thread to auxiliary thread | 1.1            |
| Handler to promoted thread        | 0.7            | Auxiliary thread to promoted thread     | 0.8            |
| Total                             | <b>1.1</b>     | Total                                   | <b>1.9</b>     |

**Table 1.** Timed handler vs. Timer and thread (on a 3.4GHz Pentium IV)

As stated before, handler functions are executed in interrupt context. In many operating systems (as in our MaRTE OS) that will imply interrupts are disabled during the full execution of the handler. Although timed handlers are intended to perform very short actions, this fact could produce an increment in the interrupt latency.

This effect does not happen in our implementation even for the relatively long duration action of raising the priority of a thread, as can be observed in the table. When using timed handlers, the longest interval with interrupts disabled corresponds to the first value in the last row (1.1ms). It must be compared by the longest interval with interrupts disabled when using the auxiliary thread, that value can be found in the last column of the third row and it is 1.1ms too. Therefore, in our implementation, using timed handlers do not increment interrupt latency provided that reasonably short handler functions are used.

## 5 Thread Group Clocks

The Thread Group Clocks [6] are a special kind of execution-time clocks that measure the execution-time consumed by a group of threads. They have been implemented in MaRTE OS as an special kind of execution-time clocks. As the standard mono-thread execution-time clocks, the group clocks can be used as base clock for POSIX timers and also for the Timed Handlers mechanism described in the previous section.

Although the Thread Group Clocks can be used with the POSIX timers, we have chosen to implement the Ada Group Budgets based on the Timed Handlers mechanism for the same efficiency reasons that were stated for the Timing Events and the Execution Time Timers implementation in the previous section.

### 5.1 Threads Sets

Before creating the execution time clocks for thread groups, it is necessary to specify a mechanism to represent the groups themselves.

The Ada Group Budgets unify the concepts of group of tasks and timer in the same software object. For the OS system support, we have chosen to create an independent object that represents a group of threads. In this way, we will be able to address future

extensions that require handling groups of threads using these same objects. Examples of such new services might be related to the requirement for supporting hierarchical scheduling, for instance to suspend or resume a group of threads atomically.

A restriction has been made so that a thread can belong to only one thread set. This restriction is also made in the Ada standard, and its rationale is that in the hierarchical scheduling environment for which thread groups are useful, threads only belong to one specific scheduling class, and therefore to one specific set. This restriction allows a more efficient implementation, because at each context switch only needs to be updated the time consumed by one thread group.

Threads can be added/removed to/from a thread set dynamically. Every thread has a pointer in its thread control block (TCB) to the set it belongs to. This field is null if the thread does not belong to any thread set.

The C language API to manage thread sets from the application level is the following:

```
// create/destroy a thread set
int marte_threadset_create(marte_threadset_id_t *set_id);
int marte_threadset_destroy(marte_threadset_id_t set_id);

// empty an existing thread set
int marte_threadset_empty(marte_threadset_id_t set_id);

// add/delete threads in a set and test membership
int marte_threadset_add(marte_threadset_id_t set_id,
                       pthread_t thread_id);
int marte_threadset_del(marte_threadset_id_t set_id,
                       pthread_t thread_id);
int marte_threadset_ismember(marte_threadset_id_t set_id,
                             pthread_t thread_id);
int marte_threadset_getset(marte_threadset_id_t *set_id,
                           pthread_t thread_id);

// iterate over the threads in a set
int marte_threadset_first(marte_threadset_id_t set_id,
                          pthread_t *thread_id);
int marte_threadset_next(marte_threadset_id_t set_id,
                          pthread_t *thread_id);
int marte_threadset_hasnext(marte_threadset_id_t set_id)
```

## 5.2 Execution time clocks for thread groups

To implement execution time clocks for groups of threads we have added the following information to the object that represents a thread set:

- Consumed\_Time: sum of the execution-time consumed for all the threads in the set. Every time a thread of a given set leaves the CPU, the time consumed by this thread

since its last activation is added to this field of the set, even if there is no timed event associated with it, because the value of the execution-time clock may be read at any time by the application.

- `Group_Timed_Event`: A reference to the timed handler associated with this thread set. A set can be associated with at most one timed handler.

The API to obtain an execution-time clock from a thread set is:

```
int marte_getgroupcpuclockid(marte_threadset_id_t set_id,
                             clockid_t *clock_id);
```

The returned id represents a clock that can be read and set through the standard POSIX API for clocks, i.e., using functions `clock_gettime`, `clock_settime`, etc. They can also be used as the base for POSIX timers and MaRTE OS timed events as any other clock defined in the system. They can not however be used as the base for the `clock_nanosleep` operation, as is also the case with the single-thread execution-time clocks. POSIX leaves this behaviour as unspecified and Ada does not define execution time as a type that can be used in the equivalent delay statements.

### 5.3 Timed handlers based on a thread group clock

As it was described in Section 4.2, in MaRTE OS all the timed events (including the timed handlers) are extensions of the base type `Timed_Event`. Timed handlers based on a thread group clock are a special kind of timed handlers. Following the same extension pattern, they extend the `Timed_Handler` type adding the following fields:

- `Group_Expiration_Time`: The event will expire when the clock of the thread set associated with the event reaches this value.
- `Task_Where_Queued`: A reference to the thread that currently has queued the event in its Execution-Time Events Queue.

Timed events based on group clocks are special execution-time events that “jump” between the Execution-Time Events Queues of the threads in the group. Each time the system dispatches a thread included in a set that has a timed handler associated, the following actions are performed:

1. The expiration time of the timed handler is set to the value of the execution-time clock of the thread that would cause, if reached, the group timer to expire.
2. The timed handler is placed in the Execution-Time Events Queue of the thread. It will be placed in order according to its expiration time.
3. The `Task_Where_Queued` field of the timed handler is updated to point to the thread where it has just been queued.
4. The hardware timer is reprogrammed to generate an interrupt at the closest expiration time of the events at the heads of the global Timed Events Queue and of the Execution-Time Events Queue of the thread.

When a group timed handler is the event with closest expiration time in the system, the hardware timer is programmed to expire at the time at which the execution-time of the running thread would cause the group timed handler to expire. In the case that a context switch changes the running thread, the former actions would be executed again and, as a consequence of that, the hardware timer would be reprogrammed.



## 5.4 Performance metrics

Dequeue and enqueue operations in the Execution-Time Events Queue of a thread are very fast, because the number of execution-time events associated to a task is typically two at most: a “standard” execution-time event and a “group event”.

Measurements performed in MaRTE OS has shown that execution-time accounting for threads increments the context switch time by less than 5%. Group execution-time accounting increments the context switch time by another 4%, representing a total of 9% increment with respect to a system with no execution-time accounting.

Apart from the overhead of the execution-time accounting, the complexity of the timed handlers based on group clocks is the same than for the rest of timed handlers. So, results in Table 1 are also applicable for this kind of time handlers.

## 6 Conclusions

Two new operating system services has been presented in the paper: the “Timed Handlers” and the “Thread Group Clocks”. These services provide the OS functionality required to implement the tree kinds of timers defined in the Real-time Systems Annex of the Ada standard: the “Timing Events”, the “Execution Time Timers” and the “Group Execution Time Timers”.

The “Timed Handlers” are a new OS service that allows executing user code at a very specific point in time. The handlers are executed directly by the hardware timer ISR and, in consequence, they are executed with the smallest deviation as possible respect to their intended time. The use of timed handlers is simpler and more efficient than the alternatives available in traditional RTOSs based on the use of timers, signals and high priority server tasks.

The “Thread Group Clocks” are a special kind of execution-time clocks that measure the execution-time consumed by a group of threads. This innovative service, not supported by the currently available RTOSs, is used together with the Timed Handlers to implement the Ada Group Execution Time Timers.

A POSIX-like interface has been provided in order to make accessible the OS services proposed to the Ada run-time library and to C/C++ applications.

## References

1. *POSIX.13-2003*. Information Technology - Standardized Application Environment Profile-POSIX Realtime and Embedded Application Support (AEP). IEEE Computer Society, 2003.
2. *POSIX.1-2008*. Information technology—Portable Operating System Interface (POSIX). IEEE Computer Society, 2008.
3. *Ada Reference Manual. Language and Standard Libraries - International Standard ISO/IEC 8652/2012 (E) with Technical Corrigendum 1 and Amendment 1*. ISO, 2013.
4. MaRTE OS website. <http://marte.unican.es/>, February 2013.
5. L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 4–13, dec 1998.
6. Mario Aldea Rivas and Michael González Harbour. Operating system support for execution time budgets for thread groups. *Ada Lett.*, XXVII(2):67–71, April 2007.

7. Mario Aldea Rivas and José F. Ruiz. Implementation of new ada 2005 real-time services in marte os and gnat. In *Proceedings of the 12th international conference on Reliable software technologies*, Ada-Europe'07, pages 29–40, Berlin, Heidelberg, 2007. Springer-Verlag.
8. A. Burns and A. J. Wellings. Accessing delay queues. *Ada Lett.*, XXII(4):72–76, April 2002.
9. A. Burns and A.J. Wellings. Programming execution-time servers in ada 2005. In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, pages 47 –56, dec. 2006.
10. Mario Aldea Rivas. Michael gonzález harbour: Marte os: An ada kernel for real-time embedded applications. In Dirk Craeynest and Alfred Strohmeier, editors, *Ada-Europe*, volume 2043 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2001.
11. S. Tucker Taft, Robert A. Duff, Randall Brukardt, Erhard Plödereder, and Pascal Leroy. *Ada 2005 Reference Manual. Language and Standard Libraries - International Standard ISO/IEC 8652/1995 (E) with Technical Corrigendum 1 and Amendment 1*, volume 4348 of *Lecture Notes in Computer Science*. Springer, 2006.

# On the Processor Utilisation Bound of the C=D Scheduling Algorithm

J. Augusto Santos-Jr<sup>\*</sup>, George Lima<sup>\*</sup>, and Konstantinos Bletsas<sup>+</sup>

<sup>\*</sup>Federal University of Bahia, Salvador, Brazil

<sup>+</sup>CISTER/INESC-TEC Research Centre, ISEP, Porto, Portugal

**Abstract.** Under semi-partitioned multiprocessor scheduling some (or most) tasks are partitioned to the available processors while the rest may migrate between different processors, under a carefully managed scheme. One of the best performing and practical to implement EDF-based semi-partitioned algorithms is C=D splitting. Under this algorithm, each migrating task always executes at the highest-priority on all but one of the processors that it uses. This arrangement allows for efficient processor utilisation in general, however no utilisation bound had been published so far for this algorithm. We address this situation by deriving the utilisation bound of  $\frac{13}{18}$  for a variant of C=D with the following constraint: at most one migrating task may utilise each processor. We also draw additional conclusions for the utilisation bound attainable under a C=D task splitting scheme in the general case.

## 1 Introduction

Consider the problem of scheduling  $n$  sporadic tasks over  $m$  identical processors to meet deadlines. Traditionally, scheduling algorithms for this problem were classified as *partitioned* or *global*. Partitioning divides the task set into disjoint subsets each of which is assigned to a respective processor and scheduled there under some uniprocessor algorithm such as Earliest-Deadline-First (EDF) or Fixed Priority Scheduling (FPS). Global scheduling on the other hand, maintains a single run queue for all tasks and, at any given instant, the  $m$  highest priority tasks execute, each on one of the  $m$  processors. Hence, under global scheduling, tasks may migrate, even halfway through the execution. Partitioning offers simplicity and re-use of techniques from uniprocessor scheduling. The family of global scheduling algorithms dominates partitioning in terms of achievable system utilisation but at the cost of higher scheduling overheads and scalability issues.

The novel *semi-partitioned* scheduling paradigm aims to combine the best aspects of partitioning (efficient implementation and low dispatching overheads) with those of global scheduling (efficient utilisation of available processing capacity). Various semi-partitioned scheduling schemes have been devised. Some are based on fixed-priority scheduling [18] [17] [13] [19] but most are based on EDF [17] [21] [1] [11] [12] [3] [2] [6] [16] [7][8]. The work on semi-partitioning has also influenced the design of novel, unconventional global scheduling schemes [24] [20] [22] [23], especially with regard to the degree of sophistication in the management of task migrations.

The C=D algorithm is an EDF-based semi-partitioned approach which has been shown to have high average-case performance. Migrating tasks are split into pieces; all

but one piece execute as a zero-laxity subtask. Both migrating and non-migrating tasks are scheduled by local EDF, which means that no special implementation requirement is enforced. These characteristics make C=D a very good algorithm from a practical perspective. From a theoretical point of view, though, the least utilisation bound that can be ensured by C=D is not known up to now. In this paper we analyse this problem and provide useful insights into the theoretical aspects of C=D by looking into a new and more restrictive variant of C=D, named Clustered C=D. We provide then a general characterization of the C=D approach in terms of utilization bound; highlight some aspects regarding the task splitting scheme to be used; and establish, as our main result, the fact that the least upper bound on processor utilisation that can be guaranteed by the C=D approach is higher than  $\frac{13}{18} = 72.2\%$  but cannot exceed 75%.

## 2 About the C=D scheduling algorithm

**Notation** We denote by  $\Gamma$  the entire set  $\{\tau_1, \dots, \tau_n\}$  to be scheduled and by  $\Gamma_p$  the set of non-migrating tasks assigned to processor  $P_p$ . The term  $U(\Gamma_p)$  denotes the utilisation of the task subset  $\Gamma_p$  whereas the utilisation of an individual task  $\tau_i$  is denoted by  $u_i$ .

### 2.1 Outline of the C=D approach

C=D splitting [9] is a well-performing EDF-based semi-partitioned scheduling approach. Its main characteristic is that each migrating task always executes at the highest-priority on all but one of the processors that it utilises; by comparison, other tasks are scheduled as background workload under EDF on their respective processors. This characteristic also accounts for the name of the approach, since one way of ensuring that a task executes at the highest priority without straying from the semantics of a pure EDF policy is to set its deadline ( $D$ ) equal to its execution requirement ( $C$ ). Therefore, each migrating task is modelled as such a zero-laxity ( $C = D$ ) task on the first  $k - 1$  of the  $k$  processors that it is assigned to. The fact that the two scheduling arrangements (assigning to a task the highest priority vs modelling the same task as zero-laxity) are equivalent is formally proven by the following lemma:

**Lemma 1.** *Let a system  $S$  consist of a task  $\tau_0$  scheduled at the highest priority on some processor  $P_p$ , together with a set  $\Gamma_p$  of implicit-deadline background tasks, which are scheduled under EDF. Let another system  $S'$  consist of the same set of tasks  $\tau_0 \cup \Gamma_p$  scheduled under pure EDF, wherein  $\tau_0$  is assigned a relative deadline  $D_0 = C_0$ . Then  $S$  is schedulable under its respective scheduling policy if and only if  $S'$  is schedulable under EDF.*

*Proof.* Consider the two systems  $S$  and  $S'$  and the set of all possible legal task arrival patterns. For each such arrival pattern there exist two complementary cases:

- **Case 1:** The respective schedules produced are identical. In this case, either both  $S$  and  $S'$  meet their deadlines or both miss deadlines.

- **Case 2:** The respective schedules differ. Then, consider the first time instant in  $[0, \infty)$  where the two schedules diverge. This can only occur when  $S'$  is executing a job by a task  $\tau_j \in \Gamma_p$ , whose absolute deadline is earlier than that of an active job by  $\tau_0$ ; in comparison,  $S$  would be executing  $\tau_0$  on the same instant. It then follows that  $S$  will miss a deadline (by  $\tau_j$ ) and  $S'$  will also miss a deadline (by  $\tau_0$ ).

Hence, we have shown that  $S$  and  $S'$  miss deadlines for the same subset of legal arrival patterns (which may be null). Hence,  $S$  is schedulable if and only if  $S'$  is schedulable.

With uniprocessor EDF, which is what the dispatcher on each processor uses, it is possible to have up to one zero-laxity task and still meet deadlines, provided that the execution requirement  $C$  of that task is sufficiently small. In practice though [9][4], it has been observed that, almost always, the execution requirement of the zero-laxity task can be made such that the processor is utilised above 90% and the system still remains schedulable. According to the authors of C=D, it was this observation that guided its design.

In terms of implementation, a migratory task is scheduled as a zero-laxity task (or, equivalently, at the highest priority) on each processor that it utilises, except for the last one, whereupon it is scheduled as a regular (non zero-laxity) EDF task. Since the task cannot be preempted when executing at zero laxity, no bookkeeping is needed for tracking its accumulated execution time and averting overruns on the respective processor. Rather, the task migrates to its next processor at the end of the pseudo-deadline associated with each piece. This migration can be set up by a timer. On its last processor, the migrating task is not modelled as zero-laxity; rather, it is scheduled under EDF with an associated relative pseudo-deadline set to  $D$  (the true deadline of the task) minus the sum of the respective relative pseudo-deadlines on the previous processors.

## 2.2 Existing variants

C=D was introduced not as a rigid algorithm but rather as a general approach, which can accommodate different strategies for task assignment and splitting. Therefore, it is not inherently tied to any particular bin-packing scheme – either First-Fit or Best-Fit can be, used for example. However, according to one classification criterion, we distinguish between the following two approaches to task splitting: (i) interleaved bin-packing and splitting or (ii) all bin-packing strictly preceding all task splitting.

**The first approach** attempts to assign as many tasks as possible on the current processor (starting from the first one) until there is no task which could be assigned there integrally with schedulability preserved, subject to existing assignments. At this stage, some unassigned task  $\tau_i$  is selected for splitting between the current processor  $P_p$  and the next one  $P_{p+1}$ . The first “piece” (or subtask) of  $\tau_i$  is modelled as a zero-laxity task on  $P_p$  whose execution requirement is set (according to exact sensitivity analysis) to the maximum value (say  $C'_i$ ) that preserves the schedulability. The remaining execution requirement of  $\tau_i$  is modelled as another subtask, with execution requirement  $C_i - C'_i$  and relative deadline  $D_i - D'_i$ , which is added to the pool of unassigned tasks. Subsequently, the bin-packing continues on  $P_{p+1}$  until there is need to split again in the same manner. The algorithm succeeds if all tasks are assigned; it fails if it runs out of processors. Note

that it is possible to ensure that the remaining “stub” subtask of  $\tau_i$  will not have to be split again by assigning it directly to  $P_{p+1}$ , prior to any other task, rather than adding to the pool of unassigned tasks.

**The second approach**, on the other hand, only switches to task splitting once no additional task can be integrally assigned to any processor with schedulability preserved, subject to existing assignments. Remaining unassigned tasks are then split in the manner earlier described, over as many processors as necessary.

These two general approaches can be used with different bin-packing schemes as well as different orderings for the selection of which task to pack or split next. In simulations, these choices are shown to have some effect on average-case performance, depending also on the way that the task parameters are generated. However, in the context of hard-real time scheduling, *a priori* guarantees for the scheduling performance of an algorithm are desirable. Yet, so far, there exists no proven utilisation bound for any variant of C=D, despite empirical evidence of its good performance. This situation motivated this work. In the next section, we will formulate a new variant of C=D, designed so as to facilitate the derivation of its utilisation bound. Subsequently, in Section 4 we are going to prove the respective utilisation bound.

### 3 The clustered variant of C=D under consideration

In order to find out what utilisation bounds are possible for an algorithm design using a C=D approach with respect to task splitting, we took a step back from the variants already outlined in [9]. Rather, we opted for a simplification of those existing variants, which would make the respective utilisation bound easier to derive – even at the expense of average-case performance.

This simplification consists in allowing at most one migratory task to utilise each processor. In other words, there is at most one piece, by a respective migratory task, on each processor. This piece (subtask) may or may not be zero-laxity. This arrangement divides the processors into disjoint clusters, which share no tasks. Hence we term the approach *Clustered C=D*. One of the motivations for our approach is that, in the context of prior work [15], we already have some results on the schedulability of implicit-deadline tasks scheduled under EDF on one processor under interference from a single higher-priority task, which we can apply.

**Outline** The high-level pseudocode in Figure 1 defines the clustered approach. Tasks are first ordered by non-increasing  $T_i$ . This particular ordering is important, as we will later show, because our derivation of the algorithm’s utilisation bound relies on it. Tasks are then assigned one by one, integrally (using First-Fit bin-packing) or, if that fails, by splitting (according to a C=D approach). Each time that a task is split it forms a cluster, consisting of consecutively indexed processors. A variable (initialised to 1) tracks the index of the processor from which the next cluster (if necessary to create) should start. The algorithm can only fail if some task (which previously could not be assigned integrally) cannot be split over the remaining processors (line 8).

Figure 2 depicts an example of a 16-task set assigned onto 6 processors using Clustered C=D. In this example, tasks  $\tau_1$  to  $\tau_{13}$  were successfully assigned integrally. This

```

1. //tasks are indexed by non-increasing  $T_i$ 
2. int q=1; //stores index of processor to split onwards from
3. for (each task  $\tau_i$ )
4. {try assigning  $\tau_i$  integrally to some processor using
   First-Fit bin-packing and an exact schedulability test;

5. if ( $\tau_i$  could not be assigned to any processor with
   schedulability preserved, subject to existing assignments)
6. {re-index processors  $P_q$  to  $P_m$  by non-decreasing utilisation;

7. split  $\tau_i$  in  $k$  (as few as possible) pieces using a C=D approach,
   over processors  $P_q$  to  $P_{q+k-1}$ , using exact schedulability test
   to maximize each zero-laxity piece;

8. if (we ran out of processors)
9. return(FAILURE);
10 else
11. q=q+k;
12. }
13. }

//this line is reached only if all tasks were
//assigned (integrally or by splitting)
14. return(SUCCESS);

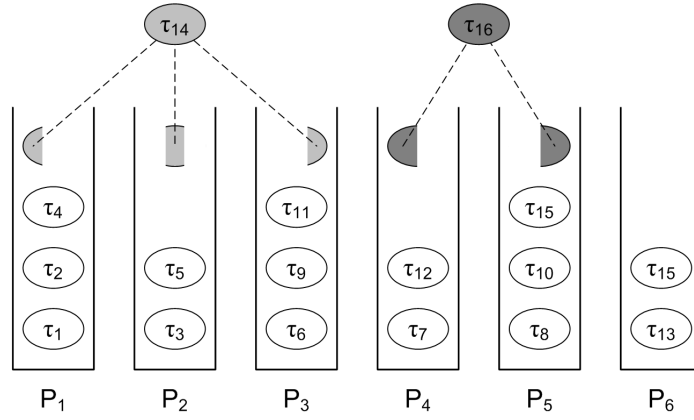
```

**Fig. 1.** Pseudocode for Clustered C=D splitting.

was not the case for  $\tau_{14}$ , which had to be split. Hence the first cluster  $\{P_1, P_2, P_3\}$  is formed. It happens that the next task  $\tau_{15}$  can be assigned integrally on some processor. However, the last task ( $\tau_{16}$ ) has to be split (this time, from processor  $P_4$  onwards) and another cluster  $\{P_4, P_5\}$  is formed. Processors with no migrating tasks can also be thought of as clusters of a single processor. In this example, this is the case of  $\{P_6\}$ .

**Motivation for clustering** As hinted earlier, enforcing this clustered arrangement entails a potential performance penalty due to fragmentation. Namely, unlike processors with zero-laxity subtasks, the last processor in each cluster may be underutilised. This is because, although that processor may still have spare processing capacity, by design, this capacity cannot be used for accommodating an additional split task piece. Nevertheless, the clustering approach formulated above can be used as an intermediate step in the derivation of a non-clustered solution which would not suffer from the fragmentation effects described. Not only that, but the derivative non-clustered variant of C=D (which we will next outline) would dominate Clustered C=D, hence also “inheriting” its least schedulable utilisation bound. This is important because, so far, no utilisation bound for the original non-clustered variants of C=D has been proven.

A dominant non-clustered variant of C=D can be obtained from Clustered C=D as a special case, by preventing the algorithm from declaring failure at line 8. Instead, at



**Fig. 2.** An example of task assignments under Clustered C=D splitting.

that point the algorithm could allow all processors with spare capacity (i.e. including the last processor of each cluster) to be used, in an attempt to accommodate remaining unassigned tasks. This would undo the clustering – but only if absolutely necessary. Therefore, we can draw the following conclusions:

**Lemma 2.** *For the non-clustered variant of C=D outlined above it holds that:*

1. *It dominates Clustered C=D.*
2. *Its least upper bound on schedulable utilisation is no less than that of Clustered C=D.*

*Proof.* By design, given the comments above.

**Some interesting observations regarding the pseudocode of Clustered C=D** Before proceeding with the derivation of the utilisation bound, for the algorithm described above, it is worth providing some insight into certain aspects of its design. We will do this by studying the behaviour of C=D in different examples, without some of our explicit provisions.

First, it is worth stressing the importance of assigning/splitting tasks in order of non-increasing inter-arrival time.

**Lemma 3.** *If tasks are not allocated to processors in order of non-increasing inter-arrival time, the least upper bound on processor utilization achieved by the C=D algorithm is not greater than 50%.*

*Proof.* Consider the implicit-deadline task set  $\Gamma = \{\tau_1, \dots, \tau_{m+1}\}$ , with each task  $(C, D, T)$  having the following attributes:

$$1 \leq i \leq m : \tau_i = (0.5 + \varepsilon, 1, 1)$$

$$\tau_{m+1} = ((0.5 - \varepsilon)m + \varepsilon, m, m)$$



where  $m$  is the number of processors and  $\varepsilon \rightarrow 0^+$ . In this case,  $\tau_{m+1}$  will be the only migrating task, and will be broken up into  $m$  zero-laxity subtasks with  $C = D = 0.5 - \varepsilon$  and  $T = m$  on processors  $P_1$  to  $P_m$ . However, this would still leave it with an outstanding execution requirement of  $\varepsilon$  time units, which cannot be accommodated by any processor. Therefore,

$$\lim_{m \rightarrow \infty} \frac{U(\Gamma)}{m} = \frac{1}{2}$$

Observe that both Clustered C=D and the non-clustered variant dominating it would fail, for the above example. This observation led us to enforce the task ordering by non-increasing inter-arrival time, as an integral aspect of Clustered C=D.

Even under this ordering, though, we can see that the least upper bound on processor utilization for C=D is not higher than 75%.

**Theorem 1.** *The least upper bound on processor utilisation for the C=D algorithm is not higher than  $\frac{3}{4}m$  for large values of  $m$ .*

*Proof.* Without loss of generality, assume that  $m$  is divisible by 4. We will construct a task set  $\Gamma$  with  $n = \frac{3m}{4} + 1$  tasks that could not be dealt with by the C=D scheme. Let  $\Gamma$  be composed of three types of tasks: there are

- $m$  type-1 tasks in the form  $(\frac{1}{2} + \varepsilon, 1, 1)$ ;
- $\frac{m}{4}$  type-2 tasks in the form  $(\frac{3}{4} - \frac{\varepsilon}{2}, \frac{3}{4}, \frac{3}{4})$ ;
- and one type-3 task in the form  $(\frac{1}{4}, \frac{1}{2}, \frac{1}{2})$ ;

where  $\varepsilon$  is a small positive constant. From Lemma 3 we know that unless the tasks are allocated in non-increasing order of their periods, the least upper bound on processor utilization cannot be greater than 50% of  $m$  in the general case. Hence, assume a non-increasing order. This means that all type-1 tasks are allocated as non-migrating and the other tasks should be defined as migrating, those of type 2 being allocated first.

When defining the subtasks of the type-2 tasks, one will see that 3 of them should be allocated as having their relative deadlines equal to their execution costs. That is, these three subtasks of each type-2 task always execute with the highest priority (recall Lemma 1). As there are up to two instances (jobs) of these type-2 subtasks interfering in the execution of the non-migrating tasks, their maximum execution cost  $c$  must satisfy the following:

$$2c + \frac{1}{2} + \varepsilon = 1 \Rightarrow c = \frac{1 - 2\varepsilon}{4}$$

Now the type-3 task must be allocated. We note that after allocating type-2 tasks, there are  $\frac{3m}{4}$  processors where no other task can be allocated and  $\frac{m}{4}$  with utilization  $\frac{1}{2} + 2\varepsilon$ , corresponding to the utilization of the type-1 task and that of the fourth subtask of a type-2 task. We also note that the maximum interference a type-2 task can suffer during its execution cannot exceed  $\frac{\varepsilon}{2}$ , which is its slack. This means that there are  $\frac{m\varepsilon}{8}$  of unallocated processor utilization that can be used but the type-3 task requires 50% of a processor. Computing  $U(\Gamma)$ ,

$$U(\Gamma) = \sum_1^m \left( \frac{1}{2} + \varepsilon \right) + \sum_1^{\frac{m}{4}} \left( \frac{3 - 2\varepsilon}{3} \right) + \frac{1}{2} = \frac{3m}{4} + \frac{2m\varepsilon}{3} + \frac{1}{2}$$

Assuming  $\varepsilon \approx 0$ , we have that

$$\lim_{m \rightarrow \infty} \frac{U(\Gamma)}{m} = \lim_{m \rightarrow \infty} \frac{3}{4} + \frac{1}{2m} = \frac{3}{4}$$

The above theorem indicates that if we would be able to prove a least utilisation bound for Clustered C=D which would be close to 75%, then the performance of this algorithm (and its non-clustered dominant variant) would also be close to the theoretical limits inherent to the C=D approach – in other words, the exact utilisation bound, whatever that is.

Another aspect of the pseudocode for Clustered C=D that merits discussion is the re-indexing of processors (performed at line 6). This occurs whenever some task needs to be split and involves all the candidate processors to accommodate it ( $P_q$  to  $P_m$ ). These processors are rearranged then by order of non-decreasing utilisation, before the algorithm proceeds with splitting the task in consideration from  $P_q$  (as derived by the re-indexing) onwards, over as many processors as needed. This arrangement, as will be seen later, is relied upon for the derivation of the utilisation bound.

Hence, after having highlighted the motivation behind the design aspects of Clustered C=D, we can now turn to identifying its utilisation bound.

## 4 Derivation of the utilisation bound of Clustered C=D

In this section, we will prove the utilisation bound of  $\frac{13}{18}$  for the clustered variant of C=D above introduced. Given that utilisation bounds are a meaningful performance metric only when the task set  $\tau$  to be scheduled is implicit-deadline, we henceforth assume that for each task  $\tau_i$ , it holds that  $C_i \leq D_i = T_i$ .

### 4.1 Useful results from the domain of uniprocessor scheduling

The following few results address the schedulability of uniprocessor systems. However, they form the foundations for later proving the utilisation bound of the multiprocessor scheduling scheme in consideration.

**Theorem 2.** *Let  $\tau_0$  be a task scheduled at the highest priority on some processor  $P_p$ , together with a set  $\Gamma_p$  of implicit-deadline background tasks, which are scheduled under EDF. Additionally, assume that  $T_0 \leq T_j$ ,  $\forall \tau_j \in \Gamma_p$ . Then, if*

$$u_0 \leq \frac{1 - U(\Gamma_p)}{1 + U(\Gamma_p)} \quad (1)$$

*no deadlines can be missed.*

*Proof.* It is known from Theorem 3 in [15] that, for the case described above, no deadlines can be missed if

$$u_0 \leq \frac{1 - U(\Gamma_p)}{1 + \frac{U(\Gamma_p)}{\left\lfloor \frac{\min_{\tau_j \in \Gamma_p} T_j}{T_0} \right\rfloor}}$$

Since, from the assumption, it holds that  $T_0 \leq \min_{\tau_j \in \Gamma_p} T_j$ , the above sufficient condition for schedulability can be relaxed to Inequality (1)

**Corollary 1.** *The utilisation of each zero-laxity subtask assigned to a processor  $P_p$  by the clustered C=D algorithm during task splitting is lower-bounded by  $\frac{1-U(\Gamma_p)}{1+U(\Gamma_p)}$ .*

*Proof.* Follows from Theorem 2 and Lemma 1 and the fact that the algorithm uses an exact schedulability test to determine the maximum execution requirement by the subtask that preserves the schedulability of  $\Gamma_p$ . Therefore, the right-hand side of Inequality (1) is a lower bound for that utilisation.

#### 4.2 Proof of utilisation bound

At this stage, we can tackle the derivation of the least schedulable utilisation bound for Clustered C=D. For that derivation, we will rely on the above results for uniprocessors, with each zero-laxity subtask of a migrating task corresponding to  $\tau_0$  on its respective processor.

**Theorem 3.** *The total task utilisation  $U_{cl}$ , accommodated by each cluster formed by the Clustered C=D algorithm is lower-bounded by  $\frac{13}{18}k$ , where  $k$  is the number of processors in that cluster.*

*Proof.* Let us assume that the cluster in consideration spans processors  $P_q$  to  $P_{q+k-1}$ . From Corollary 1 and the fact that the split task “exhausts” the available scheduling capacity on the first  $k-1$  processors of the cluster ( $P_q$  to  $P_{q+k-2}$ ) (but still needs a final piece on the  $k^{th}$  processor to be schedulable), it follows that:

$$u_i > \sum_{p=q}^{q+k-2} \frac{1-U(\Gamma_p)}{1+U(\Gamma_p)} \quad (2)$$

From the fact that  $\frac{d}{dx} \left( \frac{1-x}{1+x} \right) < 0$  and  $\frac{d^2}{dx^2} \left( \frac{1-x}{1+x} \right) > 0$  over  $[0,1]$ , via application of Jensen’s inequality [14] to the right-hand side of Inequality (2), we obtain

$$\sum_{p=q}^{q+k-2} \frac{1-U(\Gamma_p)}{1+U(\Gamma_p)} > (k-1) \cdot \frac{1-\bar{U}}{1+\bar{U}} \quad (3)$$

where

$$\bar{U} = \frac{1}{k-1} \sum_{p=q}^{q+k-2} U(\Gamma_p) \quad (4)$$

Combining this with Inequality (2), we obtain

$$u_i > (k-1) \cdot \frac{1-\bar{U}}{1+\bar{U}} \quad (5)$$

Now, let us consider two cases:

– **Case 1:**  $k = 2$

Then the cluster consists of two processors  $P_q$  and  $P_{q+1}$ , over which task  $\tau_i$  is split. From the bin-packing scheme used, it follows that:

$$\begin{aligned} U(\Gamma_q) + U(\Gamma_{q+1}) &> 1 \\ u_i + U(\Gamma_q) &> 1 \\ u_i + U(\Gamma_{q+1}) &> 1 \end{aligned}$$

Adding these inequalities together and dividing by 2 yields

$$u_i + U(\Gamma_q) + U(\Gamma_{q+1}) \geq \frac{3}{2} > 2 \cdot \frac{13}{18} \Rightarrow U_{cl.} > \frac{13}{18}k \quad (6)$$

– **Case 2:**  $k \geq 3$

Then, for the total utilisation  $U_{cl.}$  of the cluster it holds that

$$\begin{aligned} U_{cl.} &= u_i + \sum_{p=q}^{q+k-1} U(\Gamma_p) = u_i + \sum_{p=q}^{q+k-2} U(\Gamma_p) + U(\Gamma_{q+k-1}) \xrightarrow{(5)} \\ U_{cl.} &> U(\Gamma_{q+k-1}) + (k-1)\bar{U} + (k-1) \cdot \frac{1-\bar{U}}{1+\bar{U}} \Rightarrow \\ U_{cl.} &> U(\Gamma_{q+k-1}) + (k-1) \frac{\bar{U}^2 + 1}{1+\bar{U}} \end{aligned} \quad (7)$$

From the fact that processors  $P_q$  to  $P_m$  are reindexed in order of non-decreasing utilisation, each time that a task is split (line 6 of the pseudocode) it follows that, at the time that  $\tau_i$  is split,  $U(\Gamma_{q+k-1}) \geq \bar{U}$ . Taking advantage of this, in conjunction with (7), we obtain:

$$\begin{aligned} U_{cl.} &> \bar{U} + (k-1) \frac{\bar{U}^2 + 1}{1+\bar{U}} \Rightarrow \\ \frac{U_{cl.}}{k} &> \frac{1}{k} \left( \bar{U} + (k-1) \frac{\bar{U}^2 + 1}{1+\bar{U}} \right) \end{aligned} \quad (8)$$

With some algebraic rewriting, this can be equivalently expressed as

$$\frac{U_{cl.}}{k} > \bar{U} + \left( \frac{k-1}{k} \right) \left( \frac{1-\bar{U}}{1+\bar{U}} \right) \quad (9)$$

The quantity  $\frac{1-\bar{U}}{1+\bar{U}}$  is positive. The quantity  $\frac{k-1}{k}$  is also positive and an increasing function of  $k$ . Hence the right-hand side of the above inequality is minimised for the smallest value of  $k$ , which is  $k = 3$  according to the assumption of the case. Via substitution of this value we obtain

$$\frac{U_{cl.}}{k} > \bar{U} + \frac{2}{3} \left( \frac{1-\bar{U}}{1+\bar{U}} \right) \quad (10)$$

From the properties of the bin-packing scheme used, it follows that  $\bar{U} > \frac{1}{2}$ . Moreover, the right-hand side of Inequality (10) is an increasing function of  $\bar{U}$  over  $[\frac{1}{2}, 1]$ . From these two observations we obtain that a lower bound for the right-hand side of Inequality 10 can be obtained by substituting  $\bar{U} = \frac{1}{2}$ . Then, we obtain

$$\frac{U_{cl.}}{k} > \frac{13}{18} \Rightarrow U_{cl.} > \frac{13}{18}k. \quad (11)$$

Hence in either case the claim holds.

Note that the utilisation of a cluster may still increase, after its formation, via the potential integral assignment of additional tasks to its component processors.

**Definition 1.** *In the case that Clustered C=D declares failure, the term last candidate cluster denotes the set of processors  $\{P_q \dots P_m\}$  upon which the algorithm attempted to split a task immediately prior to declaring failure.*

Intuitively, the algorithm attempted to split the task in consideration over candidate clusters  $\{P_q, P_{q+1}\}$ ,  $\{P_q, P_{q+1}, P_{q+2}\}$  and so on until  $\{P_q, \dots, P_m\}$  but failed in all cases. Note that, in a trivial case, the last candidate cluster could be  $\{P_m\}$ .

**Lemma 4.** *Assume that Clustered C=D declares failure upon attempting to split a task  $\tau_i$ . Let  $U_{last}$  denote the utilisation of the last candidate cluster before the attempt to split  $\tau_i$  and let  $k$  be the number of processors in the last candidate cluster. Then, it holds that  $U_{last} + u_i > \frac{13}{18}k$ .*

*Proof.* For  $k = 1$  the claim trivially holds. For  $k \geq 2$ :

From the fact that the algorithm declares failure, we know that  $k - 1$  zero-laxity subtasks of  $\tau_i$  were assigned on the  $k - 1$  first processors in the last candidate cluster. On the final processor, the algorithm tested the schedulability of a final  $k^{th}$  subtask, with the remaining execution requirement of  $\tau_i$ , and failed.

From the fact that EDF is a sustainable scheduling algorithm [5], reducing the execution requirement of that last subtask on the last processor cannot negatively impact on its schedulability. Let us therefore reduce its execution requirement (without changing its deadline or interarrival time) to a value that renders it schedulable on the last processor, together with all other tasks assigned there. From Theorem 3, this means that the cluster would then be utilized above  $\frac{13}{18}$  – even if the utilisation of  $\tau_i$  was discounted. In turn, this means that  $U_{last} + u_i > \frac{13}{18}k$ .

**Theorem 4.** *The utilisation bound of Clustered C=D is at least  $\frac{13}{18}$ .*

*Proof.* We will prove this by showing that if the algorithm fails to schedule a task set  $\Gamma$ , this means that  $U(\Gamma) > \frac{13}{18}m$ .

Assume that the algorithm declares failure to split a task  $\tau_i$ . From Theorem 3 we know that all clusters successfully formed by the algorithm up to that point are utilised above  $\frac{13}{18}$ . From Lemma 4 we also know that, were  $\tau_i$  to be assigned to the last candidate cluster (shedulability considerations aside), its resulting utilisation would exceed  $\frac{13}{18}$ . This means that  $U(\{\tau_1, \dots, \tau_i\}) > \frac{13}{18}m$ . In turn, since  $\{\tau_1, \dots, \tau_i\} \subseteq \Gamma$ , this implies that  $U(\Gamma) > \frac{13}{18}m$ .

### 4.3 Observations regarding the bin-packing scheme

The algorithm in Figure 1 uses First-fit bin-packing but other reasonable bin-packing heuristics can be used instead [10]. For example, Best-Fit could be used instead and all of the reasoning that leads to the derivation of the utilisation bound (and thus the bound itself) would still hold.

In order to characterise what properties a candidate bin-packing scheme should possess in order to be used in the Clustered C=D algorithm, instead of First-Fit, without compromising the proven utilisation bound, let us inspect the proof of Theorem 3. Therein, the only properties resulting from the assignment of non-split tasks that are relied upon are the following:

**Property 1:**  $\frac{1}{2} < \bar{U} \stackrel{\text{def}}{=} \sum_{p=q}^{q+k-2} U(\Gamma_p)$ , for the first  $k-1$  processors in a cluster  $\{P_q, \dots, P_{q+k-1}\}$ .

**Property 2:**  $U(\Gamma_{q+k-1}) \geq \bar{U}$ , for the last processor in the cluster.

Upon closer inspection, if Property 1 holds, then Property 2 is ensured by the processor re-indexing (line 6 of the pseudocode), prior to each task splitting. Hence, only Property 1 needs to be safeguarded, when switching to an alternative bin-packing scheme.

One simple way of ensuring that this property holds is by preventing by design a task to be assigned to a processor with no other tasks yet assigned to it *unless* this task could not fit on any of the processors with tasks already assigned to them. Note that the order in which processors with tasks already assigned to them are tried is not relevant for ensuring Property 1; it could even vary for each task or it could even be random.

Preventing a task from being assigned to a processor with no other tasks yet assigned to it unless it could not be assigned to any other processor (with tasks) means that, whenever the need to split a task arises, it holds that

$$U(\Gamma_p) + U(\Gamma_r) > 1$$

where  $P_p$  and  $P_r$  are the two least utilised from among the first  $k-1$  processors of the cluster in consideration. In turn, this means that Property 1 holds.

## 5 Conclusions

Providing high utilisation guarantees without imposing too many restrictions on implementation has been a challenge for the real-time research community regarding the multiprocessor scheduling problem. In this context, the C=D algorithm is noticeable by its implementation simplicity and good average performance. However, no theoretical bound on system utilisation had been derived so far. In this paper we have looked into this open problem and have discussed some important characteristics of C=D. We have shown that if one does not consider assigning tasks to processors in order of non-increasing task periods, the utilisation bound for the C=D algorithm can be as low as 50%. Furthermore, we have established an interval of  $[0.72\bar{2}, 0.75]$  within which such a bound lies. These results bring about important theoretical aspects for a scheduling algorithm known to perform well from a practical perspective.

## Acknowledgements

José Augusto Santos-Jr has received financial support by CAPES. This work is part of a research project jointly funded by CNPq and Federal University of Bahia.

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within SMARTS project, ref. FCOMP-01-0124-FEDER-020536.

## References

1. James H. Anderson, Vasile Bud, and UmaMaheswari C. Devi. An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 199–208, 2005.
2. B. Andersson and K. Bletsas. Sporadic Multiprocessor Scheduling with Few Preemptions. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 243–252, 2008.
3. B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *Proc. 12th IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.
4. Patricia Balbastre, Ismael Ripoll, and Alfons Crespo. Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 65–74, 2006.
5. Sanjoy K. Baruah and Alan Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS)*, pages 159–168, 2006.
6. Konstantinos Bletsas and Björn Andersson. Notional processors: an approach for multiprocessor scheduling. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, 2009.
7. Konstantinos Bletsas and Björn Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. In *Proc. of 30th Real-Time Systems Symposium (RTSS)*, pages 447–456, 2009.
8. Konstantinos Bletsas and Björn Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. *Real-Time Systems*, 47(4):319–355, 2011.
9. A. Burns, R.I. Davis, P. Wang, and F. Zhang. Partitioned EDF Scheduling for Multiprocessors using a C=D Scheme. *Real-Time Systems (published online)*, 48(1):3–33, 2011.
10. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In Dorit S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.
11. François Dorin, Patrick Meumeu Yoms, Joël Goossens, and Pascal Richard. Semi-partitioned hard real-time scheduling with restricted migrations upon identical multiprocessor platforms. CoRR abs/1006.2637, 2010.
12. Frédéric Fauberteau, Serge Midonnet, and Laurent George. Improvement of schedulability bound by task splitting in partitioning scheduling. In *Proceedings of the 1st International Real-Time Scheduling Open Problems Seminar (RTSOPS 2010)*, pages 20–21, 2010.
13. Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Fixed-priority multiprocessor scheduling with Liu and Layland's utilization bound. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 165–174, 2010.
14. J. Jensen. Sur les Fonctions Convexes et les Inégalités entre les Valeurs Moyennes. *Acta Mathematica*, 30:175–193, 1906.

15. José Augusto Santos Jr. and George Lima. Sufficient schedulability tests for edf-scheduled real-time systems under interference of a high priority task. In *Proc. of the 2nd Brazilian Symposium on Computer Systems Engineering (SBESC)*, pages 1–6, 2012.
16. S. Kato, N. Yamasaki, and Y. Ishikawa. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 249–258, 2009.
17. Shinpei Kato and Nobuki Yamasaki. Portioned EDF-based scheduling on multiprocessors. In *Proceedings of the 8th ACM/IEEE International Conference on Embedded Software (EMSOFT)*, pages 139–148, 2008.
18. Shinpei Kato and Nobuki Yamasaki. Portioned static-priority scheduling on multiprocessors. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12, 2008.
19. Karthik Lakshmanan, Ragunathan Rajkumar, and John P. Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *Proc. of the 21st Euromicro Conf. on Real-Time Systems*, pages 239–248, 2009.
20. Greg Levin, Shelby Funk, Caitlin Sadowski, Ian Pye, and Scott Brandt. DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS 2010)*, pages 3–13, 2010.
21. Ernesto Massa and George Lima. A bandwidth reservation strategy for multiprocessor real-time scheduling. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 175–183, 2010.
22. G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness. In *Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 15–24, 2011.
23. G. Nelissen, V. Berten, V. Nelis, J. Goossens, and D. Milojevic. U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 13–23, 2012.
24. Paul Regnier, George Lima, Ernesto Massa, Greg Levin, and Scott Brandt. RUN: optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proceedings of the 32nd Real-Time Systems Symposium*, pages 104–115, 2011.



# Models for Real-Time Workload: A Survey

Martin Stigge and Wang Yi

Department of Information Technology  
Uppsala University, Sweden  
Email: {martin.stigge,yi}@it.uu.se

**Abstract.** This paper provides a survey on task models to characterize real-time workloads at different levels of abstraction for the design and analysis of real-time systems. It covers the classic periodic and sporadic models by Liu and Layland et al., their extensions to describe recurring and branching structures as well as general graph- and automata-based models to allow modeling of complex structures such as mode switches, local loops and also global timing constraints. The focus is on the precise semantics of the various models and on the solutions and complexity results of the respective feasibility and schedulability analysis problems for preemptable uniprocessors.

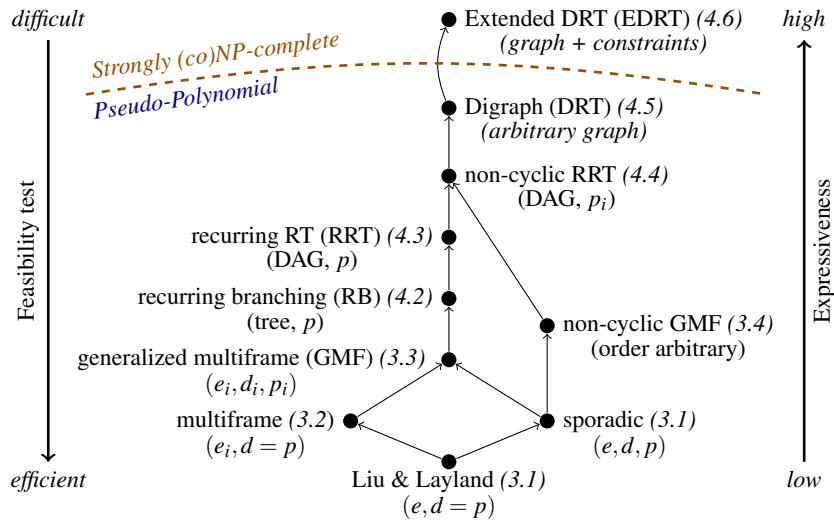
## 1 Introduction

Real-time systems are often implemented by a number of concurrent tasks sharing hardware resources, in particular the execution processors. The designer of such systems need to construct workload models characterizing the resource requirements of the tasks. With a formal description of the workload, a resource scheduler may be designed and analyzed. The fundamental analysis problem to solve in the design process is to check (and thus to guarantee) the schedulability of the workload, i.e., whether the timing constraints on the workload can be met if the scheduler is used. In addition, the workload model may also be used to optimize the resource utilization as well as the average system performance.

In the past decades, workload models have been studied intensively in the theory of real-time scheduling [20] and other contexts, for example performance analysis of networked systems [19]. The research community of real-time systems has proposed a large number of models (often known as task models) allowing for the description and analysis of real-time workloads at different levels of abstraction. One of the classic works is the periodic task model due to Liu and Layland [40, 37, 38], where tasks generate resource requests at strict periodic intervals. The periodic model was extended later to the sporadic model [42, 9, 35] and multiframe models [44, 7] to describe non-regular arrival times of resource requests, and non-uniform resource requirements. In spite of a limited form of variation in release times and worst-case execution times, these repetitive models are highly deterministic. To allow for the description of recurring and non-deterministic behaviours, tree-like recurring models based on directed acyclic graphs are introduced [11, 10] and recently extended to the Digraph model based on arbitrary directed graphs [50, 49] to allow for modeling of complex structures like mode switches and local loops as well as global timing constraints on resource requests. With origin

from formal verification of timed systems, the model of task automata [28] was developed in the late 90s. The essential idea is to use timed automata to describe the release patterns of tasks. Due to the expressiveness of timed automata, it turns out that all the models above can be described using the automata-based model.

In addition to the expressiveness, a major concern in developing these models is the complexity of their analysis. It is not surprising that the more expressive they are, the more difficult to analyze they tend to be. Indeed, the model of task automata is the most expressive model with the highest analysis complexity, which marks the borderline between decidability and undecidability for the schedulability analysis problem of workloads. On top of the operational models summarized above that capture the timed sequences of resource requests representing the system executions, alternative characterizations of workloads using functions on the time interval domain have also been proposed, notably Demand Bound Function (DBF), Request Bound Function (RBF) and Real-Time Calculus (RTC) [55] that can be used to specify the accumulated workload over a sliding time window. They have been further used as a mathematical tool for the analysis of operational workload models.



**Fig. 1.** A hierarchy of task models. Arrows indicate the generalization relationship. The higher in the hierarchy, the higher the expressiveness, but also the more expensive the feasibility test. We denote the corresponding survey section in parentheses.

This paper is intended to serve as a survey on existing workload models for pre-emptable uniprocessor systems. Figure 1 outlines the relative expressive powers vs. the degree of feasibility analysis difficulty for the models (marked with subsections where they are described). In the following sections we go through this model hierarchy and

provide a brief account for each of the well-developed models including the Real-Time Calculus representing models on a higher level of abstraction.

## 2 Terminology

We first review some terminology used in the rest of this survey. The basic unit describing workload is a *job*, characterized by a *release time*, a *worst-case execution time* (WCET) and a *deadline*. The higher-level structures which generate (potentially infinite) sequences of jobs are *tasks*. The time interval between the release time and the deadline of a job is called its *scheduling window*. The time interval between the release time of a job and the earliest release time of its succeeding job from the same task is its *exclusion window*.

A task system typically contains a set of tasks that at runtime generate sequences of jobs concurrently and a *scheduler* decides at each point in time which of the pending jobs to execute. We distinguish between two important classes of schedulers on preemptable uniprocessor systems:

**Static priority schedulers.** Tasks are ordered by priority. All jobs generated by a task of higher priority get precedence over jobs generated by tasks of lower priority.

**Dynamic priority schedulers.** Tasks are not ordered a priori; at runtime, the scheduler may choose freely which of the pending jobs to execute.

Given suitable descriptions of a workload model and a scheduler, one of the important questions is whether all jobs that could ever be generated will meet their deadline constraints, in which case we call the workload *schedulable*. This is determined in a formal *schedulability test* which can be:

**Sufficient** if it is failed by all non-schedulable task sets,

**Necessary**, if it is satisfied by all schedulable task sets, or

**Precise or exact** if it is both sufficient and necessary.

More precisely, if a task set satisfies a sufficient test, it is schedulable and if it fails a necessary test, it is non-schedulable. Workload that is schedulable with *some* effective scheduler is called *feasible*, determined by a feasibility test that can also be either sufficient, necessary or both.

## 3 Repetitive Models: From Liu and Layland Tasks to GMF

In this section, we show the development of task systems from the periodic task model to different variants of the Multiframe model, including techniques for their analysis.

### 3.1 Periodic and Sporadic Tasks

The first task model with *periodic* tasks was introduced 1973 by Liu and Layland [40]. Each periodic task  $T = (P, E)$  in a task set  $\tau$  is characterized by a pair of two integers: period  $P$  and WCET  $E$ . It generates an infinite sequence  $\rho = (J_1, J_2, \dots)$  of jobs  $J_i =$

$(r_i, e_i, d_i)$  with release time  $r_i$ , execution time  $e_i$  and deadline  $d_i$  such that  $r_{i+1} = d_i = r_i + P$  and  $e_i \leq E$ . This means that jobs are released periodically and have *implicit* deadlines at the release times of the next job.

A relaxation of this model is to allow jobs to be released at later time points, as long as *at least*  $P$  time units pass between adjacent job releases of the same task. This is called the *sporadic* task model, introduced by Mok in [42]. Another generalization is to add an explicit deadline  $D$  as a third integer to the task definition  $T = (P, E, D)$ , leading to  $d_i = r_i + D$  for all generated jobs. If  $D \leq P$  for all tasks  $T \in \tau$  then we say that  $\tau$  has *constrained deadlines*, otherwise it has *arbitrary deadlines*.

This model has been the basis for many results throughout the years. Liu and Layland give in [40] a simple feasibility test for implicit deadline tasks: defining the *utilization*  $U(\tau)$  of a task set  $\tau$  as  $U(\tau) := \sum_{T_i \in \tau} E_i / P_i$ , a task set is uniprocessor feasible if and only if  $U(\tau) \leq 1$ . As in later work, proofs of feasibility are often connected to the *Earliest Deadline First (EDF)* scheduling algorithm, which uses dynamic priorities and has been shown to be optimal for a large class of workload models on uniprocessor platforms. Because of its optimality, EDF schedulability is equivalent to feasibility.

*Demand bound function.* For the case of explicit deadlines, Baruah et al. [9] introduced a concept that was later called the *demand bound function*: for each interval size  $t$  and task  $T$ ,  $dbf_T(t)$  is the maximum accumulated worst-case execution time of jobs generated by  $T$  in *any* interval of size  $t$ . More specifically, it counts all jobs that have their full scheduling window inside the interval, i.e., release time *and* deadline. The demand bound function  $dbf(t)$  of the whole system has the property that a task system is feasible if and only if

$$\forall t. dbf(t) \leq t. \quad (1)$$

This condition is a valid test for a very general class of workload models and is of great use in later parts of this survey. It holds for all models generating sequences of independent jobs. A proof can be found in [7].

Focussing on sporadic tasks, Baruah et al. show in [9] that  $dbf(t)$  can be computed with

$$dbf(t) = \sum_{T_i \in \tau} E_i \cdot \max \left\{ 0, \left\lfloor \frac{t - D_i}{P_i} \right\rfloor + 1 \right\}. \quad (2)$$

Using this they prove that the feasibility problem is in coNP. Recently it has been shown by Eisenbrand and Rothvoß [25] that the problem is indeed (weakly) coNP-hard for systems with constrained deadlines.

Another contribution of Baruah et al. in [9] was to show that for the case of  $U(\tau) < c$  for some constant  $c$ , there is a pseudo-polynomial solution of the schedulability problem, by testing Condition (1) for a pseudo-polynomial number of values. The existence of such a constant bound (however close to 1) is a common assumption when approaching this problem since excluding utilizations very close to 1 only rules out very few actual systems.

*Static priorities.* For static priority schedulers, Liu and Layland show already in [40] that the rate-monotonic priority assignment for implicit deadline tasks is optimal, i.e., tasks with shorter periods have higher priorities. They further give an elegant sufficient

schedulability condition by proving that a task set  $\tau$  with  $n$  tasks is schedulable with a static priority scheduler under rate-monotonic priority ordering if

$$U(\tau) \leq n \cdot (2^{1/n} - 1). \quad (3)$$

For sporadic task systems with explicit deadlines, the *response time analysis* technique has been developed. It is based on a scenario in which all tasks release jobs at the same time instant with all following jobs being released as early as permitted. This maximizes the response time  $R_i$  of the task in question, which is why the scenario is often called the *critical instant*. It is shown by Joseph and Pandya [33] and independently by Audsley et al. [3] that  $R_i$  is the smallest positive solution of the recurrence relation

$$R = E_i + \sum_{j < i} \left\lceil \frac{R}{P_j} \right\rceil \cdot E_j, \quad (4)$$

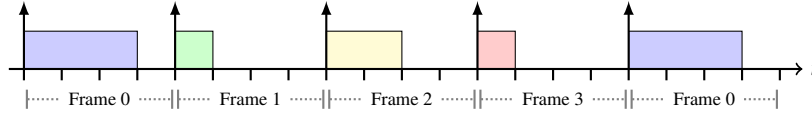
assuming that the tasks are in order of descending priority. This is based on the observation that the interference from a higher priority task  $T_j$  to  $T_i$  during a time interval of size  $R$  can be computed by counting the number of jobs  $T_j$  can release as  $\lceil R/P_j \rceil$  and multiplying that with their worst-case duration  $E_j$ . Together with  $T_i$ 's own WCET  $E_i$ , the response time is derived. Solving Equation (4) leads directly to a pseudo-polynomial schedulability test. Eisenbrand and Rothvoß show in [24] that the problem of computing  $R_i$  is indeed NP-hard.

### 3.2 The Multiframing Model

The first extension of the periodic and sporadic paradigm for jobs of different types, to be generated from the same task was introduced by Mok and Chen in [44]. The motivation is as follows. Assume a workload which is fundamentally periodic but it is known that every  $k$ -th job of this task is extra long. As an example, Mok and Chen describe an MPEG video codec that uses different types of video frames. Video frames arrive periodically, but frames of large size and thus large decoding complexity are processed only once in a while. The sporadic task model would need to account for this in the WCET of *all* jobs, which is certainly a significant overapproximation. Systems that are clearly schedulable in practice would fail standard schedulability tests for the sporadic task model. Thus, in scenarios like this where most jobs are close to an average computation time which is significantly exceeded only in well-known periodically recurring situations, a more precise modeling formalism is needed.

To solve this problem, Mok and Chen introduce in [44] the *Multiframe* model. A multiframing task  $T$  is described as a pair  $(P, \mathbf{E})$  much like the basic sporadic model with implicit deadlines, except that  $\mathbf{E} = (E_0, \dots, E_{k-1})$  is a *vector* of different execution times, describing the WCET of  $k$  potentially different *frames*.

*Semantics.* As before, let  $\rho = (J_1, J_2, \dots)$  be a job sequence with job parameters  $J_i = (r_i, e_i, d_i)$  of release time  $r_i$ , execution time  $e_i$  and deadline  $d_i$ . For  $\rho$  to be generated by a multiframing task  $T$  with  $k$  frames, it has to hold that  $e_i \leq E_{(a+i) \bmod k}$  for some offset  $a$ , i.e., the worst-case execution times cycle through the list specified by vector  $\mathbf{E}$ . The other job parameters  $r_i$  and  $d_i$  behave as before for sporadic implicit-deadline tasks, i.e.,  $r_{i+1} \geq r_i + P = d_i$ . We show an example in Figure 2.



**Fig. 2.** Example for a Multiframe task  $T = (P, \mathbf{E})$  with  $P = 4$  and  $\mathbf{E} = (3, 1, 2, 1)$ . Note that deadlines are implicit.

*Schedulability Analysis.* Mok and Chen provide in [44] a schedulability analysis for static priority scheduling. They provide a generalization of Equation (3) by showing that a task set  $\tau$  is schedulable with a static priority scheduler under rate-monotonic priority ordering if

$$U(\tau) \leq r \cdot n \cdot \left( (1 + 1/r)^{1/n} - 1 \right). \quad (5)$$

The value  $r$  in this test is the minimal ratio between the largest WCET  $E_i$  in a task and its successor  $E_{(i+1) \bmod k}$ . Note that the classic test for periodic tasks in Equation (3) is a special case of (5) with  $r = 1$ .

The proof for this condition is done by carefully observing that for a class of Multiframe tasks called *accumulatively monotonic (AM)*, there is a critical instant that can be used to derive the condition (and further even for a precise test in pseudo-polynomial time by simulating the critical instant). In short, AM means that there is a frame in each task such that all sequences starting from this frame always have a cumulative execution demand at least as high as equally long sequences starting from any other frame. After showing (5) for AM tasks the authors prove that each task can be transformed into an AM task which is equivalent in terms of schedulability. The transformation is via a model called General Tasks from [43] which is an extension of Multiframe tasks to an infinite number of frames and therefore of mainly theoretical interest.

Refined sufficient tests have been developed [32, 57, 41] with less pessimism than the test using the utilization bound in (5). They generally also allow certain tasks of higher utilization than those passing the above test to be classified as schedulable. A precise test of exponential complexity is presented in [58] based on response time analysis as a generalization of (4). The authors also include results for models with jitter and blocking.

### 3.3 Generalized Multiframe Tasks

In the Multiframe model, all frames still have the same period and implicit deadline. Baruah et al. generalize this further in [7] by introducing the *Generalized Multiframe (GMF)* task model. A GMF task  $T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$  with  $k$  frames consists of three vectors:

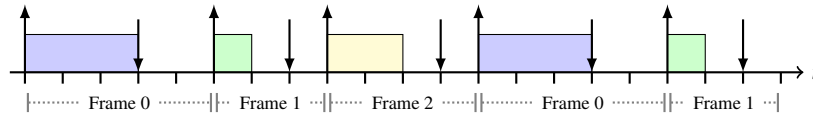
- $\mathbf{P} = (P_0, \dots, P_{k-1})$  for minimum inter-release separations,
- $\mathbf{E} = (E_0, \dots, E_{k-1})$  for worst-case execution times, and
- $\mathbf{D} = (D_0, \dots, D_{k-1})$  for relative deadlines.

For unambiguous notation we write  $P_i^T$ ,  $E_i^T$  and  $D_i^T$  for components of these three vectors in situations where it is not clear from the context which task  $T$  they belong to.

*Semantics.* As a generalization of the Multiframe model, each job  $J_i = (r_i, e_i, d_i)$  in a job sequence  $\rho = (J_1, J_2, \dots)$  generated by a GMF task  $T$  needs to correspond to a frame and the corresponding values in all three vectors. Specifically, we have for some offset  $a$  that:

1.  $r_{i+1} \geq r_i + P_{(a+i) \bmod k}$
2.  $e_i \leq E_{(a+i) \bmod k}$
3.  $d_i = r_i + D_{(a+i) \bmod k}$

An example is shown in Figure 3.



**Fig. 3.** Example for a GMF task  $T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$  with  $\mathbf{P} = (5, 3, 4)$ ,  $\mathbf{E} = (3, 1, 2)$  and  $\mathbf{D} = (3, 2, 3)$ .

*Feasibility Analysis.* Baruah et al. give in [7] a feasibility analysis method based on the demand bound function. The different frames make it difficult to develop a closed-form expression like (2) for sporadic tasks since there is in general no unique critical instant for GMF tasks. Instead, the described method (which we sketch here with slightly adjusted notation and terminology) creates a list of pairs  $\langle e, d \rangle$  of workload  $e$  and some time interval length  $d$  which are called *demand pairs* in later work [50]. Each demand pair  $\langle e, d \rangle$  describes that a task  $T$  can create  $e$  time units of execution time demand during an interval of length  $d$ . From this information it can be derived that  $dbf_T(d) \geq e$  since the demand bound function  $dbf_T(d)$  is the *maximal* execution demand possible during *any* interval of that size.

In order to derive all relevant demand pairs for a GMF task, Baruah et al. first introduce a property called *localized Monotonic Absolute Deadlines (l-MAD)*. Intuitively, it means that two jobs from the same task that have been released in some order will also have their (absolute) deadlines in the same order. Formally, this can be expressed as  $D_i \leq P_i + D_{(i+1) \bmod k}$ , which is more general than the classical notion of constrained deadlines, i.e.,  $D_i \leq P_i$ , but still sufficient for the analysis. We assume this property for the rest of this section.

As preparation, the method from [7] creates a sorted list  $DP$  of demand pairs  $\langle e, d \rangle$  for all  $i$  and  $j$  each ranging from 0 to  $k-1$  with

$$e = \sum_{m=i}^{i+j} E_{m \bmod k}, \quad d = \left( \sum_{m=i}^{i+j-1} P_{m \bmod k} \right) + D_{(i+j) \bmod k}. \quad (6)$$

For a particular pair of  $i$  and  $j$ , this computes in  $e$  the accumulated execution time of a job sequence with jobs corresponding to frames  $i, \dots, (i+j) \bmod k$ . The value

of  $d$  is the time from first release to last deadline of such a job sequence. With all these created demand pairs, and using shorthand notation  $P_{\text{sum}} := \sum_{i=0}^{k-1} P_i$ ,  $E_{\text{sum}} := \sum_{i=0}^{k-1} E_i$  and  $D_{\text{min}} := \min_{i=0}^{k-1} D_i$ , the function  $dbf_T(t)$  can be computed with

$$dbf_T(t) = \begin{cases} 0 & \text{if } t < D_{\text{min}}, \\ \max \{e \mid \langle e, d \rangle \in DP \text{ with } d \leq t\} & \text{if } t \in [D_{\text{min}}, P_{\text{sum}} + D_{\text{min}}), \\ \left\lfloor \frac{t - D_{\text{min}}}{P_{\text{sum}}} \right\rfloor E_{\text{sum}} + dbf_T(D_{\text{min}} + (t - D_{\text{min}}) \bmod P_{\text{sum}}) & \text{if } t \geq P_{\text{sum}} + D_{\text{min}}. \end{cases} \quad (7)$$

Intuitively, we can sketch all three cases as follows: In the first case, time interval  $t$  is shorter than the shortest deadline of any frame, thus not creating any demand. In the second case, time interval  $t$  is shorter than  $P_{\text{sum}} + D_{\text{min}}$  which implies that at most  $k$  jobs can contribute to  $dbf_T(t)$ . All possible job sequences of up to  $k$  jobs are represented in demand pairs in  $DP$ , so it suffices to return the maximal demand  $e$  recorded in a demand pair  $\langle e, d \rangle$  with  $d \leq t$ . In the third case, a job sequence leading to the maximal value  $dbf_T(t)$  must include at least one complete cycle of all frames in  $T$ . Therefore, it is enough to determine the number of cycles (each contributing  $E_{\text{sum}}$ ) and looking up the remaining interval part using the second case.

Finally, [7] describes how a feasibility test procedure can be implemented by checking Condition (1) for all  $t$  at which  $dbf(t)$  changes up to a bound

$$D := \frac{U(\tau)}{1 - U(\tau)} \cdot \max_{T \in \tau} (P_{\text{sum}}^T - D_{\text{min}}^T)$$

with  $U(\tau) := \sum_{T \in \tau} E_{\text{sum}}^T / P_{\text{sum}}^T$  measuring the *utilization* of a GMF task system. If  $U(\tau)$  is bounded by a constant  $c < 1$  then this results in a feasibility test of pseudo-polynomial complexity. Baruah et al. include also an extension of this method to task systems without the *l-MAD* property, i.e., with arbitrary deadlines. As an alternative test method, they even provide an elegant reduction of GMF feasibility to feasibility of sporadic task sets by using the set  $DP$  to construct a *dbf*-equivalent sporadic task set.

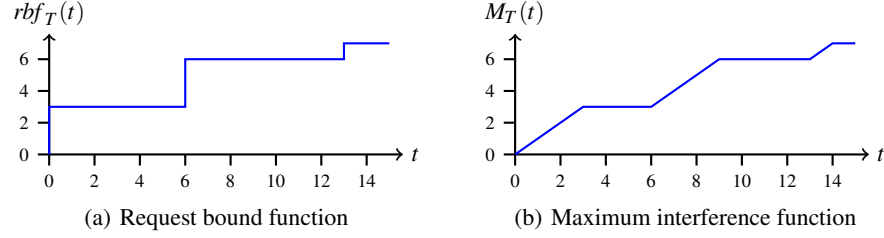
*Static priorities.* An attempt to solve the schedulability problem for GMF in the case of static priorities was presented by Takada and Sakamura [53]. The idea is to use a function called *maximum interference function (MIF)*  $M(t)$ . It is based on the *request bound function rbf(t)* which for each interval size  $t$  counts the accumulated execution demand of jobs that can be *released* inside any interval of that size. (Notice that in contrast, the demand bound function also requires the job deadline to be inside the interval.) The MIF is a “smoother” version of that, which for each task only accounts for the execution demand that could actually execute inside the interval. We show examples of both functions in Figure 4.

The method uses the MIF as a generalization in the  $\Sigma$ -summation term in Equation (4), leading to a generalized recurrence relation for computing the response time:

$$R = E_i + \sum_{j < i} M_j(R) \quad (8)$$

Note that this expresses the response time of a *job* generated by one particular *frame*  $i$  of a GMF task with  $M_j(t)$  expressing the corresponding maximum interference func-





**Fig. 4.** Examples for request bound function and maximum interference function of the same task.

tions of higher priority tasks. Computation of  $M_i(t)$  is essentially the same process as determining the demand bound function  $dbf(t)$  from above.

It was discovered by Stigge and Yi in [52] that the proposed method does *not* lead to a precise test since the response time computed by solving Equation (8) is overapproximate. The reason is that  $M_i(t)$  overapproximates the actual interference caused by higher priority tasks. They give an example in which case the test using (8) determines a task set to be unschedulable while none of the concrete executions would lead to a deadline miss.

Stigge and Yi show in [52] that this is inherent by demonstrating that one single integer-valued function on the time interval domain can not adequately capture the information needed to compute exact response times. Different concrete task sets with different resulting response times need to be abstracted by identical functions, ruling out a precise test. Indeed, they show that the problem of an exact schedulability test for GMF tasks in case of static priority schedulers is strongly coNP-hard implying that there is no adequate replacement for  $M_i(t)$ . The rather involved proof is mostly focussing on a more expressive task model (DRT, see Section 4.5) but is shown to even hold in the GMF case. Still, the test for GMF presented in [53] is a *sufficient* test of pseudo-polynomial time complexity.

### 3.4 Non-cyclic GMF

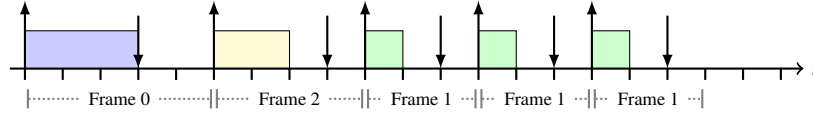
The original motivation for Multiframe and GMF task models was systems consisting of frames with different computational demand and possibly different deadlines and inter-release separation times, arriving in a pre-defined pattern. Consider again the MPEG video codec example where video frames of different complexity arrive, leading to applicability of the Multiframe model. For the presented analysis methods, the assumption of a pre-defined release pattern is fundamental. Consider now a system where the pattern is not known *a priori*, for example if the video codec is more flexible and allows different types of video frames to appear adaptively, depending on the actual video contents. Similar situations arise in cases where the frame order depends on other environmental decisions, e.g. user input or sensor data. These systems can not be modeled with the GMF task model.

Moyo et al. propose in [54] a model called *Non-Cyclic GMF* to capture such behavior adequately. A Non-Cyclic GMF task  $T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$  is syntactically identical to GMF

task from Section 3.3, but with *non-cyclic semantics*. In order to define the semantics formally, let  $\phi : \mathbb{N} \rightarrow \{0, \dots, k-1\}$  be a function choosing frame  $\phi(i)$  for the  $i$ -th job of a job sequence. Having  $\phi$ , each job  $J_i = (r_i, e_i, d_i)$  in a job sequence  $\rho = (J_1, J_2, \dots)$  generated by a non-cyclic GMF task  $T$  needs to correspond to frame  $\phi(i)$  and the corresponding values in all three vectors:

1.  $r_{i+1} \geq r_i + P_{\phi(i)}$
2.  $e_i \leq E_{\phi(i)}$
3.  $d_i = r_i + D_{\phi(i)}$

This contains cyclic GMF job sequences as the special case where  $\phi(i) = (a + i) \bmod k$  for some offset  $a$ . An example of non-cyclic GMF semantics is shown in Figure 5.



**Fig. 5.** *Non-cyclic semantics of the GMF example Figure 3*

For analysing non-cyclic GMF models, Moyo et al give in [54] a simple density-based sufficient feasibility test. Defining  $D(T) := \max_i C_i^T / D_i^T$  as the *density* of a task  $T$ , a task set  $\tau$  is schedulable if  $\sum_{T \in \tau} D(T) \leq 1$ . This generalizes a similar test for the sporadic task model with explicit deadlines. In addition to this test, [54] also includes an exact feasibility test based on efficient systematic simulation.

A different exact feasibility test is presented in [13] for constrained deadlines using the demand bound function condition (1). A dynamic programming approach is used to compute demand pairs (see Section 3.3) based on the observation that  $dbf_T(t)$  can be computed for larger and larger  $t$  reusing earlier values. More specifically, a function  $A_T(t)$  is defined which denotes for an interval size  $t$  the accumulated execution demand of any job sequence where jobs have their full exclusion window inside the interval. It is shown that  $A_T(t)$  for  $t > 0$  can be computed by assuming that some frame  $i$  was the last one in a job sequence contributing a value to  $A_T(t)$ . In that case, the function value for the remaining job sequence is added to the execution time of that specific frame  $i$ . Since frame  $i$  is not known a priori, the computation has to take the maximum over all possibilities. Formally,

$$A_T(t) = \max_i \{A_T(t - P_i^T) + E_i^T \mid P_i^T \leq t\}. \quad (9)$$

Using this,  $dbf_T(t)$  can be computed via the same approach by maximising over all possibilities of the last job in a sequence contributing to  $dbf_T(t)$ . It uses that the execution demand of the remaining job sequence is represented by function  $A_T(t)$ , leading to

$$dbf_T(t) = \max_i \{A_T(t - D_i^T) + E_i^T \mid D_i^T \leq t\}. \quad (10)$$

This leads to a pseudo-polynomial time bound for the feasibility test if  $U(\tau)$  is bounded by a constant, since  $dbf(t) > t$  implies  $t < (\sum_{T,i} E_i^T) / (1 - U(\tau))$  which is pseudo-polynomial in this case.

The same article also proves that evaluating the demand bound function is a (weakly) NP-hard problem. More precisely: Given a non-cyclic GMF task  $T$  and two integers  $t$  and  $B$  it is coNP-hard to determine whether  $dbf_T(t) \leq B$ . The proof is via a rather straight-forward reduction from the Integer Knapsack problem. Thus, a polynomial algorithm for computing  $dbf(t)$  is unlikely to exist.

*Static priorities.* A recent result by Bertin and Goossens [18] proposes a sufficient schedulability test for static priorities. It is based on the request bound function similar to [53] and its efficient computation. Similar to the approach in [53] the function is inherently overapproximate and the test is of pseudo-polynomial time complexity.

## 4 Graph-oriented Models

The more expressive workload models become, the more complicated structures are necessary to describe them. In this section we turn to models based on different classes of directed graphs. We start by recasting the definition of GMF in terms of a graph release structure.

### 4.1 Revisiting GMF

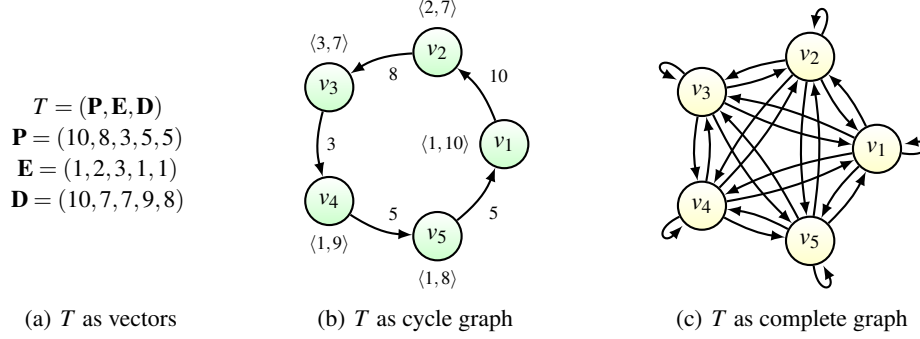
Recall the Generalized Multiframe task model from Section 3.3. A GMF task  $T = (\mathbf{P}, \mathbf{E}, \mathbf{D})$  consists of three vectors for minimum inter-release separation times, worst-case execution times and relative deadlines of  $k$  frames. The same structure can be imagined as a *directed cycle graph*<sup>1</sup>  $G = (V, E)$  in which each vertex  $v \in V$  represents the release of a job and each edge  $(v, v') \in E$  represents the corresponding inter-release separation. A vertex  $v$  is associated with a pair  $\langle e(v), d(v) \rangle$  for WCET and deadline parameters of the represented jobs. An edge  $(u, v)$  is associated with a value  $p(u, v)$  for the inter-release separation time.

The cyclic graph structure directly visualizes the cyclic semantics of GMF. In contrast, non-cyclic GMF can be represented by a *complete digraph*. Figure 6 illustrates the different ways of representing a GMF task with both semantics.

### 4.2 Recurring Branching Tasks

A first generalization to the GMF model was presented in [11]. It is based on the observation that real-time code may include *branches* that influence the pattern in which jobs are released. As the result of some branch, a sequence of jobs may be released which may differ from the sequence released in a different branch. In a schedulability analysis, none of the branches may be universally worse than the others since that may depend on the situation, e.g., which tasks are being scheduled together with the branching one.

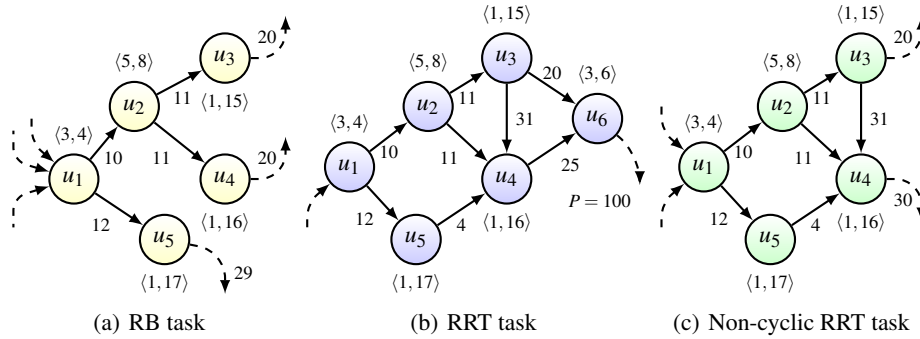
<sup>1</sup> A cycle graph is a graph consisting of one single cycle, i.e., one closed chain.



**Fig. 6.** Different ways of representing a GMF task  $T$ . The vector-representation in 6(a) from Section 3.3 does by itself not imply cyclic or non-cyclic semantics. This is more clear with graphs in 6(b) and 6(c). Note that we omit vertex and edge labels in 6(c) for clarity.

Thus, all branches need to be modeled explicitly and a proper representation is needed, different from the GMF release structure.

A natural way of representing branching code is a *tree*. Indeed, the model proposed in [11] is a tree representing job releases and their minimum inter-release separation times. We show an example in Figure 7(a). Formally, a *Recurring Branching (RB)* task  $T$  is a directed tree  $G(T) = (V, E)$  in which, as in Section 4.1, each vertex  $v \in V$  represents a type of job to be released and each edge  $(u, v) \in E$  the minimum inter-release separation times. They have labels  $\langle e(v), d(v) \rangle$  and  $p(u, v)$  as before. In addition to the tree, each leaf  $u$  has a separation time  $p(u, v_{root})$  to the root vertex  $v_{root}$  in order to model that the behavior *recurs* after each traversal of the tree.



**Fig. 7.** Examples for RB, RRT and non-cyclic RRT task models

In order to simplify the feasibility analysis, the model is syntactically restricted in the following way. For each path  $\pi = (v_0, \dots, v_l)$  of length  $l$  from the root  $v_0 = v_{root}$  to a leaf  $v_l$ , its duration when going back to the root must be the same, i.e., the value  $P := \sum_{i=0}^{l-1} p(v_i, v_{i+1}) + p(v_l, v_{root})$  must be independent of  $\pi$ . We call  $P$  the *period* of  $T$ . Note that this is a generalization of GMF since GMF can be expressed as a linear tree.

*Semantics.* A job sequence  $\rho = (J_1, J_2, \dots)$  is generated by an RB task  $T$  if it corresponds to a path  $\pi$  through  $G(T)$  in the following way. Path  $\pi$  starts at some vertex  $v_1$  in  $G(T)$ , follows the edges to a leaf, then starts again at the root vertex, traverses  $G(T)$  again in a possibly different way, etc. (Very short  $\pi$  may of course never reach a leaf.) The correspondence between  $\rho$  and  $\pi$  means that for all  $J_i = (r_i, e_i, d_i)$ , we have:

1.  $r_{i+1} \geq r_i + p(v_i, v_{i+1})$ ,
2.  $e_i \leq e(v_i)$ ,
3.  $d_i = r_i + d(v_i)$ .

*Feasibility Analysis.* The analysis presented in [11] is based on the concept of *demand pairs* as described before. We sketch the method from [11] in a slightly adjusted manner. First, a set  $DP_0$  is created consisting of all demand pairs corresponding to paths not containing both a leaf and a following root vertex. This is straight forward since for each pair of vertices  $(u, v)$  in  $G(T)$  connected by a directed path  $\pi$ , this connecting path is unique. Thus, a demand pair  $\langle e, d \rangle$  can be created by enumerating all vertex pairs  $(u, v)$  and computing for their connecting path  $\pi = (v_0, \dots, v_l)$  the values

$$e := \sum_{i=0}^l e(v_i), \quad d := \sum_{i=0}^{l-1} p(v_i, v_{i+1}) + d(v_l). \quad (11)$$

Second, all paths  $\pi$  which *do* contain both a leaf and a following root vertex can be cut into three subpaths  $\pi_{head}$ ,  $\pi_{middle}$  and  $\pi_{tail}$ :

$$\pi = (\underbrace{v_1, \dots, v_l}_{\pi_{head}}, \underbrace{v_{root}, v'_1, \dots, v'_l, v_{root}, v''_1, \dots, v''_l}_{\pi_{middle}}, \underbrace{v_{root}, v'''_1, \dots, v'''_l}_{\pi_{tail}})$$

We use  $v_l, v'_l$ , etc. for arbitrary leaf nodes. The first part  $\pi_{head}$  is the prefix of  $\pi$  up to and including the first leaf in  $\pi$ . The second part  $\pi_{middle}$  is the middle part starting with  $v_{root}$  and ending in the last leaf which  $\pi$  visits. Note that  $\pi_{middle}$  may traverse the tree several times. The third part  $\pi_{tail}$  starts with the last occurrence of  $v_{root}$  in  $\pi$ . For each of the three parts, a data structure is created so that demand pairs for a full path  $\pi$  can be assembled easily.

For representing  $\pi_{head}$ , a set  $UP_{leaf}$  is created. For all paths  $\pi_{head} = (v_0, \dots, v_l)$  that end in a leaf it contains a pair  $\langle e, p \rangle$  with

$$e := \sum_{i=0}^l e(v_i), \quad p := \sum_{i=0}^{l-1} p(v_i, v_{i+1}) + p(v_l, v_{root}). \quad (12)$$

For representing  $\pi_{middle}$ , the maximal accumulated execution demand  $e_{max}$  of any path completely traversing the tree is computed. Note that all paths from the root to a leaf

have the same sum of inter-release separation times and this sum is the period  $P$  of  $T$ . Finally, for representing  $\pi_{tail}$ , a set  $DP_{root}$  is computed as a subset of  $DP_0$  only considering paths starting at  $v_{root}$ .

Using these data structures,  $dbf_T(t)$  can be computed easily. If  $t \leq P$ , then a job sequence contributing to  $dbf_T(t)$  either corresponds to a demand pair in  $DP_0$  (not passing  $v_{root}$ ) or is represented by items from  $UP_{leaf}$  and  $DP_{root}$  (since it is passing  $v_{root}$  exactly once):

$$F_1(t) = \max \{e \mid \langle e, d \rangle \in DP_0 \text{ with } d \leq t\} \quad (13)$$

$$F_2(t) = \max \{e_1 + e_2 \mid \langle e_1, p \rangle \in UP_{leaf} \wedge \langle e_2, d \rangle \in DP_{root} \wedge p + d \leq t\} \quad (14)$$

$$dbf_T(t) = \max \{F_1(t), F_2(t)\} \quad \text{if } t \leq P \quad (15)$$

In case  $t > P$ , such a job sequence *must* pass through  $v_{root}$  and traverses the tree completely for either  $\lfloor t/P \rfloor$  or  $\lfloor t/P \rfloor - 1$  times. For the parts that can be represented by  $\pi_{head}$  and  $\pi_{tail}$  of the corresponding path  $\pi$ , we can use  $dbf_T(t)$  from Equation (15), since  $\pi_{head}$  concatenated with  $\pi_{tail}$  correspond to a job sequence without a complete tree traversal. Putting it together for  $t > P$ :

$$F_3(t) = \left\lfloor \frac{t}{P} \right\rfloor \cdot e_{\max} + dbf_T(t \bmod P) \quad (16)$$

$$F_4(t) = \left\lfloor \frac{t-P}{P} \right\rfloor \cdot e_{\max} + dbf_T((t-P) \bmod P) \quad \text{if } t \geq P, 0 \text{ otherwise} \quad (17)$$

$$dbf_T(t) = \max \{F_3(t), F_4(t)\} \quad \text{if } t > P \quad (18)$$

Finally, in order to do the feasibility test, i.e., verify Condition (1), the demand bound function  $dbf(t) = \sum_T dbf_T(t)$  is computed for all  $t$  up to a bound  $D$  derived in a similar way as for GMF in Section 3.3.

### 4.3 Recurring Real-Time Tasks (RRT) – DAG structures

In typical branching code, the control flow is joined again after the branches are completed. Thus, no matter which branch is taken, the part after the join is common to both choices. In the light of a tree release structure as in the RB task model above, this means that many vertices in the tree may actually represent the same types of jobs to be released, or even whole subtrees are equal. In order to make use of these redundancies, Baruah proposes in [10] to use a *directed acyclic graph (DAG)* instead of a tree. The impact is mostly efficiency: each DAG can be unrolled into a tree, but that comes at the cost of potentially exponential growth of the graph.

A *Recurring Real-Time Task (RRT)*  $T$  is a directed acyclic graph  $G(T)$ . The definition is very similar to RB tasks in the previous section, we only point out the differences. It is assumed that  $G(T)$  contains one unique *source* vertex (corresponding to  $v_{root}$  in an RB task) and further one unique *sink* vertex (corresponding to leafs in a tree). An RRT task has an explicitly defined period parameter  $P$  that constrains the minimum time between two releases of jobs represented by the source vertex. An RRT behaves just like an RB task by following paths through the DAG. We skip the details and give an example of an RRT task in Figure 7(b).

*Feasibility Analysis.* Because of its close relation to RB tasks, the feasibility analysis method presented in [10] is very similar to the method presented above for RB tasks and we skip the details. However, the adapted method has exponential complexity since it enumerates paths explicitly.

Chakraborty et al. present a more efficient method in [21] based on a dynamic programming approach, leading back to pseudo-polynomial complexity. Instead of enumerating all pairs of vertices  $(u, v)$  in the DAG, the graph is traversed in a breadth-first manner. The critical observation is that all demand pairs representing a path ending in any particular vertex  $v$  can be computed from those for paths ending in all parent vertices. It is *not* necessary to have precise information about which the actual paths are that the demand pairs represent. Even though Chakraborty et al. consider a limited variant of the model in which paths traverse the DAG only once, the ideas can be applied in general to the full RRT model.

The feasibility problem is further shown to be NP-hard in [21] via a reduction from the Knapsack problem and the authors give a fully polynomial time approximation scheme. For the special case where all vertices have equal WCET annotations, they show a polynomial time solution, similar to the dynamic programming technique above.

*Static Priorities.* A sufficient test for schedulability of an RRT task set with static priorities is presented in [12]. It is shown that, up to a polynomial factor, the *priority assignment* problem in which a priority order has to be found is equivalent to the *priority testing* problem where a task set with a given priority order is to be tested for schedulability. At the core of both is the test whether a given task  $T \in \tau$  will meet its deadlines if it has the lowest priority of all tasks in  $\tau$  (whose relative priority order does not matter). In that case  $T$  is called *lowest-priority feasible*.

The proposed solution gives a condition involving both the demand bound function  $dbf_T(t)$  and the request bound function  $rbf_T(t)$ . It is shown that a task  $T$  is lowest-priority feasible if

$$\forall t. \exists t' \leq t. t' \geq dbf_T(t) + \sum_{T' \in \tau \setminus \{T\}} rbf_{T'}(t'). \quad (19)$$

It is shown that  $rbf_T(t)$  can be computed with just a minor modification to the computation procedure of  $dbf_T(t)$  and that Condition (19) only needs to be checked for a bounded *testing set* of  $t$ , similar to the bound  $D$  introduced in checking Condition (1) in feasibility tests. For each  $t$ , checking the existence of a  $t' \leq t$  is essentially identical to an iterative procedure of solving the recurrence relation in Equation (8) of which (19) is a generalization.

A tighter and more efficient test is shown in [21] based on a smoother variant of the request bound function, denoted  $rbf'_T(t)$ . Using this, a task  $T$  is lowest-priority feasible if

$$\forall v \in G(T). \exists t' \leq d(v). t' \geq e(v) + \sum_{T' \in \tau \setminus \{T\}} rbf'_{T'}(t'). \quad (20)$$

This test is a more direct and tighter generalization of the sufficient test (8) for GMF tasks.

#### 4.4 Non-cyclic RRT

A further generalization of RRT is *non-cyclic RRT*<sup>2</sup> [14] where the assumption of one single sink vertex is removed. Specifically, a non-cyclic RRT task  $T$  is a DAG  $G(T)$  with vertex and edge labels as before that has a unique source vertex  $v_{source}$ . Additionally, for every sink vertex  $v$ , there is a value  $p(v, v_{source})$  as before. We give an example in Figure 7(c). Note that a non-cyclic RRT task does *not* have a general period parameter, i.e., paths through  $G(T)$  visiting  $v_{source}$  repeatedly may do so in differing time intervals when doing so through different sinks.

*Feasibility Analysis.* The analysis technique presented in [14] is similar to the ones of RB and RRT. The author uses the dynamic programming technique from [21] to compute demand pairs inside the DAG in order to keep pseudo-polynomial complexity and assumes a partition of paths  $\pi$  into  $\pi_{head}$ ,  $\pi_{middle}$  and  $\pi_{tail}$  as before. The difference here is that paths traversing  $G(T)$  completely from  $v_{source}$  to a sink may have different lengths, i.e.,  $\pi_{middle}$  is not necessarily a multiple of some period  $P$ . Thus, the expressions for partial  $dbf_T(t)$  computation in (16) and (17) can't just assume a fixed length  $P$  and a fixed computation time  $e_{max}$ . The idea to solve this is to first use the technique from [21] to compute demand pairs for full DAG traversals. These can then be interpreted as *frames* with a length and an execution time requirement, which can be concatenated to achieve a certain interval length, like a very big non-cyclic GMF task. Similar to (9) for solving non-cyclic GMF feasibility, all possible paths going from source to a sink can be represented in a function  $A_T(t)$  that expresses for each  $t$  the amount of execution demand these special paths may create during intervals of length  $t$ . Similar to (10), this function is integrated into (16) and (17), resulting in an efficient procedure.

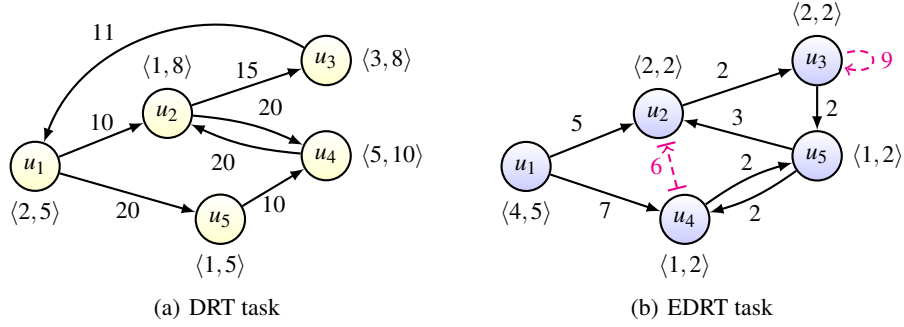
The procedure is generalized in the following section which generalizes and unifies all feasibility tests presented so far.

#### 4.5 Digraph Real-Time Tasks

Stigge et al. observe in [50] that the non-cyclic RRT model can be generalized to *any* directed graph. They introduce the *Digraph Real-Time (DRT)* task model and describe a feasibility test of pseudo-polynomial complexity for task systems with utilization bounded by a constant. A DRT task  $T$  is described by a directed graph  $G(T)$  with edge and vertex labels as before. There are no further restrictions, any directed graph can be used to describe a task. Using any graph allows to model local loops which was not possible in any model presented above. Even in the non-cyclic RRT model, all cycles in that model have to pass through the source vertex. An example of a DRT task is shown in Figure 8(a).

<sup>2</sup> The name “non-cyclic RRT” can be a bit misleading. The behavior of a non-cyclic RRT task *is* cyclic, in the sense that the source vertex is visited repeatedly. However, in comparison to the RRT model, the behavior is *non-periodic*, in the sense that revisits of the source vertex may happen in different time intervals.





**Fig. 8.** Examples for DRT and EDRT task models. The EDRT task in Figure 8(b) contains two additional constraints  $(u_4, u_2, 6)$  and  $(u_3, u_2, 9)$ , denoted with dashed arrows. Note that these dashed arrows do *not* represent edges that can be taken. They only denote additional timing constraints.

*Semantics.* The behavior of a DRT task  $T$  is similar to earlier models. A job sequence  $\rho = (J_1, J_2, \dots)$  is generated by  $T$  if there is a path  $\pi = (v_1, v_2, \dots)$  through  $G(T)$  such that for each job  $J_i = (r_i, e_i, d_i)$  it holds that

1.  $r_{i+1} \geq r_i + p(v_i, v_{i+1})$ ,
2.  $e_i \leq e(v_i)$ , and
3.  $d_i = r_i + d(v_i)$ .

Note that this implies sporadic job releases as before. However, worst-case sequences usually release jobs as soon as permitted.

*Feasibility Analysis.* Stigge et al. present in [50] a feasibility analysis method which is again based on demand pairs. However, there is no single vertex through which all paths pass and neither does the graph  $G(T)$  represent a partial order as DAGs do. Thus, earlier dynamic programming approaches can not be applied. It is not possible either to simply enumerate all paths through the graph since that would lead to an exponential procedure. Instead, the authors propose a *path abstraction* technique which is essentially also based on dynamic programming, generalizing earlier approaches.

For the case of constrained deadlines, each path  $\pi = (v_0, \dots, v_l)$  through  $G(T) = (V, E)$  is abstracted using a *demand triple*  $\langle e(\pi), d(\pi), v_l \rangle$  with

$$e(\pi) := \sum_{i=0}^l e(v_i), \quad d(\pi) := \sum_{i=0}^{l-1} p(v_i, v_{i+1}) + d(v_l). \quad (21)$$

It contains as first two components a demand pair and the third component is the last vertex in the path. This abstraction allows to create all demand pairs for  $G(T)$  iteratively. The procedure starts with all demand triples that represent paths of length 0, i.e., with the set

$$\{\langle e(v), d(v), v \rangle \mid v \in V\}. \quad (22)$$

In each step, a triple  $\langle e, d, v \rangle$  is picked from the set and extended to create new triples  $\langle e', d', v' \rangle$  via

$$e' := e + e(v'), \quad d' := d - d(v) + p(v, v') + d(v'), \quad (v, v') \in E. \quad (23)$$

This is done for all edges  $(v, v') \in E$ . The procedure abstracts from concrete paths since the creation of each new triple  $\langle e', d', v' \rangle$  does *not* need the information of the full path represented by  $\langle e, d, v \rangle$ . Instead, the last vertex  $v$  of any such path suffices. The authors show that this procedure is efficient since it only needs to be executed *once* up to a bound  $D$  as before and the number of demand triples is bounded.

Further contributions of [50] include an extension of the method to arbitrary deadlines and a few optimisation suggestions for implementations. One of them is considering *critical* demand triples  $\langle e, d, v \rangle$  for which no other demand triple  $\langle e', d', v' \rangle$  exists with

1.  $e' \geq e$ ,
2.  $d' \leq d$ , and
3.  $v' = v$ .

It is shown that only critical demand triples need to be stored during the procedure. All other, non-critical, demand triples can be discarded since they and all their future extensions will be *dominated* by others when considering contributions to the demand bound function  $dbf_T(t)$ . This optimization reduces the cubic time complexity of the algorithm to about quadratic in the task parameters.

The presented method is applicable to *all* models in the previous sections since DRT generalizes all of them. In a few special cases, custom optimizations may speed up the process.

*Static Priorities.* Even though sufficient tests involving  $rbf(t)$  and  $dbf(t)$  similar to Conditions (19) and (20) can be applied to DRT as well, no exact schedulability test for DRT with static priorities is currently known. It was shown in [52] that the problem is strongly NP-hard via a reduction from the 3-PARTITION problem. This implies that an exact test with pseudo-polynomial complexity is impossible, assuming  $P \neq NP$ . Further, even fully polynomial-time approximation schemes can not exist either. The result holds for all models at least as expressive as GMF.

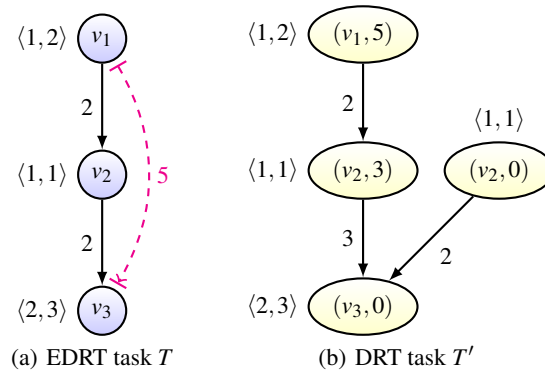
#### 4.6 Global Timing Constraints

In an effort to investigate the border of how far graph-based workload models can be generalized before the feasibility problem becomes infeasible, Stigge et al. propose in [49] a task model called *Extended DRT (EDRT)*. In addition to a graph  $G(T)$  as in the DRT model, a task  $T$  also includes a set  $C(T) = \{(from_1, to_1, \gamma_1), \dots, (from_k, to_k, \gamma_k)\}$  of *global inter-release separation constraints*. Each constraint  $(from_i, to_i, \gamma_i)$  expresses that between the visits of vertices  $from_i$  and  $to_i$ , at least  $\gamma_i$  time units must pass. An example is shown in Figure 8(b).

*Feasibility Analysis.* The feasibility analysis problem for EDRT indeed marks the tractability borderline. In case the number of constraints in  $C(T)$  is *bounded* by a constant  $k$  which is a restriction called  $k$ -EDRT, [49] presents a pseudo-polynomial feasibility test. If the number is not bounded, they show that feasibility analysis becomes strongly NP-hard by a reduction from the Hamiltonian Path problem, ruling out pseudo-polynomial methods.

For the bounded case, we illustrate why the iterative procedure described in Section 4.5 for DRT can not be applied directly and then sketch a solution approach. The demand triple method can not be applied without change since the abstraction loses too much information from concrete paths. In the presence of global constraints, the procedure needs to keep track of which constraints are *active*. Consider path  $\pi = (u_4, u_5)$  from the example in Figure 8(b), which would be abstracted with demand triple  $\langle 2, 4, u_5 \rangle$ . An extension with  $u_2$  leading to demand triple  $\langle 4, 7, u_2 \rangle$  is *not* correct, since the path  $\pi' = (u_4, u_5, u_2)$  includes a global constraint, separating releases of the jobs associated with  $u_4$  and  $u_2$  by 6 time units. A correct abstraction of  $\pi'$  would therefore be  $\langle 4, 8, u_2 \rangle$ , but it is impossible to construct that from  $\langle 2, 4, u_5 \rangle$  which lost information about the active constraint.

A solution to this problem is to integrate information about active constraints into the analysis. For each constraint  $(from_i, to_i, \gamma_i)$ , the demand triple is extended by a *countdown* that represents how much time must pass until  $to_i$  may be visited again. This slows down the analysis, but since the number of constraints is bounded by a constant, the slowdown is only a polynomial factor. Stigge et al. choose in [49] to not directly integrate the countdown information into the iterative procedure but to transform each EDRT task  $T$  into a DRT task  $T'$  where the countdown information is integrated into the vertices. The transformation leads to an equivalent iterative procedure and has the advantage that an already existing graph exploration engine for feasibility tests of DRT task sets can be reused without any change. The transformation is illustrated in Figure 9.



**Fig. 9.** A basic example of a 1-EDRT task  $T$  being transformed into an equivalent DRT task  $T'$ .

## 5 Beyond DRT

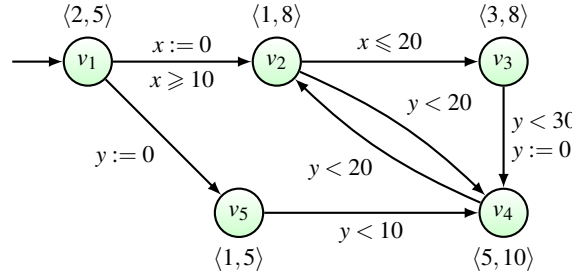
We now turn to models that either extend the DRT model in other directions or are outside the hierarchy in Figure 1 since they operate on a different abstraction level.

### 5.1 Task Automata

A very expressive workload model called task automata is presented by Fersman et al. in [30]. It is based on Timed Automata that have been studied thoroughly in the context of formal verification of timed systems [1, 17]. Timed automata are finite automata extended with real-valued clocks to specify timing constraints as enabling conditions, i.e., guards on transitions. The essential idea of task automata is to use the timed language of an automaton to describe task release patterns.

In DRT terms, a task automaton (TA)  $T$  is a graph  $G(T)$  with vertex labels as in the DRT model, but labels on edges are more expressive. An edge  $(u, v)$  is labeled with a *guard*  $g(u, v)$  which is a boolean combination of clock comparisons of the form  $x \bowtie C$  where  $C$  is a natural number and  $\bowtie \in \{\leq, <, \geq, >\}$ . Further, an edge may be labeled with a *clock reset*  $r(u, v)$  which is a set of clocks to be reset to 0 when this edge is taken. Since the value of clocks is an increasing real value which represents that time passes, guards and resets can be used to constrain timing behavior on generated job sequences. We give an example of a task automaton in Figure 10.

A task automaton has an initial vertex. In addition to resets and guards, task automata have a synchronization mechanism. This allows them to synchronize on edges either with each other or with the scheduler on a job finishing time.



**Fig. 10.** Example for a task automaton.

Note that DRT tasks are special cases of task automata where only one clock is used. Each edge  $(u, v)$  in a DRT task with label  $p(u, v) = C$  can be expressed with an edge in a task automaton with guard  $x \geq C$  and reset  $x := 0$ .

The authors of [28] show that the schedulability problem for a large class of systems modeled as task automata can be solved via a reduction to a model checking problem for ordinary Timed Automata. A tool for schedulability analysis of task automata is

presented in [2, 29]. In fact, the feasibility problem is decidable for systems where at most two of the following conditions hold:

**Preemption.** The scheduling policy is preemptive.

**Variable Execution Time.** Jobs may an interval of possible execution times.

**Task Feedback.** The finishing time of a job may influence new job releases.

However, the feasibility problem is undecidable if all three conditions are true [28]. Task automata therefore mark a borderline between decidable and undecidable problems for workload models.

## 5.2 Real-Time Calculus

A more abstract formalism to model real-time workload is the *Real-Time Calculus* (RTC) introduced in [55]. Its abstractions are based on the notions of

**Event streams**, representing the workload,

**Computing capacity**, representing the computing resource for the workload, and

**Processing units**, modeling the process of the workload being executed, using up computing capacity.

Formally, a request function  $R(t)$  models the accumulated amount of workload up to a time point  $t$ , and similarly, a capacity function  $C(t)$  models the amount of computation resource available until time  $t$ . Both functions are the inputs to a processing unit, which outputs a new event stream modeled by a function  $R'(t)$ , and leaves remaining computing capacity modeled by  $C'(t)$ . The four functions are related by

$$R'(t) = \min_{0 \leq u \leq t} \{R(u) + C(t) - C(u)\}, \quad (24)$$

$$C'(t) = C(t) - R(t). \quad (25)$$

Modeling static priority scheduling is rather straight forward in this model by creating a number of processing units which all have their own input of request functions (representing different tasks) but all being chained through the corresponding capacity functions, i.e.,  $C'(t)$  of one unit being  $C(t)$  of the next one. Other schedulers can also be modeled, with more involved constructions.

A concrete system may create many different request functions  $R(t)$ . In order to represent these, Real-Time Calculus considers upper and lower bounds in the time interval domain. Specifically, for abstracting the request function  $R(t)$ , two *arrival curves*  $\alpha^u(\Delta)$  and  $\alpha^l(\Delta)$  are introduced, so that for all  $t \geq 0$  and  $\Delta \geq 0$

$$\alpha^l(\Delta) \leq R(t + \Delta) - R(t) \leq \alpha^u(\Delta). \quad (26)$$

Clearly, a pair  $(\alpha^u, \alpha^l)$  abstracts an infinite set of event streams. Analogously, a pair of *service curves*  $(\beta^u, \beta^l)$  is defined in the time domain to abstract computing capacity.

All analysis can be executed on the level of arrival and service curves. For example, the remaining service curve can be computed via

$$\beta''(\Delta) = \max_{0 \leq u \leq \Delta} \{\beta^l(u) - \alpha^u(u)\}. \quad (27)$$

Response-time analysis can be performed by deriving the response time from the horizontal distance between  $\alpha''$  and  $\beta^l$ .

For many classes of systems, arrival curves can be specified directly with closed form expressions. This includes the sporadic task model with an expression similar to (2), but is also possible for related models like periodic events with jitter [46].

It is worth noting that arrival curves and the request bound function introduced earlier are equivalent concepts. However, RTC models systems are directly using arrival curves, i.e., they are the low-level construct used for system representation. In contrast, the request bound function (and similarly, the demand bound function) is a tool for schedulability analysis that abstracts from a more concrete system description. It is a potentially overapproximate abstraction and the process of deriving request and demand bound functions for task models presented in Sections 3 and 4 is more or less computationally demanding. Thus, the advantage of RTC is that arrival curves can be assumed as given input to the analysis procedures, at the cost of introducing imprecision.

## 6 Conclusions and Outlook

This survey is by no means complete. In preparing this paper, we have intended to cover only works on independent tasks for preemptive uniprocessors. Other sources for a survey on related topics can be found in [5] on the broad area of static-priority scheduling and analysis, [8] on scheduling and analysis of repetitive models for preemptable uniprocessors, [22] on real-time scheduling for multiprocessors and [48] on the historic development of real-time systems research in general. To complement the survey provided by [8], we have added recent work on models with richer structures, that are based on graphs and automata as well as the Real-Time Calculus based on functions. Apart from their importance in theoretical studies, we believe that these expressive models may find their applications in for example model- and component-based design for timed systems.

The following is a list of areas that we consider important, but did not include them in this paper due to time and page limits.

*Resource Sharing.* Real-time systems even on uniprocessor platforms contain not only the execution processor, but often many resources shared by concurrently running jobs using a locking mechanism during their execution. Locking can introduce unwanted effects like deadlocks and priority inversion. For periodic and sporadic task systems, the Priority Ceiling (and Inheritance) Protocols [47] and Stack Resource Policy [6] are established solutions with efficient schedulability tests. While these protocols can be used for more expressive models like DRT, the analysis methods do not necessarily apply. Some initial work exists on extending the classical results to graph-based models, e.g., [31] which develops a new protocol to deal with branching structures. The non-deterministic behaviours introduced by DRT-like models are not well understood in the context of resource sharing.

*Real-Time Multiprocessor Scheduling.* In contrast to uniprocessor systems, scheduling real-time workloads on parallel architectures is a much harder challenge. There are

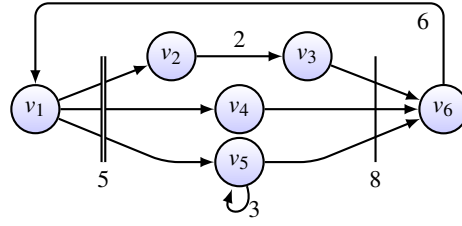
various scheduling strategies, e.g., global and partition-based scheduling with mostly sufficient conditions for schedulability tests. A comprehensive survey on this topic is found in [22]. The introduction of multicore platforms adds another dimension of complexity due to on-chip resource contention. The known techniques for multiprocessor scheduling all rely on a safe WCET bound of tasks under idealized assumptions on the underlying platform. For multicore platforms, without proper isolation, it seems impossible to achieve WCET bounds for tasks running in parallel on different processor cores. To the best of our knowledge, there is no work on bridging WCET analysis and multiprocessor scheduling.

*Mixed-Criticality Systems.* Integrating applications of different levels of criticality on the same processor chip is attracting increasing interest. Scheduling mixed-criticality workloads on uniprocessors has been studied intensively in recent years [15, 16, 26]. For multiprocessor systems, it is still an open area for research; a seminal work is found in [39]. Due to their potentially high computing capacity, we believe that multicore processors are more adequate for such mixed types of applications. Typically, a mixed criticality system operates in different modes. In the normal mode, the computation power of multicores may be used to improve the average performance of low-criticality applications and in case of exceptions, e.g., a timing error occurs, high-criticality applications should be prioritized. However, it is not well understood how to isolate and avoid interferences among the applications without fully partitioning the on-chip resources.

*Further extensions of workload models.* One may add new features or high-level structures on the existing models for expressiveness and abstraction purposes. We see two interesting directions:

**Fork-Join Real-Time (FJRT) tasks:** Recently, an extension of the DRT task model to incorporate fork/join structures is proposed by Stigge et al. Instead of following just *one* path through the graph, the behavior of a task includes the possibility of *forking* into different paths at certain nodes, and *joining* these paths later. Syntactically, this is represented using hyperedges. A hypergraph generalizes the notion of a graph by extending the concept of an edge between two vertices to hyperedges between two sets of vertices. For details, we refer to [51]. The model is illustrated with an example in Figure 11. For the time of writing this survey, no efficient analysis method for FJRT is known.

**Mode Switches:** A system may operate in different modes demonstrating different timing behaviours. We consider a simple model using general directed graphs where the nodes stand for modes, assigned with a set of tasks to be executed in the corresponding mode, and edges for mode switches that may be triggered by an internal or external event and guarded by a timing constraint such as a minimal separation distance. Mode switching is a concept that has been studied in different contexts. Many protocols for mode switches have been proposed [45] to specify what should be done during the switch. In a recent work [27], the authors show that a mixed criticality task system can be modeled nicely using a chain of modes representing the criticality levels where mode switches are triggered by a task over-run that may



**Fig. 11.** Example FJRT task. The fork edge is depicted with an intersecting double line, the join edge with an intersecting single line. All edges are annotated with minimum inter-release delays  $p(U, V)$ . The vertex labels are omitted in this example. Assuming that vertex  $v_i$  releases a job of type  $J_i$ , a possible job sequence containing jobs with their types and absolute release times is  $\sigma = [(J_1, 0), (J_2, 5), (J_5, 5), (J_4, 6), (J_3, 7), (J_5, 8), (J_6, 16), (J_1, 22)]$ .

occur at any time, and on each mode switch, tasks of lower-level criticality will be dropped and only tasks of higher-level criticality are scheduled to run according to their new task parameters in the new mode. The authors present a technique for scheduling the mixed criticality workload described in directed acyclic graphs. An interesting direction for future work is scheduling of mode switches in general directed graphs, which involves fixed-point computation due to cyclic structures.

*Tools for schedulability analysis.* Over the years, many models and analysis techniques have been developed. It is desirable to have a software tool that as input takes a workload description in some of the models and a scheduling policy and determines the schedulability. A tool for task automata has been developed using techniques for model checking timed automata [2]. Due to the analysis complexity of timed automata, it suffers from the state-explosion problem. For the tractable models including DRT in the hierarchy of Figure 1, a tool for schedulability analysis is currently under development in Uppsala based on the path abstraction technique of [50].

## References

1. Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. TIMES - A Tool for Modelling and Implementation of Embedded Systems. In *Proc. of TACAS 2002*, pages 460–464. Springer-Verlag, 2002.
3. N.C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. In *Proc. of RTOSS 1991*, pages 133–137, 1991.
4. Neil Audsley. On priority assignment in xed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.
5. Neil C Audsley, Alan Burns, Robert I Davis, Ken W Tindell, and Andy J Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2):173–198, 1995.
6. T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Syst.*, 3(1):67–99, April 1991.



7. Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Syst.*, 17(1):5–22, 1999.
8. Sanjoy Baruah and Joël Goossens. Scheduling real-time tasks: Algorithms and complexity. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 3, 2004.
9. Sanjoy K. Baruah, , A.K. Mok, and L.E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proc. of RTSS 1990*, pages 182–190, 1990.
10. Sanjoy K. Baruah. A general model for recurring real-time tasks. In *Proc. of RTSS*, pages 114–122, 1998.
11. Sanjoy K. Baruah. Feasibility Analysis of Recurring Branching Tasks. In *Proc. of EWRTS*, pages 138–145, 1998.
12. Sanjoy K. Baruah. Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks. *Real-Time Systems*, 24(1):93–128, 2003.
13. Sanjoy K. Baruah. Preemptive Uniprocessor Scheduling of Non-cyclic GMF Task Systems. In *Proc. of RTCSA 2010*, pages 195–202, 2010.
14. Sanjoy K. Baruah. The Non-cyclic Recurring Real-Time Task Model. In *Proc. of RTSS 2010*, pages 173–182, 2010.
15. Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D’Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Trans. Computers*, 61(8):1140–1152, 2012.
16. Sanjoy K. Baruah, Alan Burns, and Robert I. Davis. Response-time analysis for mixed criticality systems. In *RTSS*, pages 34–43, 2011.
17. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124, 2003.
18. Vandy Berten and Joël Goossens. Sufficient FTP Schedulability Test for the Non-Cyclic Generalized Multiframe Task Model. *CoRR*, abs/1110.5793, 2011.
19. J.Y.L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science. Springer, 2001.
20. G.C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Realtime Systems. Springer, 2011.
21. Samarjit Chakraborty, Thomas Erlebach, and Lothar Thiele. On the complexity of scheduling conditional real-time code. In *Proc. of WADS 2001*, pages 38–49, 2001.
22. Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.
23. T. Ebenlender, M. Krcal, and J. Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, pages 1–19, 2012. (Extended abstract published in the SODA 2008 Proceedings).
24. Friedrich Eisenbrand and Thomas Rothvoß. Static-priority Real-time Scheduling: Response Time Computation is NP-hard. In *Proc. of RTSS 2008*, pages 397–406, 2008.
25. Friedrich Eisenbrand and Thomas Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proc. of SODA 2010*, pages 1029–1034, 2010.
26. Pontus Ekberg and Wang Yi. Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks. In *ECRTS*, pages 135–144, 2012.
27. Pontus Ekberg and Wang Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. Norwell, MA, USA, 2013. Kluwer Academic Publishers.
28. Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.
29. Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theor. Comput. Sci.*, 354(2):301–317, 2006.
30. Elena Fersman, Paul Pettersson, and Wang Yi. Timed Automata with Asynchronous Processes: Schedulability and Decidability. In *Proc. of TACAS 2002*, pages 67–82. Springer-Verlag, 2002.

31. Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. Resource Sharing Protocols for Real-Time Task Graph Systems. In *Proc. of ECRTS 2011*, pages 272–281, 2011.
32. C.-C. J. Han. A Better Polynomial-Time Schedulability Test for Real-Time Multiframe Tasks. In *Proc. of RTSS*, pages 104–, Washington, DC, USA, 1998. IEEE Computer Society.
33. Mathai Joseph and Paritosh K. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29:390–395, 1986.
34. Edward A. Lee, Stephen Neuendorffer, and Michael J. Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers*, 2, 2003.
35. John P. Lehoczky, Lui Sha, and Jay K. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In *Proc. of RTSS*, pages 261–270, 1987.
36. J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.
37. Joseph Y.-T. Leung and M.L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3):115 – 118, 1980.
38. Joseph Y.-T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237 – 250, 1982.
39. Haohan Li and Sanjoy K. Baruah. Outstanding paper award: Global mixed-criticality scheduling on multiprocessors. In *ECRTS*, pages 166–175, 2012.
40. C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, 1973.
41. Wan-Chen Lu, Kwei-Jay Lin, Hsin-Wen Wei, and Wei-Kuan Shih. New Schedulability Conditions for Real-Time Multiframe Tasks. In *Proc. of ECRTS*, pages 39–50, 2007.
42. A. K. Mok. Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment. Technical report, Cambridge, MA, USA, 1983. Ph.D. thesis.
43. Aloysius K. Mok and Deji Chen. A General Model for Real-Time Tasks. Technical report, 1996.
44. Aloysius K. Mok and Deji Chen. A Multiframe Model for Real-Time Tasks. *IEEE Trans. Softw. Eng.*, 23(10):635–645, 1997.
45. L.T.X. Phan, Insup Lee, and O. Sokolsky. A semantic framework for mode change protocols. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 91 –100, april 2011.
46. Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models*. PhD thesis, Technical University of Braunschweig, Braunschweig, Germany, 2004.
47. L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *Computers, IEEE Transactions on*, 39(9):1175 –1185, sep 1990.
48. Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 28(2-3):101–155, November 2004.
49. Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. On the Tractability of Digraph-Based Task Models. In *Proc. of ECRTS 2011*, pages 162–171, 2011.
50. Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The Digraph Real-Time Task Model. In *Proc. of RTAS 2011*, pages 71–80, 2011.
51. Martin Stigge, Pontus Ekberg, and Wang Yi. The Fork-Join Real-Time Task Model. In *Proc. of ACM SIGBED Review*, 2012. To appear.
52. Martin Stigge and Wang Yi. Hardness Results for Static Priority Real-Time Scheduling. In *Proc. of ECRTS 2012*, pages 189–198, 2012.
53. Hiroaki Takada and Ken Sakamura. Schedulability of Generalized Multiframe Task Sets under Static Priority Assignment. In *Proc. of RTCSA 1997*, pages 80–86, 1997.

54. Noel Tchidjo Moyo, Eric Nicollet, Frederic Lafaye, and Christophe Moy. On Schedulability Analysis of Non-cyclic Generalized Multiframe Tasks. In *ECRTS 2010*, pages 271–278, 2010.
55. L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCAS 2000*, volume 4, 2000.
56. José Verschae and Andreas Wiese. On the configuration-lp for scheduling on unrelated machines. In *ESA*, pages 530–542, 2011.
57. Tei wei Kuo, Li pin Chang, Yu hua Liu, and Kwei jay Lin. Efficient online schedulability tests for real-time systems. *IEEE Transactions On Software Engineering*, 29:734–751, 2003.
58. A. Zuhily and A. Burns. Exact Scheduling Analysis of Non-Accumulatively Monotonic Multiframe Tasks. *Real-Time Systems Journal*, 43:119–146, 2009.

# The Utilization Bound of CAN: Theory and Practice

Eduardo Tovar, Nuno Pereira, Ricardo Gomes

CISTER/INESC TEC Research Unit, ISEP, Polytechnic Institute of Porto, Portugal  
{emt, nap, rftg}@isep.ipp.pt

**Abstract.** The notion of utilization bound is well established in the real-time scheduling literature and there are a number of results that enable this simple and effective schedulability test in many systems. We outline the main results on utilization bounds for message scheduling in the CAN bus and expand previous work by presenting extensive testing in real CAN-enabled platforms. These experiments show the effectiveness of the bound and provide a measure of its pessimism when compared to the average-case behaviour. The bounds discussed in this work are enabled by the fact that the CAN standard imposes limitations on the maximum size of a message and tell us that, if priorities to message streams are assigned using rate-monotonic (RM) and if the requested capacity of the CAN bus does not exceed the bound, then all deadlines are met. For CAN2.0A, the utilization bound is approximately 25% and the utilization bound for CAN2.0B is approximately 29%.

## 1 Introduction

Designers of embedded/real-time systems have been extremely successful deploying distributed systems in industrial control, in-vehicle control systems for avionics and automotive, building management systems and many other. A key component for the successful development of these distributed systems is the communication network, and there are a number of technologies available (such as TTP, FlexRay or PROFIBUS) to meet different application needs. One of the most widely used communication technologies in embedded/real-time systems is the Controller Area Network (CAN) bus [5, 9].

CAN offers very appealing features for distributed systems designers, some examples are: (i) CAN controllers are available at low cost and (ii) can use low-cost cabling; (iii) CAN networks are flexible in the sense that new computer nodes can be connected without changing the configuration of existing computer nodes; (iv) CAN is a fully distributed protocol, supporting redundancy. As a consequence, CAN has been widely deployed and, each year, 400 million CAN-enabled microcontrollers are manufactured [9].

Since several nodes share the CAN bus, the communication delay experienced by one node depends on the traffic generated by other nodes. Lacking better theory to develop their systems, designers of CAN-based systems in the early 1990s often adopted a rule of thumb: the CAN bus should be utilized to at most 30% [9]. Researchers soon observed that the characteristics of the access control performed by CAN allows to build upon existing real-time scheduling theory for static priorities and derive a scheduling

theory for the CAN bus that determines the individual maximum delays of message transmissions on CAN [26]. This allowed designers to build systems with much higher utilizations and a utilization of 80% of the CAN bus became common [9].

In this paper, we present a test to determine if a given set of messages can be transmitted in a CAN bus within the assigned deadlines. This test does not compute the delays of each message. Instead, it is based on the well-known notion of *utilization bound*. The utilization bound of a scheduling algorithm is the maximum requested utilization of the resource such that all deadlines are met.

Knowing the utilization bound of a real-time scheduling algorithm is important. Utilization-based tests provide a scalable and robust tool to determine the schedulability of a system. Based on the fundamentals of resource usage, they provide an intuitive performance metric that a system designer can use to, at design time, rapidly access and reason on the schedulability of a system without having to compute each individual delay. Given that we know that the parameters of a message set can be modified without compromising the system schedulability as long as we remain below the utilization bound, they provide some stability to the system in face of dynamic changes. For example in embedded applications that need to dynamically reconfigure and perform, at runtime, schedulability tests that take into account the current workload.

The utilization bounds for non-preemptive static-priority scheduling on a CAN bus presented here assume rate-monotonic (RM) priority assignment, with implicit deadlines and are possible thanks to the fact that the CAN standard requires that the data payload of a message be at most 8 bytes. Consequently, only messages that comply with that requirement need to be considered. In this paper, we extend our previous work where this test was initially developed [3], to present further related work, motivation for the relevance of the result and provide a report on extensive experiments performed. The experiments reported in this paper included many experiment runs lasting for several weeks, comprehensively testing the utilization bound in real-world CAN platforms and showing a measure of its pessimism.

The remainder of this paper is organized as follows. Section 2 provides some context to this work and overviews the relevant related work. Sections 6 and 5 review the utilization bound of CAN developed in previous work [3]. Section 7 studies the utilization of real-world CAN networks, and Section 8 provides some closing remarks.

## 2 Context and Related Work

The CAN bus was introduced in the mid 1980s, by Bosch [5] for the purpose of replacing dedicated wires in vehicles with a shared communication bus. Data transmission in CAN is based on the principle of performing a bitwise arbitration, invented earlier in the 1970s [19, 21].

The medium access control protocol used in the CAN bus implements non-preemptive static-priority scheduling [12, 26]. The latter implies that efficient real-time scheduling techniques (such as RM) can be used and they can be analyzed using real-time scheduling theory for systems using non-preemptive scheduling [9, 12]. This scheduling theory is particularly useful for so-called *sporadic message streams*, that is, message streams where the exact time of an individual messages transmission request is unknown, but

the minimum inter-arrival time between messages transmission requests from the same message stream is known.

CAN proved to be a very attractive solution for distributed embedded systems designers and, given its commercial success, the scientific community has been eager to contribute with improvements to the scheduling theory developed for CAN [26]. Proposals have been made on how to perform Earliest-Deadline-First scheduling on CAN [11], and stochastic analysis of CAN-based distributed systems have also been developed [29]. Other works have dealt with fault-tolerance issues in CAN, such as ideas on how to analyze response times in the presence of retransmissions due to communication faults [23], and many other works have shown approaches to different aspects of fault-tolerance (e.g. [6, 4, 24]). The analysis of the CAN bus can be conveniently incorporated into the holistic analysis framework [25] which gives designers the capability of analyzing end-to-end delays in distributed real-time systems (for example, in a vehicle). More recently, practical aspects of the implementation of CAN were introduced into the analysis [10].

This paper focuses on utilization bounds for non-preemptive static-priority scheduling on a CAN bus. Utilization bounds are well-established in the real-time scheduling literature. For example, on a single processor, the utilization bound of RM is 69% [17] and a number of extensions to this result have been developed [20, 22, 16, 14, 7, 28]. On a multiprocessor a utilization bound of 33% [2] has been achieved with static-priority scheduling.

Utilization bounds of static-priority schedulers for timed token protocols in Fiber Distributed Data Interface (FDDI) networks were developed in [1, 18, 30, 31].

Utilization-based admission control for static-priority schedulers in a networked environment have also been addressed by [8]. The driving model of this paper is the Differentiated Services (diffserv) architecture and it is based on the idea of deriving utilization bounds for a concretely defined networked system in terms of the number of nodes (routers and switches in this particular work), topology and message flows.

In [15] authors study the several problems of hard real-time bus scheduling, including utilization bounds. Unfortunately, the bound is zero, if the number of messages is infinite. In this paper, we exploit the particular characteristics of the CAN bus to develop a utilization bound greater than zero.

Another work which underscores the role of the utilization bound in communication systems was developed for links scheduled by weighted round-robin scheduling [27].

Noticeably, prior to [3], no utilization bound for the CAN bus was known. It has been previously established in the literature that non-preemptive static-priority scheduling (as implemented by CAN) has utilization bound of zero [13]. While this is true in general, the CAN standard stipulates bounds on the message transmission times; these are a consequence of the bounds on the number of bytes in the data payload allowed by the CAN standard. This fact enabled the development of a utilization bound for the CAN bus in [3]. The work in [3] did not perform experimentation on real-world platforms to test the theory in practice. As exemplified by previous research (e.g. [10]), it is relevant to analyse the theory in face of the actual characteristics of the implementations of CAN controllers.

### 3 Assumptions, Terminology and Problem statement

Consider a distributed computer system comprising computer nodes, each interconnected with a Controller Area Network (CAN) bus. The workload is described by a set of  $n$  message streams. This set is partitioned into subsets; one subset for each computer node. Each subset is assigned to a computer node and consequently, each message stream is assigned to exactly one computer node. It is permitted for many message streams to be assigned to a single computer node. We do not make any assumptions on the assignment of message streams to computer nodes.

A message stream  $\tau_i$  generates a (potentially infinite) sequence of messages. The time when these messages arrive cannot be controlled by the CAN controllers. It is assumed that the time between two consecutive arrivals of messages from the same message stream  $\tau_i$  is at least  $T_i$ . A message from message stream  $\tau_i$  has a message transmission time  $C_i$ ; this includes the frame header and the arbitration for the CAN bus.  $R_i$  denotes the response time of message stream  $\tau_i$ .  $R_i$  is defined as the minimum number such that for every message transmission from  $\tau_i$ , it holds that the time from the message transmission request until the message of that transmission request has finished transmission is at most  $R_i$  time units. If  $R_i \leq T_i$  then we say that the message stream  $\tau_i$  meets its deadlines; otherwise it misses its deadline. It is assumed that  $0 < C_i \leq T_i$ , and that  $T_i$  and  $C_i$  are multiples of QUANTUM where QUANTUM denotes the time duration of a single bit in the CAN bus. (Many previous works on schedulability analysis of CAN used the symbol  $\tau_{bit}$  to denote the duration of single bit. We do not use that notation because, as follows from the Liu-and-Layland notation, we use  $\tau_i$  to denote a message stream.)

We assume that a message stream is assigned a priority, an integer. This integer is unique, that is, if  $\tau_i$  has priority  $x$  then there is no other message stream  $\tau_j$  with priority  $x$ . We assume that message streams are assigned priorities according to rate-monotonic (RM) with the identifier used as a tie-breaker, that is:

$$(T_i < T_j) \vee ((T_i = T_j) \wedge (i < j)) \\ \Rightarrow \tau_i \text{ has higher priority than } \tau_j$$

When we say that  $\tau_i$  has higher priority than  $\tau_j$  then it means that  $\tau_i$  has lower priority number than  $\tau_j$ . We assume that a message has the same priority as the priority of the message stream it belongs to. For convenience we use the following notations:

$$\text{hep}(i) = \{j : \tau_j \text{ has higher priority than } \tau_i \\ \text{or } \tau_j \text{ has equal priority to } \tau_i\}$$

and

$$\text{hp}(i) = \{j : \tau_j \text{ has higher priority than } \tau_i\}$$

and

$$\text{lp}(i) = \{j : \tau_j \text{ has lower priority than } \tau_i\}$$

We assume that all outstanding message transmission requests on a computer node are sorted in a priority queue and the highest-priority message transmission request competes with the highest-priority transmission request on the other nodes. The message transmission request with the highest priority "wins" and it sends its message on the CAN bus. All these assumptions are typical to CAN and its intended use in industrial systems. The utilization of a set of message streams is defined as:

$$U = \sum_{i=1}^n \frac{C_i}{T_i}$$

We address the problem of finding the utilization bound of CAN using RM. The utilization bound of CAN using RM is the maximum number such that if a set of message streams has a utilization that does not exceed the utilization bound of CAN and if messages are scheduled by CAN and if priorities are assigned according to RM then all deadlines are met. In order to solve this problem, it is helpful to understand non-preemptive RM scheduling.

#### 4 Background on Non-Preemptive Rate-Monotonic Scheduling

The CAN bus uses non-preemptive static-priority scheduling and consequently the scheduling theory for non-preemptive static-priority scheduling can be used to compute the response time  $R_i$ .

It is known from previous research [9, 12] that  $R_i$  can be correctly computed by the following steps.  $B_i$  denotes the time that a message may need to wait for a lower-priority message to finish its transmissions.  $B_i$  is computed as:

$$B_i = \max_{k \in lp(i)} C_k \quad (1)$$

The method for computing  $R_i$  explores all messages released from message stream  $\tau_i$  during an interval whose length is  $t_i$ , computed as follows:

$$t_i^0 = C_i \quad (2)$$

and iterate according to:

$$t_i^{k+1} = B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{t_i^k}{T_j} \right\rceil \cdot C_j \quad (3)$$

When Equation 3 converges with  $t_i^{k+1} = t_i^k$  then this is the value of  $t_i$ .

We can now compute  $w_i(q)$ , the queuing time of the  $q^{th}$  message in the interval of duration  $t_i$ . It is computed iteratively until we obtain convergence,  $w_i^{k+1}(q) = w_i^k(q)$  for the following iterative procedure:

$$w_i^0(q) = B_i + q \cdot C_i \quad (4)$$



and iterate according to:

$$w_i^{k+1}(q) = B_i + q \cdot C_i + \sum_{\forall j \in \text{hep}(i)} \left\lceil \frac{w_i^k(q) + \text{QUANTUM}}{T_j} \right\rceil \cdot C_j \quad (5)$$

when Equation 5 converges with  $w_i^k(q) = w_i^{k+1}(q)$  then this is the value of  $w_i(q)$ . We compute the response-time for the  $q^{\text{th}}$  arrival in the priority level- $i$  busy period as:

$$R_i(q) = w_i(q) - q \cdot T_i + C_i \quad (6)$$

This is used to calculate the response time:

$$R_i = \max_{q=0..Q_i} R_i(q) \quad (7)$$

where  $Q_i$  is defined as:

$$Q_i = \left\lceil \frac{t_i}{T_i} \right\rceil$$

This analysis works for the case that

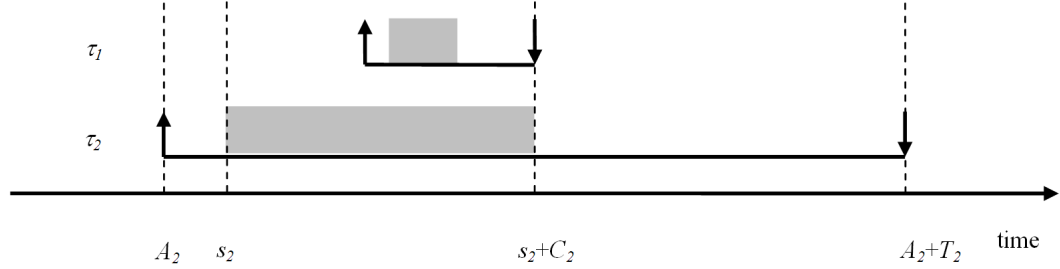
$$\sum_{i=1}^n \frac{C_i}{T_i} < 1 \quad (8)$$

From the definition of  $R_i$  it clearly holds that: If and only if  $\tau_i: R_i \leq T_i$  then all deadlines are met.

This exact schedulability analysis for non-preemptive scheduling is useful but unfortunately it does not offer a utilization bound. In fact, it is known [13] that all non-preemptive scheduling algorithms have a utilization bound zero. Since this result is important for our purposes (studying the utilization bound of CAN) we formalize it in Example 1.

*Example 1.* Consider two message streams to be scheduled with non-preemptive RM. Let message stream  $\tau_1$  be characterized as  $T_1 = \varepsilon$  and  $C_1 = 2\varepsilon^2$ . Let message stream  $\tau_2$  be characterized as  $T_2 = 1$  and  $C_2 = 2\varepsilon$ . We choose  $\varepsilon < 1/4$ . We let  $O_i$  denote the time when  $\tau_i$  arrives for the first time. The scheduling algorithm is not permitted to choose  $O_i$ . In order to make the discussion as general as possible, we consider both the case where the scheduling algorithm is not allowed to insert idle time and the case where the scheduling algorithm is allowed to insert idle time.

Let us consider an arbitrary transmission request by  $\tau_2$ ; Figure 1 illustrates this, and later we will test this scenario in practice, as shown in Figure 7.1. Let  $A_2$  denote the arrival time of that transmission request and let  $s_2$  denote the start time of the transmission of the message of that transmission request. Because the transmission time of  $\tau_2$  is twice as long as  $T_1$  we know that every time  $\tau_2$  transmits a message, it is possible (according to the sporadic model) that  $\tau_1$  will perform a message transmission request such that  $s_2 < A_1 < A_1 + T_1 < s_2 + C_2$ , and hence there is a time when both  $\tau_1$  and  $\tau_2$  must transmit simultaneously in order to meet deadlines. Hence a deadline is missed. Here,  $U = 4\varepsilon$ . Choosing  $\varepsilon \rightarrow 0$  yields that the utilization bound is zero. This example shows that for every non-preemptive scheduling algorithm, the utilization bound is zero.



**Fig. 1.** All non-preemptive scheduling algorithms have the utilization bound 0.

Based on Example 1, it would be tempting to believe that the utilization bound of CAN is zero. It can be seen however that in this example, the transmission time of one message stream is infinitely larger than the transmission time of another message stream. The CAN standard stipulates bounds on the message transmission times; these are a consequence of the bounds on the number of bytes in the data payload allowed by the CAN standard. As a result, the behavior as illustrated in Example 1 cannot happen. Example 2 shows a scenario which (as we will see later) is the scenario with the lowest utilization such CAN using RM misses a deadline.

*Example 2.* Consider two message streams ( $\tau_1$  and  $\tau_2$ ) to be scheduled with non-preemptive RM. Let us consider an arbitrary transmission request by  $\tau_2$  (observe that the time when a message arrives for the first time is not a decision of the scheduling algorithm). Immediately after starting transmission of the message by  $\tau_2$ , a new message is queued by  $\tau_1$ . It is clear that, in order for  $\tau_1$  to not miss deadlines,  $T_1 \geq C_2 + C_1$ . Considering that  $T_1 = C_2 + C_1$  and  $T_2 = L$ , where  $L \rightarrow \infty$ , the utilization of this scenario is  $U = C_1/(C_2 + C_1) + C_2/\infty$ . If we assign  $T_1 = 182$ ,  $C_1 = 47$  and  $T_2 = L$ ,  $C_2 = 136$ , we have  $U = 47/182 + 136/L$ , which is approximately 25%. This example will be later (Section ??) tested in real-world CAN platforms and is illustrated in Figure 7.1.

For this reason, we will (in Section 5) prove the utilization bound of non-preemptive RM scheduling with restrictions on message transmission times. And then (in Section 6) we will apply that utilization bound by applying the values required by CAN.

## 5 Utilization Bound for Non-Preemptive Rate-Monotonic Scheduling

Let us first prove two lemmas (Lemma 1 and Lemma 2) that are instrumental and then see a theorem (Theorem 1) which states that if the relationship between transmission times is known and the utilization does not exceed a certain bound then all deadlines are met.

**Lemma 1.** Consider a message stream  $\tau_i$ . If it holds that all message streams in  $\text{hp}(i)$  meet their deadlines and it holds that for  $\tau_i$  that

$$B_i + C_i + \sum_{\forall j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j} \cdot T_i) \leq T_i$$

then  $\tau_i$  meets all its deadlines.

*Proof.* The proof is by contradiction. Let us assume that the lemma was false. Then it must be that:

$$B_i + C_i + \sum_{\forall j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j} \cdot T_i) \leq T_i \quad (9)$$

and a deadline of  $\tau_i$  was missed.

Of all messages of  $\tau_i$  that missed a deadline let  $M$  denote the message with the earliest deadline. And let  $t_1$  denote the deadline of message  $M$ . Let  $t_0$  denote the arrival time of  $M$ . We know that the time interval  $[t_0, t_1)$  belongs to a priority level- $i$  busy period (if this was not the case then message  $M$  would have been transmitted and met its deadline). We know that:

$$\begin{aligned} \text{During } [t_0, t_1), \text{ a message of lower priority than } M \\ \text{can transmit for at most } B_i \text{ time units, where } B_i \\ \text{is given by Equation 1.} \end{aligned} \quad (10)$$

We also know that:

$$\begin{aligned} \text{During } [t_0, t_1), \text{ a message } M \text{ can transmit} \\ \text{for at most } C_i \text{ time units} \end{aligned} \quad (11)$$

Let us now study messages from a message stream  $\tau_j$ , which has higher priority than  $\tau_i$ . The time interval  $[t_0, t_1)$  can be subdivided into  $[t_0, t')$  and  $[t', t'')$  and  $[t'', t_1)$  such that  $t'$  is the earliest arrival time of  $\tau_j$  in  $[t_0, t_1)$  and  $t''$  is the latest arrival time of  $\tau_j$  in  $[t_0, t_1)$ . Based on this decomposition and using the knowledge that  $T_j \leq T_i$ , it is easy to see that:

$$\begin{aligned} \text{During } [t_0, t_1), \text{ a message from } \tau_j \text{ can transmit} \\ \text{for at most } \lfloor \frac{T_i}{T_j} \rfloor \cdot C_j + 2 \cdot C_j \text{ time units} \end{aligned} \quad (12)$$

It is easy to show (using our knowledge that  $T_j \leq T_i$ ) that:

$$\lfloor \frac{T_i}{T_j} \rfloor \cdot C_j + 2 \cdot C_j \leq 3 \cdot \frac{C_j}{T_j} \cdot T_i \quad (13)$$

Combining Inequality 12 with Inequality 13 and applying it on all message streams with higher priority than  $\tau_i$  yields:

$$\begin{aligned} &\text{During } [t_0, t_1), \text{ messages from } \text{hp}(i) \text{ can transmit} \\ &\text{for at most } \sum_{j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j} \cdot T_i) \text{ time units} \end{aligned} \quad (14)$$

Combining Inequality 10, Inequality 11 and Inequality 14 yields:

$$\begin{aligned} &\text{During } [t_0, t_1), \text{ the channel is busy for at most} \\ &B_i + C_i + \sum_{j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j} \cdot T_i) \text{ time units} \end{aligned} \quad (15)$$

In order to miss a deadline, it follows from (i) the fact that  $[t_0, t_1)$  is a priority level- $i$  busy period and (ii) the use of Inequality 15 that:

$$B_i + C_i + \sum_{j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j} \cdot T_i) > T_i \quad (16)$$

But this contradicts Inequality 9 and hence the lemma is true.

**Lemma 2.** *If it holds for all  $\tau_i$  that*

$$B_i + C_i + \sum_{j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j} \cdot T_i) \leq T_i$$

*then all deadlines of all message streams are met.*

*Proof.* Follows by induction on  $i$  and using Lemma 1.

**Theorem 1.** *Let  $x$  denote a real number such that  $2 \leq x$ . We claim that: If it holds that*

$$\forall i, j : \frac{C_i}{C_j} \leq x$$

*and if all message streams are scheduled with non-preemptive RM and if*

$$\sum_{i=1}^n \frac{C_i}{T_i} \leq \frac{1}{x+1}$$

*then all deadlines are met.*

*Proof.* The proof is by contradiction. Let us assume that the theorem was false. Then it must be that there is an assignment to  $x$  such that  $2 \leq x$  and there is a set of message streams such that:

$$\forall i, j : \frac{C_i}{C_j} \leq x \quad (17)$$

and message streams are scheduled with non-preemptive RM and

$$\sum_{i=1}^n \frac{C_i}{T_i} \leq \frac{1}{x+1} \quad (18)$$

and a deadline was missed.

Since a deadline was missed it follows (from Lemma 2) that there is an  $i$  such that

$$B_i + C_i + \sum_{\forall j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j} \cdot T_i) > T_i \quad (19)$$

Rewriting (19) yields:

$$1 < \frac{B_i}{T_i} + \frac{C_i}{T_i} + \sum_{\forall j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j}) \quad (20)$$

Rewriting (18) yields:

$$(\sum_{\forall j \in \text{hp}(i)} \frac{C_j}{T_j}) + (\sum_{\forall j \in \{i\} \cup \text{lp}(i)} \frac{C_j}{T_j}) \leq \frac{1}{x+1} \quad (21)$$

Multiplying (21) by  $x+1$  yields:

$$(\sum_{\forall j \in \text{hp}(i)} (x+1) \cdot \frac{C_j}{T_j}) + (\sum_{\forall j \in \{i\} \cup \text{lp}(i)} (x+1) \cdot \frac{C_j}{T_j}) \leq 1 \quad (22)$$

Since  $3 \leq x+1$  we have:

$$(\sum_{\forall j \in \text{hp}(i)} 3 \cdot \frac{C_j}{T_j}) + (\sum_{\forall j \in \{i\} \cup \text{lp}(i)} (x+1) \cdot \frac{C_j}{T_j}) \leq 1 \quad (23)$$

Combining (23) with (20) yields:

$$\begin{aligned} & (\sum_{\forall j \in \text{hp}(i)} 3 \cdot \frac{C_j}{T_j}) + (\sum_{\forall j \in \{i\} \cup \text{lp}(i)} (x+1) \cdot \frac{C_j}{T_j}) \\ & < \frac{B_i}{T_i} + \frac{C_i}{T_i} + \sum_{\forall j \in \text{hp}(i)} (3 \cdot \frac{C_j}{T_j}) \end{aligned} \quad (24)$$

Simplifying (24) yields:

$$(\sum_{\forall j \in \{i\} \cup \text{lp}(i)} (x+1) \cdot \frac{C_j}{T_j}) < \frac{B_i}{T_i} + \frac{C_i}{T_i} \quad (25)$$

We can rewrite (25) into:

$$(x+1) \cdot \frac{C_i}{T_i} + (\sum_{\forall j \in \text{lp}(i)} (x+1) \cdot \frac{C_j}{T_j}) < \frac{B_i}{T_i} + \frac{C_i}{T_i} \quad (26)$$

Case 1.  $\text{lp}(i)$  is empty

Since  $\text{lp}(i)$  is empty, it clearly holds that  $B_i = 0$ . Using this on (26) yields:

$$(x+1) \cdot \frac{C_i}{T_i} < \frac{C_i}{T_i} \quad (27)$$

Multiplying (27) by  $T_i$  and using the knowledge that  $2 \leq x$  yields:

$$3 \cdot \frac{C_i}{T_i} < \frac{C_i}{T_i} \quad (28)$$

It is easy to see that (28) is impossible. (End of Case 1.)

Case 2.  $\text{lp}(i)$  is non-empty

We know from (1) that  $B_i$  is obtained as:

$$B_i = \max_{k \in \text{lp}(i)} C_k$$

Let  $k$  denote the index of the message stream with the maximum  $C_k$  among all indices in  $\text{lp}(i)$ . Since a sum cannot be less than one of its terms it clearly holds that:

$$(x+1) \cdot \frac{C_k}{T_k} \leq \left( \sum_{\forall j \in \text{lp}(i)} (x+1) \cdot \frac{C_j}{T_j} \right) \quad (29)$$

Applying (29) on (26) yields:

$$(x+1) \cdot \frac{C_i}{T_i} + (x+1) \cdot \frac{C_k}{T_k} < \frac{B_i}{T_i} + \frac{C_i}{T_i} \quad (30)$$

And using the knowledge that  $B_i = C_k$  yields:

$$(x+1) \cdot \frac{C_i}{T_i} + (x+1) \cdot \frac{C_k}{T_k} < \frac{C_k}{T_i} + \frac{C_i}{T_i} \quad (31)$$

Multiplying with  $T_i$  yields:

$$(x+1) \cdot C_i + (x+1) \cdot \frac{C_k}{T_k} \cdot T_i < C_k + C_i \quad (32)$$

Clearly in general it holds that:

$$0 \leq (x+1) \cdot \frac{C_k}{T_k} \cdot T_i \quad (33)$$

Combining (33) and (32) yields and simplifying yields:

$$x < \frac{C_k}{C_i} \quad (34)$$

But this contradicts (17). (End of Case 2.)

We can see that regardless of which case occurs we have a contradiction. Hence the theorem is correct.

## 6 The Utilization Bound of CAN

The CAN standard specifies that the data payload is at most 8 bytes and at least 0 bytes. Let  $S_i$  denote the number of bytes of payload of a packet and recall that QUANTUM denotes the time duration of a single bit. CAN uses a technique called *bit-stuffing* that ensures that not too many consecutive zeros or consecutive ones are on the bus. Based on this bit stuffing and using the notation, a well-known expression [9, 12] exists for computing the maximum transmission time and for convenience we state it below:

$$C_i = (g + 8 \cdot S_i + 13 + \lfloor \frac{g + 8 \cdot S_i - 1}{4} \rfloor) \cdot \text{QUANTUM} \quad (35)$$

where  $g$  is the number of bits in the frame header that is subject to bit-stuffing. For CAN2.0A (which have 11-bit priorities) we have  $g=34$  and for CAN2.0B (which have 29-bit priorities) we have  $g = 54$  [9, 12]. Let CMAX denote the maximum transmission time and CMIN denote the minimum transmission time. By applying  $S_i = 8$  on (37):

$$CMAX = (g + 64 + 13 + \lfloor \frac{g + 64 - 1}{4} \rfloor) \cdot \text{QUANTUM} \quad (36)$$

By applying  $S_i = 0$  on (37) and observing that the expression in the floor in (37) represents the extra time due to bit-stuffing, which may be zero, we obtain:

$$CMIN = (g + 13) \cdot \text{QUANTUM} \quad (37)$$

We can now apply these to both CAN2.0A and CAN2.0B.

### 6.1 CAN2.0A

Applying  $g = 34$  on (38) yields:

$$\begin{aligned} CMAX &= (34 + 64 + 13 + \lfloor \frac{34 + 64 - 1}{4} \rfloor) \cdot \text{QUANTUM} \\ &= 135 \cdot \text{QUANTUM} \end{aligned} \quad (38)$$

and applying  $g = 34$  on (39) yields:

$$\begin{aligned} CMIN &= (g + 13) \cdot \text{QUANTUM} \\ &= 47 \cdot \text{QUANTUM} \end{aligned} \quad (39)$$

Applying these values gives us Theorem 2.

**Theorem 2.** *If messages are scheduled with CAN2.0A and if messages comply with the length restrictions stipulated by CAN2.0A and if message streams are scheduled with non-preemptive RM and if*

$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq \frac{47}{182}$$

*then all deadlines are met.*

*Proof.* Follows from applying  $x = 135/47$  in Theorem 1.

It is worth to observe that  $47/182$  is approximately 0.2582 and hence the utilization bound of CAN2.0A is 25%.

## 6.2 CAN2.0B

Applying  $g = 54$  on (38) yields:

$$\begin{aligned} CMAX &= (56 + 64 + 13 + \lfloor \frac{56 + 64 - 1}{4} \rfloor) \cdot \text{QUANTUM} \\ &= 160 \cdot \text{QUANTUM} \end{aligned} \quad (40)$$

and applying  $g = 54$  on (39) yields:

$$\begin{aligned} CMIN &= (g + 13) \cdot \text{QUANTUM} \\ &= 67 \cdot \text{QUANTUM} \end{aligned} \quad (41)$$

Applying these values gives us Theorem 3.

**Theorem 3.** *If messages are scheduled with CAN2.0B and if messages comply with the length restrictions stipulated by CAN2.0B and if message streams are scheduled with non-preemptive RM and if*

$$U = \sum_{i=1}^n \frac{C_i}{T_i} \leq \frac{67}{227}$$

*then all deadlines are met.*

*Proof.* Follows from applying  $x=160/67$  in Theorem 1.

It is worth to observe that  $67/227$  is approximately 0.2951 and hence the utilization bound of CAN2.0B is 29%.

## 7 Evaluation

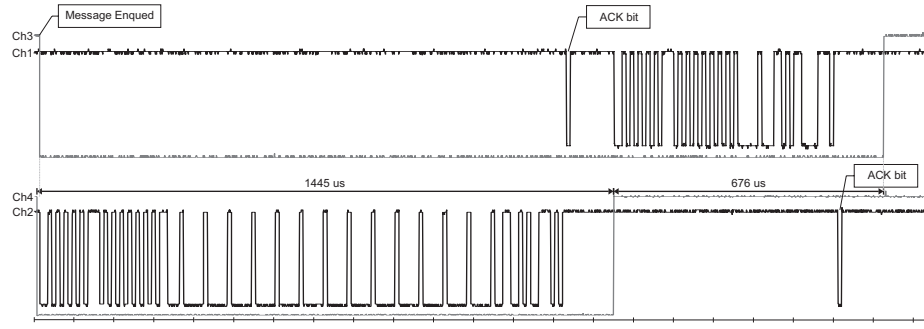
We have extensively tested the utilization bound for CAN. Our experiments were executed using embedded computer platforms with an integrated CAN controller. The embedded computer platform have an Atmel AVR AT90CAN128 microcontroller unit (MCU) running at 8 MHz. This microcontroller provides a CAN controller compliant with V2.0A and V2.0B CAN standards. We used CAN2.0B and the bus was running at a data rate of 100 Kbit/s in all experiments.

### 7.1 Testing the CAN Utilization Bound in Practice

We tested the scenario with the lowest utilization such that CAN using RM misses a deadline (as proven by the CAN utilization bound presented previously) using CAN-enabled embedded computer platforms. This scenario was previously described in Example 2.

Testing the scenario in Example 2 will allow us to test the utilization bound in reality, and see how close we can get to the idealized situation. To do so, we have generated





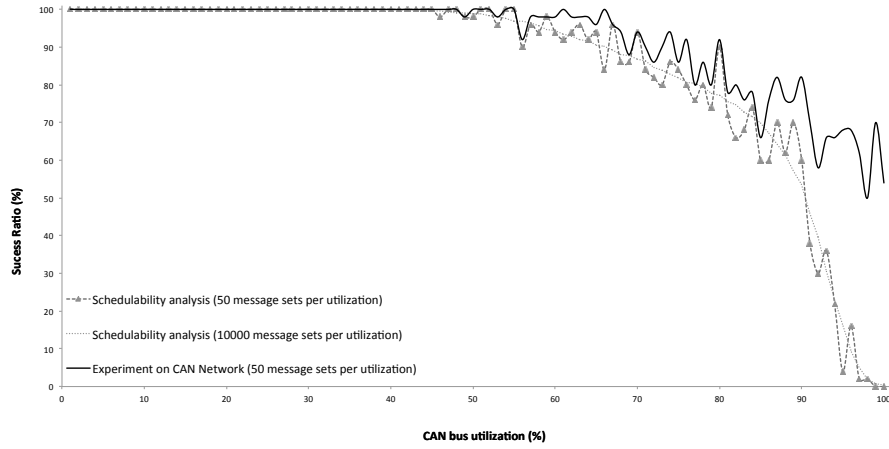
**Fig. 2.** Transmission of messages  $\tau_1$  and  $\tau_2$  in the CAN bus. CH1 and CH2 represent the bits transmitted for messages  $\tau_1$  and  $\tau_2$  respectively. CH3 and CH4 were obtained using a GPIO in the CAN-enabled MCU and change the state to low when the message is queued and change the state to high when the message is signaled to the MCU as transmitted

a message  $\tau_2$  in the CAN bus with a fixed priority and the maximum payload allowed by the protocol (8 bytes) and another message  $\tau_1$  with no payload that is queued for transmission when the message with the maximum payload starts transmission. A different node transmitted each message stream and we achieved synchronization between the nodes using a wire connected to an I/O port of the microcontrollers.

The transmission times of the messages were measured using an oscilloscope connected directly to the CAN bus. Examining the oscilloscope trace, we determined that the message with no payload had a transmission time (including the interframe space)  $C_1 = 676\mu s$  (the transmission time equation in [9] predicts that this message is  $670\mu s$  long; the difference is due to the resolution of our measurements using the oscilloscope; this is less than 1 bit time) and  $C_2 = 1445\mu s$  (approximately the time to transmit 144 bits ( $1440\mu s$ ); this is 14 bits less than the maximum predicted by the transmission time equation in [9] due to a bit stuffing lower the predicted by the theory. We note that the theory in [9] provides the maximum transmission time of a message when a theoretical maximum number of stuffing bits are inserted.

Using these measurements, the period for message  $\tau_1$  is  $T_1 = 676 + 1445 = 2121\mu s$ . As explained in Example 2, this is the minimum period we can set for message  $\tau_1$  such that this message stream does not miss deadlines. Figure 7.1 shows an oscilloscope trace of the message transmission on the CAN bus. In Figure 7.1, we can see that message  $\tau_1$  is queued after message  $\tau_2$ , to ensure that the latter is transmitted first in the CAN bus. With the trace in Figure 7.1, we can clearly observe the same scenario described in Example 2 occurring in a real CAN bus, and also that setting a shorter period for  $\tau_1$  would result in a deadline miss.

The utilization of the above experiment is  $U = 676/2121 \approx 32\%$ , which is higher than the calculated bound for CAN2.0B (the calculated bound is 29%) due to the fact that the packets sent in this experiment did not reproduce the bit stuffing which leads to the maximum transmission time of a message as predicted by the theory in [9].



**Fig. 3.** Average-case performance of exact schedulability analysis and transmission of message streams on CAN using RM.

Observe that this scenario (as described by Example 2) also illustrates the fact that the CAN utilization bound is tight, in the sense that no tighter bound can be derived.

## 7.2 Average Case Behavior

We have seen that the utilization bound for CAN is 25% and 29% for CAN2.0A and CAN2.0B, respectively. This result does however not rule out the possibility that deadlines can be met at a higher utilization. It is therefore interesting to know if the CAN bus can be used at a higher bus load. Previous work [9] has hinted that a much higher utilization can be used. In order to find out the answer, we performed a study that extensively compares the existing theory with real-world experiments. We generated message sets randomly (details are given below) and computed the number of these message sets that were feasible based on the exact schedulability analysis in [9]. Then, we programmed several CAN-enabled nodes with randomly selected message sets and ran long experiments to observe how the success rate of the message sets is in practice, compared to the theory.

The parameters for the randomly generated message streams are as follows.  $S_i$  is a uniformly distributed random variable from  $\{1, \dots, 8\}$ , given in bytes.  $C_i$  is computed from  $S_i$ . The parameter  $T_i$  is a uniformly distributed random variable from  $\{CMIN \times 2, \dots, 500000\}$  ( $T_i$  is given in  $\mu s$ ), where  $CMIN$  denotes the minimum message transmission time. The number of message streams in a set is between 2 and 9 (we want to assign one message stream per node and nine is the number of CAN nodes available for the experiment).

We generated sets of message streams randomly and for each set, we calculate the utilization and put it in one of 100 buckets. For example, if a set of message streams has a utilization equal to 0.6734 then it belongs to the bucket of sets with a utilization in

the range  $[0.67, 0.68]$ . We defined a size for each bucket and generated message streams until all the buckets were full. For each set in the bucket we also decided (based on the exact schedulability analysis in [9]) whether all messages meet deadlines. A set of message streams that meet all deadlines is said to be successful. The success ratio of a bucket is said to be the number of message streams that are successful in a bucket divided by the number of message streams in the bucket.

We first generated message streams for bucket sizes 50 and 10000. That is, we generated 50 message sets per utilization value (a total of  $50 \times 99$  message sets) and 10000 (a total of  $50 \times 10000$  message sets) and plotted the success ratio of the message streams in Figure 2. It can be seen that the a bucket size of 50 message sets per utilization oscillates quite noticeably, but the bucket size of 10000 exhibits a very steady success ratio curve. Nevertheless, the curve for a bucket size of 50 does follow the line for a bucket size of 10000.

Figure 2 also shows the results for the experiment using real CAN-enabled nodes. For this experiment, we used the same message sets generated previously. Note that it was not practical to experiment with a bucket size of 10000, as that would result in very long experiments (however we did run test experiments with a large bucket size to confirm our results). We ran all the message sets for the bucket size 50, and this resulted in an experiment that ran for 28 days. The experiment was executed by programming nine (this was the maximum number of message streams generated in each message set) platforms with an AT90CAN128 MCU, using CAN 2.0B. Each node was programmed with at most one message stream (not all message sets had exactly nine message streams, thus nodes might be idle in some runs). A master node sent an indication of the start of each run and the first message of each message stream was queued synchronously at each node. Nodes used the onboard real-time clock to measure the time since a message was queued until it was transmitted and signaled the master node, in case a deadline miss was detected. Message sets where a deadline miss was encountered were marked unsuccessful. Clearly, this experiment did not guarantee to find all deadline misses as the schedulability analysis does, nevertheless, the success ratio of this experiment follows the trend of the schedulability analysis.

Observe (Figure 2) that up to a 45% utilization, all message streams meet deadlines. This is expected considering that the utilization bound is 29%. It can also be seen that the experiments show a similar behavior to the curve with the schedulability analysis for a bucket size of 50. We find this experiment an interesting stress test of both theory and practice.

## 8 Conclusions

We have revisited previous results on the CAN utilization bound [3], which state that the utilization bound for CAN is 25% and 29% for CAN2.0A and CAN2.0B respectively. This work reviewed related work on schedulability analysis of CAN, other utilization bounds and provided additional evidence of the relevance of the result. We have also performed comprehensive tests of the utilization bound of CAN using real-world CAN platforms and provide a measure of the pessimism of the bound.

These results finally provide theoretical foundation to empirical evidence found by designers of distributed CAN-based systems. Furthermore, a utilization bound provides a scalable and robust tool to determine the schedulability of a system. Knowing that the utilization bound is different from zero, a system designer can also rest assured that all deadlines can be met by using sufficiently high speed data rate (assuming that such components are available; in CAN it is possible for a designer to tradeoff the length of the bus for data rate).

## Acknowledgment

This work was supported by the CISTER Research Unit (608FCT), co-funded by national funds (PIDDAC) through the FCT-MCTES (Portuguese Foundation for Science and Technology) and by FEDER (European Regional Development Fund) through COMPETE (POFC - Operational Programme "Thematic Factors of Competitiveness").

## References

1. G. Agrawal, B. Chen, W. Zhao, and S. Davari. Guaranteeing synchronous message deadlines with the timed token medium access control protocol. *IEEE Trans. Comput.*, 43:327–339, March 1994.
2. B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *22nd IEEE Real-Time Systems Symposium (RTSS'01)*, pages 193–202, 2001.
3. Björn Andersson and Eduardo Tovar. The utilization bound of non-preemptive rate-monotonic scheduling in controller area networks is 25%. In *IEEE Fourth International Symposium on Industrial Embedded Systems - SIES, Lausanne, Switzerland*, pages 11–18, July 8 - 10, 2009.
4. M. Barranco, J. Proenza, G. Rodriguez-Navas, and L. Almeida. An active star topology for improving fault confinement in can networks. *Industrial Informatics, IEEE Transactions on*, 2(2):78 – 85, may 2006.
5. Bosch GmbH, Stuttgart, Germany. *CAN Specification, ver. 2.0*, 1991.
6. G. Buja, J. R. Pimentel, and A. Zuccollo. Overcoming babbling-idiot failures in CAN networks: A simple and effective bus guardian solution for the FlexCAN architecture. *IEEE Transactions on Industrial Informatics*, 3:225–233, 2007.
7. Deji Chen, A.K. Mok, and Tei-Wei Kuo. Utilization bound revisited. *Computers, IEEE Transactions on*, 52(3):351 – 361, march 2003.
8. Byung-Kyu Choi, Dong Xuan, Riccardo Bettati, Wei Zhao, and Chengzhi Li. Utilization-based admission control for scalable real-time communication. *Real-Time Systems*, 24:171–202, 2003. 10.1023/A:1021778402786.
9. R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Journal of Real-Time Systems*, 35(3):239–272, 2007.
10. Robert I. Davis, Steffen Kollmann, Victor Pollex, and Frank Slomka. Controller area network (can) schedulability analysis with fifo queues. In *Proceedings of the 2011 23rd Euromicro Conference on Real-Time Systems, ECRTS '11*, pages 45–56, Washington, DC, USA, 2011. IEEE Computer Society.
11. M. DiNatale. Scheduling the CAN bus with earliest deadline techniques. In *21st IEEE Real-Time Systems Symposium (RTSS'00)*, pages 259–268, 2000.

12. L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical report, INRIA, Technical Report RR-2966, 1996.
13. K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *12nd IEEE Real-Time Systems Symposium (RTSS'91)*, pages 129–139, 1991.
14. T.-W. Kuo and A.K. Mok. Load adjustment in adaptive real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 160–170, dec 1991.
15. John P. Lehoczky and Lui Sha. Performance of real-time bus scheduling algorithms. *SIGMETRICS Perform. Eval. Rev.*, 14:44–53, May 1986.
16. J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209, dec 1990.
17. C.L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20:46–61, 1973.
18. Nicholas Malcolm and Wei Zhao. Guaranteeing synchronous messages with arbitrary deadline constraints in an fddi network. In *In Proceedings of the IEEE Conference on Local Computer Networks*, pages 186–195, 1993.
19. A. K. Mok and S. Ward. Distributed broadcast channel access. *Computer Networks*, 3:327–335, 1979.
20. Aloysius K. Mok and Deji Chen. A multiframe model for real-time tasks. *IEEE Trans. Softw. Eng.*, 23:635–645, October 1997.
21. L. Niesnevich and E. Strasbourger. Decentralized priority in data communication. In *Second International Symposium on Computer Architecture*, pages 1–6, 1975.
22. Dar-Tzen Peng and Kang G. Shin. A new performance measure for scheduling independent real-time tasks. *J. Parallel Distrib. Comput.*, 19:11–26, September 1993.
23. S. Punnekkat, H. Hansson, and C. Norström. Response time analysis under errors for CAN. In *6th IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, pages 258–265, 2000.
24. M. Short and M.J. Pont. Fault-tolerant time-triggered communication using can. *Industrial Informatics, IEEE Transactions on*, 3(2):131–142, may 2007.
25. K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40:117–134, 1994.
26. K. Tindell, H. Hansson, and A. Wellings. Analysing real-time communications: Controller area network (CAN). In *15th Real-Time Systems Symposium (RTSS'94)*, pages 259–263, 1994.
27. J. Wu, J.-C. Liu, and W. Zhao. Utilization-bound based schedulability analysis of weighted round robin schedulers. In *28th IEEE Real-Time Systems Symposium (RTSS'07)*, pages 435–446, 2007.
28. Jianjia Wu, Jyh-Charn Liu, and Wei Zhao. On schedulability bounds of static priority schedulers. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 529–540, Washington, DC, USA, 2005. IEEE Computer Society.
29. Haibo Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli. Stochastic analysis of can-based real-time automotive systems. *Industrial Informatics, IEEE Transactions on*, 5(4):388–401, nov. 2009.
30. Sijing Zhang and Alan Burns. An optimal synchronous bandwidth allocation scheme for guaranteeing synchronous message deadlines with the timed-token mac protocol. *IEEE/ACM Trans. Netw.*, 3:729–741, December 1995.
31. Sijing Zhang and E. Stewart Lee. Efficient global allocation of synchronous bandwidths for hard real-time communication with the timed token mac protocol. In *in Proc. 6th Int. Conf. Real-Time Computing Systems and Applications, Hong Kong*, pages 472–479, 1999.

# When Theory Meets Technology

Tullio Vardanega

Department of Mathematics, University of Padua,  
via Trieste, 63 - 35121 Padua, Italy  
tullio.vardanega@math.unipd.it

**Abstract.** The distance between state-of-the-art research and industrial practice is often frustratingly large. Bridging that gap is not obvious, for many reasons, cultural and practical alike. One of the cures is to inject solid research results into technology ready for industrial use. When this happens, the making of that technology educates the technology developers and the user community with them, considerably more than simply publishing papers. To make that happen very rare skills are required, among which persistence and authority across border. Some of this has happened with the Ada technology: this paper tells the story of how two particular sets of language features came directly from state-of-the-art research results under lead and push by Alan Burns.

## 1 Introduction

This paper celebrates some of the many research contributions made by Alan Burns in his long career. The angle taken in this paper is slanted to the world of the Ada programming language. The reason for this choice is that taking state-of-the-art research results into technology ready for industrial use is a much wanted contribution, which takes very rare skills and authority across borders. The author of this paper had the good fortune of witnessing the effects of Alan Burns' influence on the evolution of Ada's support for real-time programming from the early '90s until now. I have learned a lot in that process and I enjoyed choosing some of the contributions for this paper, among those that I felt most impressed with.

In keeping with the intent of this collection, most of the materials presented in this paper are not original: the contents of section 2 refresh material already published in [26, 24, 27]; the contents of section 3 refresh material already published in [9, 28]. The selection and the refreshing were by the author, the rest from various sources, including of course Alan Burns.

## 2 Contribution #1

### 2.1 Context

In the mid 90's the high-integrity embedded systems industry of the time, which the author witnessed from the angle of the European Space Agency, was confronted by the surge of the "cheaper-faster-better" mantra. The aspiration behind it was to increase functional capability, decrease all manners of development and operation costs, achieve

shorter time to market: by and large, a contamination from the consumer electronics market where those demands are innate.

Where high integrity is required, verification and validation needs dominate development costs and choices. In that context, innovation in methods and technology is welcome so long as the benefits it can bring are shown to be much superior to the disruption that its adoption may cause. Progress in that domain therefore occurs more by evolution than by revolution. Discontinuities may occur of course, for a number of reasons, which are most frequently exogenous.

The cheaper-faster-better challenge had much to do with the software, more like an enabling factor than a vector of cost. It is instructive to consider that the software embedded on board a satellite or an avionics system still accounts for less than 10% of the overall development cost, but with a many times larger contribution to the ultimate value added system of system.

## 2.2 Angle of attack

The development process almost invariably in use at high-integrity systems industry is normally referred to as the "V model", in recognition of its V-shaped graphical representation. The descending branch of it embraces the definition and implementation phase. The ascending branch departs from the tail of the other branch to include integration and verification.

Common wisdom at the time had it that the effort, time and costs associated with the descending branch were well mastered so long as the user requirements were stable enough across the entire development process. In fact, requirements are never fully stable when the volume of unit production is small and numerous variants must be accommodated between one system and its successor. As development schedules are compressed, the descending branch of V model becomes fragile against incomplete or unstable user specifications. This hazard can be countered with a software design method that can absorb variations in requirements while retaining architectural integrity.

The ascending branch, which encompasses component integration and system testing, absorbs no less than 50% of the total software development time. Its tail position in the process and the high effort intensity that descends from its requiring integration on target (single processors then and now) expose it to serious risks of delay and cost overrun. All on-target activities possess a centralised and sequential connotation that obstructs the reduction of their duration. Schedule hazards however can be reduced by avoiding regression. In the mid 90's, there was ample room for improving the way in which the real-time requirements were specified, the software was designed to meet them, and analysis was used to verify them.

This is where one of the two contributions celebrated in this paper occurred.

The angle of attack to solve the problem at hand was to spread the load of concerns charged to the ascending branch over a broader portion of the development process. The idea was to replace much of the real-time testing occurring late by a functionally equivalent amount of static real-time analysis occurring earlier. Whereas the state-of-the-art in real-time systems theory at the time was successfully past that problem, industrial practice lagged much behind, impeded by the distance between what theory postulated and what technology was able to produce. One of the key issues was to make sure that

the extent of real-time verification performed at design level was not undermined by incoherent implementation or inadequate technology.

### 2.3 Outline of the solution

The essence of the solution was for the design method and the implementation technology to become able to jointly accommodate the refinement iterations and development increments that would ensue from replacing exploratory real-time testing with corroborative verification. The following two ingredients were central to this goal:

- A unified *design framework* in which the user may coherently consolidate and commit the system concept, verify the feasibility of the physical model in the time domain, and build on the relevant evidence in the construction of the system integration and verification strategy.
- A well-defined *computational model* which specifies: (a) the type of components to be used in the construction of the real-time architecture of the system, the means available for the communication and synchronisation between them, and the concurrency paradigm assumed for their run-time execution; (b) the execution properties and real-time attributes that characterise those components as well as the constraints placed on their use and interaction; and (c) an underpinning analytical model that allows the static analysis of the real-time behaviour of systems constructed in terms of those components. This is none other than the *workload model*, so central to real-time systems theory, only elevated to software design level.

The process segment covered by the design framework is comprised of four steps:

- B.1** *real-time design*: this stage is initially entered with the definition of the problem as determined by the system specification; it encompasses the definition of the logical model and the physical model of the system; it focuses on the characterisation of the real-time commitments required for the individual components of the system;
- B.2** *binding to computational model*: this stage is repeatedly entered when the abstract design of the system is transformed to implementation; in the envisioned process, this step occurs by a specialized tool automatically generating the application program from the design model: this notion anticipated what years later became known as the model transformation part of the model-driven architecture initiative [20]: the central trait of that vision was the sharp separation between the user responsibility of specifying and coding *jobs* and the transformation technology's of producing the *tasks* that would issue those jobs, in strict and correct-by-construction conformance with the workload model and scheduling algorithm selected for the system;
- B.3** *static analysis*: this stage is entered to statically determine whether the current implementation is capable of meeting the real-time commitments of the corresponding specification when executed on the designated target platform;
- B.4** *feedback to design*: this is entered to review the results obtained from static analysis and determine the corrective measures (if any) required to ensure that the system implementation meets the designated real-time requirements; the determinations established at this level are fed back to the design level in order for the designer to keep current the actual real-time attributes of the system and either conclude the

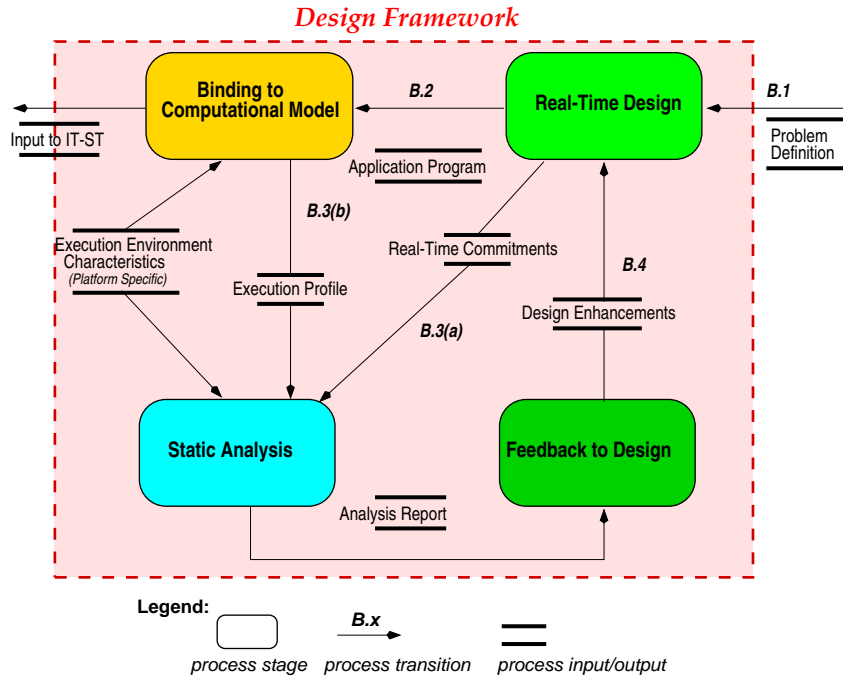


design process or perform further steps, incremental or corrective, of design and implementation.

The above development process assumes a design method that supports the construction of a real-time system in terms of the abstractions, execution characteristics, usage rules and constraints defined by the computational model of choice (**B.1**). For systems designed in this manner, the implementation effort is comprised of two complementary activities (**B.2**): (a) the derivation of the concurrent structure of the system, which is obtained by transformation of the structural components of the design into the corresponding code structures (tasks and shared resources); and (b) the incremental production of the functional components of the system (jobs) and their incorporation in the relevant structural element.

At any stage of the development process, the system thus constructed is comprised of: (i) the real-time requirements that must be met (**B.3(a)**); (ii) the execution profile of the system corresponding to its current task architecture and the timing measurements or estimates for their jobs (**B.3(b)**); and (iii) the timing cost of the execution environment.

Figure 1 depicts the resulting flow of design activities.



**Fig. 1.** Design Activities in the Proposed Process.

Real-time systems theory had much to say at the time (and still has, of course) about workload models and scheduling algorithms. What was needed was for theory to meet

technology in a manner that allowed the changes required to industrial practice to be evolutionary, thereby standing chances of acceptance.

The grand plan that occurred for this to happen was: (i) the conception of the HRT-HOOD design method [7, 8], as an extension to the base HOOD method at the time in use at European space industry; and (ii) the continued use of Ada as the implementation technology, building on its native support for fixed-priority preemptive scheduling [5, 6].

Systems designed with HRT-HOOD are concurrent and can be provided with a direct mapping to the tasking profile identified at the 8th International Real-Time Ada Workshop, subsequently called the *Ravenscar profile* [3]. The Ravenscar Profile was designed to minimize the upper bound on blocking time, to prevent deadlocks, and, by tool support, to statically verify that there is sufficient processing power available to ensure that all critical tasks meet their deadlines.

HRT-HOOD enriches the HOOD notion of *active* object, originally an entity with internal concurrency and control over when its invoked operations are executed, differentiating it between: *cyclic* objects, which are active objects that model time-triggered activities; and *sporadic* objects, which are active objects that model event-triggered activities. HRT-HOOD active objects are threaded and each of them possesses a well-defined mapping to one particular type of Ada task designed to match the corresponding execution semantics.

HRT-HOOD also includes the *protected* object to denote resources that provide protected access to shared data, as well as the *passive* object as defined in HOOD, to denote an entity with no internal concurrency and no control over when its invoked operations are executed.

The threads of control associated with active objects that conform to the Ravenscar Profile are subject to the following rules of conduct, which ensure that an HRT-HOOD design is amenable to schedulability analysis by construction:

- threads may access shared data by means of mutually exclusive calls to dedicated *resource* servers, which are modelled as HRT-HOOD protected objects;
- resource servers may offer a variety of services, each denoted by one distinct execution request; the execution requests exposed by a resource server must ensure mutual exclusion to concurrent callers and must be internally non-blocking;
- threads may synchronise with one another by means of mutually exclusive calls to dedicated *synchronisation* servers;
- synchronisation servers enable threads to suspend on synchronisation calls and other threads to perform non-blocking triggering calls; the model requires that any thread that wishes to suspend on a synchronisation call be allowed to do so exclusively through the services of a dedicated synchronisation server. As a consequence, no two threads can suspend on one and the same synchronisation call so that no suspension queue be required for any synchronisation service.

The cyclic thread maps to an independent thread of control implemented as an entry-less Ada task with a single, time-triggered invocation event and a potentially unbounded number of job invocations. The Ravenscar Profile requires that the clock that issues periodic releases be implemented to support high-accuracy absolute time delays, free

of local drift. Cumulative drift can easily be avoided using well-known coding patterns cf. e.g.: [22].

The sporadic thread maps to an independent thread of control implemented as an entry-less Ada task with a single, event-triggered invocation event and a potentially unbounded number of job invocations. The release event is transferred to the sporadic thread from a call to a synchronisation server. Minimum separation between subsequent sporadic releases can easily be ensured using well-known coding patterns cf. e.g.: [22].

The protected object was Ada 95's [16] ingenious solution for the implementation of resource servers and synchronisation servers.

Each protected object provides either a resource access control function, including a repository for the private data to manage and implement the resource, or a synchronisation function, or a combination of both for use in data-oriented synchronisation.

The locking protocol that was selected for use with protected objects was Ada's adaptation of Baker's stack-based protocol [4]: it became known as the "Immediate Priority Ceiling Inheritance Protocol" (aka Priority Ceiling Emulation). With it, a task that obtains a resource immediately gets its active priority raised to the ceiling of that resource. This was easy to implement on single-processor systems, as it did not require any queues of blocked tasks on access to resources: tasks waiting for a resource will have lower priority than the task holding the resource and therefore simply stay ready.

Protected object entries implement monitor procedures guarded by condition variables: callers queue up on them until the condition holds, akin to Dijkstra's guarded commands [12]: tasks can use this facility to signal and/or wait on events. The Ravenscar Profile allows at most one entry per protected object, to prevent multiple barriers from becoming open simultaneously as the result of a protected action and to avoid non-deterministic selection of the entry to service first. Moreover, the Ravenscar Profile requires that at most one task can wait on the signal event delivered via a synchronisation object: this restriction prevents the formation of queues of tasks on a protected entry, avoiding non-deterministic wait time in the queue. These prohibitions collectively warrant deterministic task release from protected entry queues. The evaluation of entry barriers must also be free of side effects. To this end, the Ravenscar Profile requires the barrier value to either be static or be read directly, without computation, from one of the protected object components. Applications that require composite barrier expressions can simply declare an additional Boolean value within the protected data and assign to it the result of the composite expression whenever the evaluation result may change.

The Ravenscar Profile finally disallows the current caller of a closed protected entry to be dynamically transferred to another entry queue, by which Ada's most powerful requeue statement supports condition synchronisation.

## 2.4 Ramifications

Both elements of contribution #1 made it into industrial use. HRT-HOOD was used in industry, with the European Space Agency as a most committed backer. With the maturation of modelling language and methods, including the already cited model-driven architecture initiative [20], it later evolved into HRT-UML [19] and then further into ESA's software reference architecture [21], always keeping its initial imprint.

The Ravenscar Profile officially made into the Ada 2005 standard as an "enforced profile" [17], which allowed for some polishing up and completion. It was very much used in industry, with solid success and interesting lessons learned cf. e.g.: [2, 13, 25]. One of those, reported in [15], is especially interesting for this paper collection, as it shows how diligently the industrial team sought conformance to the profile and its intent.

The system architecture in question had a data-flow oriented nature, typical of the onboard software controlling satellite systems.

- R1.** The system includes a range of physical I/O channels that multiplex data and messages of several types to and from a variety of devices with differing real-time requirements and which require the use of a suspensive event, either an interrupt or a time delay, to synchronise the acquisition of the read-out
- R2.** the I/O channels are shared among concurrent tasks, which may therefore compete for the execution of multiple data transfers over the same channel
- R3.** both the application software and its peripheral units have limited and costly buffering resources and the system allows no data to be lost (lest the mission product decrease), which implies that the size of data buffers must be kept as small as possible and buffer overflow must be avoided.

The industrial team saw some conflict between requirements R1-3 and the restricted interpretation of the Ravenscar Profile that they took to allow a compiler plug-in of their own production to automatically extract a model description to submit to schedulability analysis:

- R4.** every task must have a *single* suspension point;
- R5.** every protected entry (which cannot be more than one per protected object) must have a *single* caller task.

Restriction **R4** enforces sharp separation at design level between periodic and sporadic tasks. Restriction **R5** constructively ensures that no more than one task can ever suspend on the single entry of any protected object.

These restrictions are not part of the Ravenscar Profile per se. Yet they can be viewed as direct corollaries of it in any supporting design methodology which aims to warrant verifiability *by construction*, without requiring the use of overly complex analysis techniques.

The lesson learned from adhering to restriction **R4** was that matching the data-flow architecture of the system hardware with the chosen restrictions requires to capture – early in the design process – all potential task suspension points. Each such suspension point in fact carries an activation event for the task concerned. The developers' concern was that in that manner the software design strongly depends on the hardware architecture, and specifically on whether the peripheral units controlled by the on-board software are memory mapped (which would not be suspensive) or else connected to I/O channels (which would). The consequence was that any changes in the way hardware units may be interacted with – which may indeed occur in problematic circumstances – can no longer be concealed in low-level procedures, but it directly impacts the concurrent architecture of the system.

A design pattern emerged from the interaction between restrictions R1-3 and R4: the task that encapsulated external I/O was split a pair of single-suspension-point tasks: a "starter" task, which initiates the read-out calling a non-suspending protected procedure; and a "continuer" task, which undertakes to complete the job, once released from the closed protected entry which sanctions the availability of the read-out and which is opened by the relevant interrupt procedure. The "start" task can freely be made sporadic or periodic as the application requires. The "continuer" task instead is forced to be sporadic by the nature of the I/O channel interface.

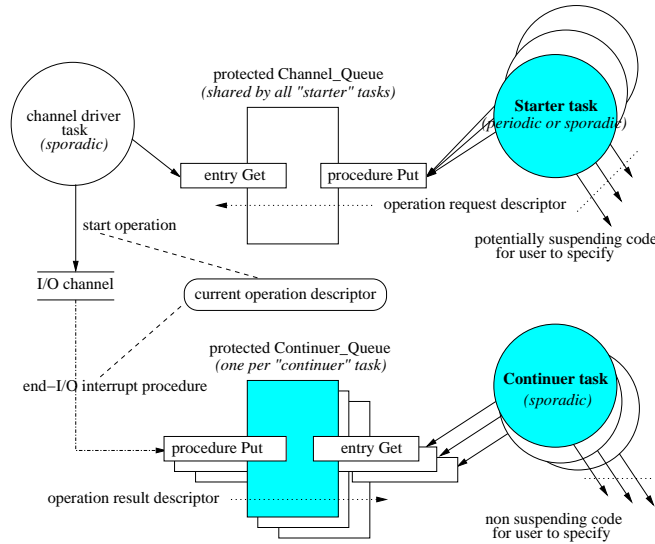
In fact, this archetypal design pattern needs further sophistication to fully meet requirements R1-3. It often occurred in the system in question that the same I/O channel used for the acquisition of read-outs is shared for several other hardware devices among multiple client tasks. This situation gave rise to two further restrictions:

- R6.** the channel can only perform I/O operations in strict sequential order, one at a time, from start to completion (which implies mutual exclusion on access to the channel resources to protect the execution of individual operations from possible interference);
- R7.** each process that shares the channel must post its request for a specific I/O operation to a multiplexer service and then wait for its required operation to complete

As the I/O operations must be serialised, the "starter" task must post its operation request into a protected queue and must do so in a way that incurs no suspension. This is easy to warrant, for the maximum number of incoming requests is as statically bounded as the number of clients, so that the size of the queue can be set accordingly. A dedicated sporadic task will fetch individual requests from the queue by calling on a protected entry with a barrier that stays open as long as requests are enqueued *and* the channel can accept new requests, and will subsequently initiate the corresponding operation on the channel.

The "continuer" tasks of the involved processes must not all queue on the one and the same protected entry whose barrier signals the completion of the current operation and which delivers the result of it. Each "continuer" task shall then wait on an *own* protected synchronisation object whose address will have to be tagged to the operation request issued by the corresponding "starter" task. Each such synchronisation object will have a protected entry for the "continuer" task to wait on, and a signalling protected procedure that sets the entry barrier to true when the corresponding operation has completed. The signalling procedure will be invoked by the interrupt protected procedure attached to the I/O channel hardware interrupt to deliver the operation result, to which end it shall have (shared) access to the current operation descriptor. Figure 2 depicts the pattern.

The developers' opinion was that this design pattern complicates static analysis in two ways. Firstly, the time offset that separates the release of the "starter" task from that of the "continuer" task in the same client process depends on the number and the nature of the enqueued I/O operations, which may be difficult to bound without incurring excessive pessimism. Secondly, in order for the pattern not to incur undesirable architectural coupling, the "end-IO interrupt procedure" shown in figure 2, should use an access-to-protected value (stored in the current operation descriptor) to call the protected signal procedure for the completed operation, which made the corresponding



**Fig. 2.** Block diagram of the design pattern.

call graph harder to construct for a worst-case execution time analysis technology of the time [23]. Both issues have been solved since: the former by applying to single-processor systems the principle of precedence-constrained analysis developed in [14] and the subsequent work by the same group; the latter by considerable improvement in the capability of timing analysis tool acting on the back-end of compilers cf. e.g.: [1].

### 3 Contribution #2

For many years after the publication of the seminal work in [18] the real-time theory community has much debated over which was better between EDF (Earliest Deadline First) and FP (Fixed Priority, initially known as rate monotonic). It was clear to all contenders that EDF had the advantage of optimality: it makes the best use of the available processor [11]. FP instead had the advantage of predictability and the important plus of efficient implementation on common real-time operating systems.

Ada 95 [16] incorporated support for FP, with a complete and consistent model that facilitated massive use in real industrial applications. By intent, Ada did not restrict itself to FP only: it indeed allowed other dispatching policies to be defined. The expectation was that vendors would develop other dispatching policy, after users' choice, and ratify them as secondary standards in ISO speak. By the early 2000 it became clear however that without the force of the language definition, the support for new dispatching policies in Ada would not see the light of day. In fact, whereas common wisdom has it that Ada is a world of its own with no influence on the rest of the software technol-

ogy landscape, the absence of support for EDF in Ada was also a factor for its absence elsewhere.

The move was therefore made to include support for EDF in Ada 2005: the ingenious trait of which was that EDF support was designed to allow it to be combined with FP. It is interesting for this collection to recall what the solution was.

Supporting EDF in the language required two new features:

1. representation of deadline for a task;
2. representation of preemption level for a protected object.

The reason for feature 1 is obvious. Feature 2 instead is the EDF equivalent of priority ceilings and allows protected objects to be shared by multiple tasks. A deadline can be defined as absolute or relative. The former needs the notion of current time. The EDF scheme can be more correctly expressed as *earliest absolute deadline first*. Ironically, the "cultural ban" on EDF in Ada caused Ada to miss direct representation for "deadline". To remedy this anomaly a new library package was added just underneath the root to allow deadlines to be set and read. At that point, all tasks were given a deadline: `Time.Last` (the farther point in time that can be represented on the processor) is given as the default for all tasks that have no use for deadlines. A configuration pragma (later evolved into a more elegant "specification aspect" in Ada 2012) was added to set the initial relative deadline to a task to control dispatching during activation. Ada's native support for asynchronous transfer of control allows tasks to catch deadline miss events in a very elegant manner.

The next step – and that is the interesting part – was to incorporate Baker's stack-based protocol [4] so that, while dispatching would be controlled by EDF, preemption levels would control protected access to shared data. In Baker's protocol, each task is assigned a static preemption level, and each protected object is assigned a ceiling value that is the maximum of the preemption levels of all tasks that call it. At run time a newly released task  $\tau_1$  can preempt the currently running task  $\tau_2$  if and only if  $\tau_1$ 's absolute deadline is earlier than  $\tau_2$ 's deadline *and*  $\tau_1$ 's preemption level is higher than the preemption level of any protected object currently in use. With preemption levels assigned inversely to relative deadline (the smaller the relative deadline, the higher the preemption level), Baker's protocol offers exactly the same guarantees as the Immediate Priority Ceiling Inheritance Protocol does for FP scheduling (cf. section 2.3). On this account, the EDF proposal was formulated so as to use the ceiling locking rules native in Ada 95, without change, with "regular" priority to represent preemption levels. EDF scheduling is defined by a new dispatching policy.

With that provision, the challenge was to extend Ada's definition of *dispatching point* to capture the run-time events that would cause scheduling to occur. In Ada 95 dispatching was defined in terms of ready queues, one per priority level, each ordered in a FIFO manner. Tasks have an assigned priority (set by configuration and changed by a call to `Dynamic Priority.Set Priority`) and an active priority – strictly higher than the base level – inherited when executing within a protected object. Preemption occurs when there is a non-empty ready queue at a priority higher than the current task.

The preemption rule specified in Baker's protocol does not give a complete model. One of the ramifications that need defining is what happens to a newly released task that is not entitled to preempt. The authors of [9, 28] derived an ingenious model that

would fit, with no bending or twisting, in the context of Ada's model based on ready queues. The EDF model was defined as follows:

1. All ready queues are ordered with EDF;
2. Execution within a protected object occurs at the ceiling priority of that object, as native to Ada 95;
3. Any task  $\tau$  that becomes runnable is placed on the highest-priority non-empty ready queue R such that  $\tau$ 's absolute deadline is less than the deadline of the task at the tail of queue R and  $\tau$ 's base priority is strictly greater than R. If no such R exists,  $\tau$  is added to the lowest-priority ready queue.

With this model the base priority of a task is *not* directly used to control dispatching. Moreover, with rule 3 the task's active priority is determined by the state of the other ready tasks: a task's active priority may therefore be *less* than its base priority.

In fact, rule 3 is quite sophisticated and not immediately obvious.

When no protected objects exist all released tasks will always be placed on the ready queue for the lowest priority. This will be the only ready queue that will ever have tasks "on" it. That queue is ordered by deadline, which gives EDF scheduling.

This also happens when a task is released when no protected objects are in use, because no task has been enqueued in a queue at priority level greater than the lowest.

When protected objects are in use, rule 3 has task  $\tau_r$  running at ceiling priority  $\pi_s$ : for preemption to occur the newly released task  $\tau_n$  must have a shorter deadline than  $\tau_r$ , and base priority higher than  $\pi_s$ . Task  $\tau_n$  is placed on the ready queue for level  $\pi_s$ , in front of  $\tau_r$  since it has a shorter deadline. Incidentally, this is the reason why ready tasks holding a resource can only be found at the tail of the ready queue for the ceiling priority of that resource. If  $\tau_n$  did not preempt  $\tau_r$ , it is placed on the lowest-priority ready queue: this happens because either  $\tau_n$ 's deadline is later than  $\tau_r$ 's or  $\tau_n$  has lower preemption level but shorter deadline than  $\tau_r$ . In the latter case  $\tau_n$  will preempt  $\tau_r$  as soon as that will lose ceiling priority. The attentive reader will notice at this point that in the latter scenario  $\tau_n$  will suffer a bounded duration of "deadline inversion", much like the Immediate Priority Ceiling Inheritance protocol causes bounded priority inversion. A further observation is that the proposed protocol only works if there are no protected objects with a ceiling set at the lowest priority. Such ceiling values must be prohibited, which limits user's freedom but not much really.

The proposal was approved by the language authorities and made it into the language specification for Ada 2005. To give the reader a taste of how language legality rules read to the non-initiated, the relevant specification clauses are reported below:

In Ada 2005 *EDF\_Across\_Priorities* is specified for a user-configurable range Low..High of priorities. All ready queues in this range are ordered by deadline. The task at the head of a queue is the one with the earliest deadline. A task dispatching point occurs for the currently running task T to which policy *EDF\_Across\_Priorities* applies when:

- a change to the deadline of T occurs;
- there is a task on the ready queue for the active priority of T with a deadline earlier than the deadline of T; or



- there is a non-empty ready queue for that processor with a higher priority than the active priority of the running task.

In these cases, the currently running task is said to be preempted and is returned to the ready queue for its active priority.

For a task *T* to which policy *EDF\_Across\_Priorities* applies, the base priority is not a source of priority inheritance; the active priority when first activated or while it is blocked is defined as the maximum of the following:

- the lowest priority in the range specified as *EDF\_Across\_Priorities* that includes the base priority of *T*;
- the priorities, if any, currently inherited by *T*;
- the highest priority *P*, if any, less than the base priority of *T* such that one or more tasks are executing within a protected object with ceiling priority *P* and task *T* has an earlier deadline than all other tasks on ready queues with priorities strictly less than *P*.

When a task *T* is first activated or becomes unblocked, it is added to the ready queue corresponding to this active priority. Until it becomes blocked again, the active priority of *T* remains no less than this value; it will exceed this value only while it is inheriting a higher priority. When the setting of the base priority of a ready task takes effect and the new priority is in a range specified as *EDF\_Across\_Priorities*, the task is added to the ready queue corresponding to its new active priority, as determined above.

Much of the textual changes from the formulation of the conceptual model to its legalese specification are a consequence of allowing multiple dispatching policies to coexist within one and the same system. The sole condition for it apply is that the priority ranges to which the different dispatching policies are attached do not overlap. This is a truly brilliant result that has given rise, among other works, to [10] that integrates EDF and FP schedulability analysis and can be rightly regarded as Ada's contribution to reviving the study of hybrid-priority scheduling.

Whereas the technology resulting from the contribution described in section 2 had made significant inroad in industrial use, the use of EDF support in Ada has stayed at the level of research tool. The good news however is that, presumably on the spin of the Ada effort, EDF support in Linux has come of age ([www.evidence.eu.com/sched\\_deadline.html](http://www.evidence.eu.com/sched_deadline.html)), which gives rise to hope for industrial use.

## 4 Conclusions

This paper has told the story of how two important language features of modern Ada have come about as direct application of state-of-the-art research result. Many individuals were behind the making of those features; one of them – Alan Burns, the person to whom this volume is dedicated – should be regarded as the key facilitator to all that. That theory meets technology is not a very frequent event. Having witness all of that happen twice in about a decade has been a great job for the author of this paper and a strong motivator to this write up.

## References

1. AbsInt GmbH. aiT WCET Analyzer, 2012. <http://www.absint.com/ait/>.
2. C.M. Bailey, E. Fyfe, T. Vardanega, and A. Wellings. The Use of Preemptive Priority-Based Scheduling for Space Applications. In *Proceedings of the Real-Time Systems Symposium*, volume 14, pages 253–257, Raleigh-Durham, NC (USA), December 1993. IEEE.
3. Ted Baker and Tullio Vardanega. Session summary: Tasking profiles. In *Proceedings of the 8th International Ada Real-Time Workshop*, volume XVII(5), pages 5–7, 1997.
4. T.P. Baker. A stack-based resource allocation policy for realtime processes. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 191–200, dec 1990.
5. A. Burns. Scheduling Hard Real-Time Systems: a Review. *Software Engineering Journal*, 6(3):116–128, 1991.
6. Alan Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S.H. Son, editor, *Advances in Real-Time Systems*. 1994.
7. Alan Burns and Andy Wellings. HRT-HOOD: A design method for hard-real-time. 6(1):73–114, 1994.
8. Alan Burns and Andy Wellings. *HRT-HOOD(TM): A Structured Design Method for Hard Real-Time Ada Systems*. Amsterdam, NL, 1995. ISBN 0-444-82164-3.
9. Alan Burns, Andy J. Wellings, and S. Tucker Taft. Supporting deadlines and edf scheduling in ada. In Albert Llamas and Alfred Strohmeier, editors, *Reliable Software Technologies - Ada-Europe 2004, 9th Ada-Europe International Conference on Reliable Software Technologies, Palma de Mallorca, Spain, June 14-18, 2004, Proceedings*, volume 3063 of *Lecture Notes in Computer Science*, pages 156–165. Springer, 2004.
10. Alan Burns, Andy J. Wellings, and Fengxiang Zhang. Combining EDF and FP scheduling: Analysis and implementation in Ada 2005. In *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies, Ada-Europe '09*, pages 119–133, Berlin, Heidelberg, 2009. Springer-Verlag.
11. Giorgio C. Buttazzo. Rate Monotonic vs. EDF: Judgment Day. *Real-Time Systems*, 29(1):5–26, 2005.
12. E.W. Dijkstra. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *CACM*, 18(8):453–457, 1975.
13. Brian Dobbing and George Romanski. The Ravenscar profile: Experience report. In *Proceedings of the 9th International Real-Time Ada Workshop*, volume XIX(2), pages 28–32, 1999.
14. José C. Palencia Gutiérrez and Michael González Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *RTSS*, pages 328–339, 1999.
15. Niklas Holsti and Thomas Långbacka. Impact of a restricted tasking profile: The case of the GOCE platform application software. In *Ada-Europe*, pages 92–101, 2003.
16. Intermetrics. *Ada 95 Rationale: Language and Standard Libraries.*, 1995. Available from Springer-Verlag, LNCS no. 1247.
17. ISO SC22/WG9. Ada Reference Manual. Language and Standard Libraries. Consolidated Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1, 2005.
18. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.
19. Silvia Mazzini, Massimo D’Alessandro, Marco Di Natale, Giuseppe Lipari, and Tullio Vardanega. Issues in mapping HRT-HOOD to UML. In *ECRTS 2003*, pages 221–228, 2003.
20. J. Mukerji and J. Miller. MDA Guide Version 1.0.1. <http://www.omg.org/docs/omg/03-06-01.pdf>, 2003.

21. Marco Panunzio and Tullio Vardanega. A component model for on-board software applications. In *EUROMICRO-SEAA*, pages 57–64, 2010.
22. Marco Panunzio and Tullio Vardanega. Ada ravenstar code archetypes for component-based development. In *Ada-Europe*, pages 1–17, 2012.
23. Bound-T Execution Time Analyzer, Home Page. <http://www.bound-t.com>, June 2003.
24. Tullio Vardanega. Ravenscar design patterns?: reflections on use of the ravenstar profile. In *Proceedings of the 12th international workshop on Real-time Ada*, IRTAW '03, pages 65–73, New York, NY, USA, 2003. ACM.
25. Tullio Vardanega and Gert Caspersen. Using the Ravenscar Profile for space applications: The OBOSS case. In Michael González-Harbour, editor, *Proceedings of the International Workshop on Real-Time Ada Issues*, volume XXI, pages 96–104, 2001.
26. Tullio Vardanega and Jan van Katwijk. A software process for the construction of predictable on-board embedded real-time systems. *Softw., Pract. Exper.*, 29(3):235–266, 1999.
27. Tullio Vardanega, Juan Zamorano, and Juan Antonio de la Puente. On the dynamic semantics and the timing behavior of Ravenscar kernels. *Real-Time Systems*, 29(1):59–89, 2005.
28. Alexandros Zerzelidis, Alan Burns, and Andy J. Wellings. Correcting the EDF protocol in Ada 2005. In *Proceedings of the 13th international workshop on Real-time Ada*, IRTAW '07, pages 18–22, New York, NY, USA, 2007. ACM.

# A Linear Model for Setting Priority Points in Soft Real-Time Systems

Bryan C. Ward, Jeremy P. Erickson and James H. Anderson

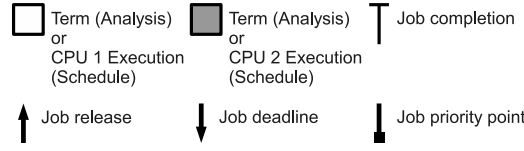
Department of Computer Science  
The University of North Carolina at Chapel Hill

**Abstract.** The earliest-deadline-first (EDF) scheduling algorithm, while not optimal for globally-scheduled hard real-time systems, can support any feasible task system with bounded lateness. Furthermore, global-EDF-like (GEL) scheduling algorithms, which prioritize jobs by assigning them fixed *priority points* based on per-task *relative priority points*, have been shown to share this same property. Some such algorithms exhibit better lateness bounds than G-EDF. This paper surveys existing research on bounded-lateness analysis for GEL schedulers, and formulates one such analysis technique called *compliant vector analysis (CVA)* as a *linear program (LP)*. Using this LP, per-task relative priority points can be set to optimize for application-specific lateness constraints, and minimize average and/or maximal lateness bounds. An empirical study is presented that compares the lateness bounds of a variety of GEL schedulers and analysis techniques on randomly generated task systems.

## 1 Introduction

For some types of computing workloads, such as multimedia applications, it is not necessary to use schedulers amenable to *hard real-time analysis* that ensures that all deadlines are met. A larger system utilization can often be supported by instead using a scheduler that is amenable to *soft real-time analysis* that ensures that the *lateness* of any job, or the amount of time between its deadline and its completion, remains bounded.

Although bounded lateness is achievable by a wide range of real-time scheduling algorithms with no utilization loss [6], one class of such algorithms, called *G-EDF-like (GEL)* algorithms [7], is of particular interest because such algorithms are straightforward to implement. In a GEL algorithm, each task has a *relative priority point*, and each job has an *absolute priority point* defined by adding its task's relative priority point to its release. Jobs are scheduled globally on an earliest-absolute-priority-point-first basis. Devi and Anderson [2] proved that G-EDF itself has bounded lateness, and subsequently improvements to this original analysis were proposed in [3–5]. Erickson and Anderson [3] further proposed the *global fair lateness (G-FL)* scheduler, a GEL scheduler that provably provides the smallest maximum (over all tasks) lateness bound possible using current analysis for GEL schedulers. In addition, Leontyev and Anderson [6] analyzed a much more general class of schedulers in which not all processors are necessarily fully available. However, due to its generality, this analysis is not as tight as the analysis in [3].



**Fig. 1.** Key for all figures in this paper.

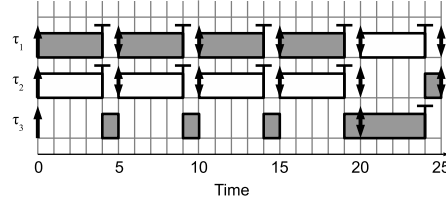
Lateness bounds under GEL schedulers can be computed using an algorithm described in [5], and the maximum lateness bound can be minimized by using G-FL as described in [3]. However, G-FL will ensure the *same* lateness bound for all tasks in a system, even though it is sometimes possible to reduce the lateness of some tasks while maintaining the same maximum lateness bound as G-FL. Furthermore, [3] does not propose a straightforward way to optimize criteria other than maximum lateness. In this paper, we propose to model current lateness analysis using linear programming in order to broaden the set of achievable lateness criteria.

## 2 SRT Lateness Bounds

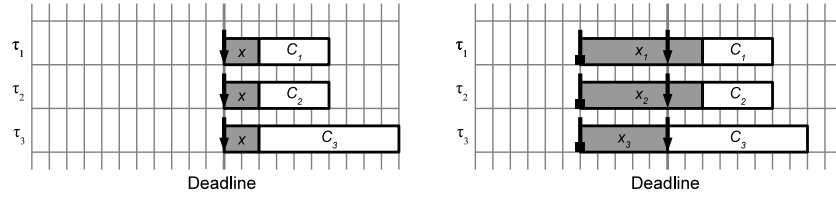
We consider a system  $\tau$  of  $n$  sporadic tasks  $\{\tau_1, \tau_2, \dots, \tau_n\}$  running on  $m$  processors. Each task  $\tau_i$  is defined by a tuple  $(C_i, T_i)$ , where  $C_i$  is the worst-case per-job execution time of  $\tau_i$  and  $T_i$  is the minimum separation time between jobs of  $\tau_i$ . We assume *implicit deadlines*, i.e., if a job of  $\tau_i$  is released at time  $r$ , then it has a deadline at time  $r + T_i$ . The *utilization* of  $\tau_i$ ,  $U_i = C_i/T_i$ , is the long-term processor share it requires. Under any GEL scheduler,  $\tau_i$  has a scheduler-defined *relative priority point* (PP)  $Y_i$  and a job of  $\tau_i$  released at time  $r$  has an *absolute PP* at time  $r + Y_i$ . Jobs with earlier PPs are prioritized over those with later PPs. If a job completes at time  $c$ , then it has a *response time* of  $c - r$  and a *lateness* of  $c - (r + T_i)$ .

In our analysis, we assume that time is continuous. For each task  $\tau_i$ , we also assume that  $U_i \leq 1$  and that  $\sum_{\tau_i \in \tau} U_i \leq m$ . As demonstrated in [2, 6], these conditions are necessary to achieve bounded lateness. We further assume that  $n > m$ , because otherwise each task can execute on its own processor, and no job of task  $\tau_i$  will have a response time exceeding  $C_i$ .

*Example.* We will illustrate several forms of lateness-bound analysis through an example task system  $\tau = \{(4, 5), (4, 5), (8, 20)\}$  on  $m = 2$  processors. We will show two types of figures: analysis and actual schedules. We emphasize that the figures depicting analysis do not show actual schedules. All presented forms of analysis determine how long a job can execute after its PP. Such analysis pessimistically assumes that each job cannot begin execution until after some non-negative constant after its PP. Both this constant and the worst-case execution time of the task are terms of the lateness bound. In our analysis figures, we align deadlines rather than PPs to depict lateness bounds. The key for all figures is shown in Fig. 1.



(a) Example G-EDF schedule.



(b) Devi and Anderson's lateness analysis

(c) CVA lateness analysis (see Sec. 2.2).

**Fig. 2.** (a) shows a G-EDF schedule of the example task system  $\tau = \{(4,5), (4,5), (8,20)\}$ , and (b) and (c) show the lateness analysis of G-EDF from [2] and CVA [3–5], respectively. In (b) and (c), the deadlines of all tasks are aligned to depict how lateness bounds compare among tasks.

## 2.1 Devi and Anderson's Lateness Bound

Lateness bounds for G-EDF were first provided by Devi and Anderson [2]. Let  $C_{\min}$  be the smallest  $C_i$  value of  $\tau_i$  in  $\tau$ . (In our example,  $C_{\min} = 4$ .) Letting

$$x = \frac{\sum_{m-1 \text{ largest } C_i} C_i - C_{\min}}{m - \sum_{m-2 \text{ largest } U_i}}, \quad (1)$$

Devi and Anderson [2] established  $x + C_i$  as a lateness bound for  $\tau_i$ . In our example,

$$x = \frac{C_3 - C_1}{m} = \frac{8 - 4}{2} = 2. \quad (2)$$

Therefore,  $\tau_1$  and  $\tau_2$  each have a lateness bound of  $2 + 4 = 6$ , and  $\tau_3$  has a lateness bound of  $2 + 8 = 10$ . A G-EDF schedule of  $\tau$  is depicted in Fig. 2(a), and Devi and Anderson's analysis is depicted in Fig. 2(b).

## 2.2 Compliant Vector Analysis

Erickson *et. al.* [4] introduced *compliant-vector analysis* (CVA), an improved method to analyze lateness under G-EDF. This analysis was later extended in [3, 5] to apply to systems with arbitrary *PP* assignments. We present this more general analysis and apply it to G-EDF, which is modeled by setting  $Y_i = T_i$  for each  $\tau_i$ .

CVA is similar to the analysis described in [2], but rather than defining a single  $x$  for the task system so that the response time of each task  $\tau_i$  is at most  $T_i + x + C_i$ , a

vector  $x = \langle x_1, x_2, \dots, x_n \rangle$  is derived that ensures that the response time of each task  $\tau_i$  is at most

$$Y_i + x_i + C_i. \quad (3)$$

Note that response times are determined based on  $Y_i$  rather than  $T_i$ .

We apply the following optimization rule before analysis.

**PP Reduction Rule.** If all relative PPs in the system are decreased by the same constant, then the ordering of absolute PPs will not change, so the resulting schedule will not change. Under CVA, lateness bounds are minimized for a given GEL scheduler when, analytically, all relative PPs are reduced by the smallest relative PP, so that the smallest relative PP becomes zero [3].

We note that a more complicated alternative optimization rule is presented in [4], which is not compatible with the PP Reduction Rule because it assumes that  $Y_i = T_i$ . For simplicity of presentation we do not describe this rule here, but we consider it in our experimental study in Sec. 3.

In order to find lateness bounds for a system, the vector  $x$  must be computed. In [3], the authors demonstrate that there exists a constant  $s$  such that  $x_i = v_i(s)$  for each  $i$  where

$$v_i(s) = \frac{s - C_i}{m}. \quad (4)$$

As a result, determining the correct value for  $s$  is sufficient to determine the value of the entire vector  $x$ . In order to determine this value of  $s$ , the authors of [3] define

$$S_i = C_i \cdot \max \left\{ 0, 1 - \frac{Y_i}{T_i} \right\}, \quad (5)$$

$$G(s) = \sum_{m-1 \text{ largest}} (v_i(s)U_i + C_i - S_i), \quad (6)$$

and

$$S(\tau) = \sum_{\tau_i \in \tau} S_i. \quad (7)$$

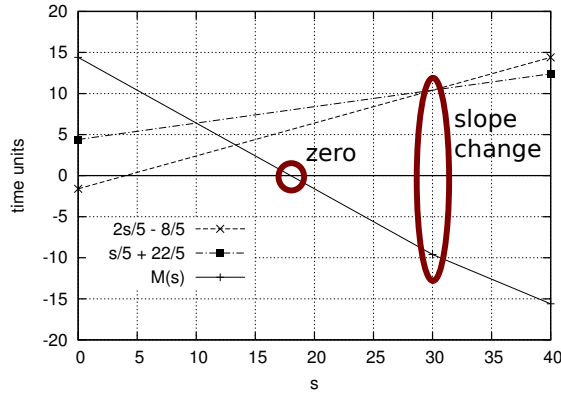
They demonstrate that the smallest lateness bounds valid under CVA occur iff

$$s = G(s) + S(\tau). \quad (8)$$

An algorithm to compute the necessary  $s$ , along with a proof that a unique such  $s$  exists, is provided in [5]. It works by finding the zero of

$$M(s) = G(s) + S(\tau) - s. \quad (9)$$

By the uniqueness of  $s$ ,  $M(s)$  will have one zero. Because each of (4)–(7) above is piecewise linear with respect to  $s$ ,  $M(s)$  is piecewise linear with respect to  $s$ . The computation algorithm works by attempting to find a zero between each pair of points where the slope of  $M(s)$  changes.



**Fig. 3.** Graph of expressions used to compute CVA bounds for G-EDF.

Here we will demonstrate the computation algorithm via our example system. For G-EDF, each relative PP  $Y_i$  is initially  $T_i$ . However, after applying the PP Reduction Rule,  $Y_1 = Y_2 = 0$  and  $Y_3 = 15$ . By (5),

$$S_1 = S_2 = 4 \cdot (1 - 0) = 4 \quad (10)$$

and

$$S_3 = 8 \cdot \left(1 - \frac{15}{20}\right) = 2. \quad (11)$$

Thus, by (6):

$$\begin{aligned} G(s) &= \max \left\{ \frac{s-4}{2} \cdot \frac{4}{5} + 4 - 4, \frac{s-8}{2} \cdot \frac{8}{20} + 8 - 2 \right\} \\ &= \max \left\{ \frac{2s}{5} - \frac{8}{5}, \frac{s}{5} + \frac{22}{5} \right\}, \end{aligned} \quad (12)$$

and by (7),

$$S(\tau) = 4 + 4 + 2 = 10. \quad (13)$$

Because  $S(\tau)$  is a constant and  $s$  is a simple linear function, by (9), the slope of  $M(s)$  can only change where the slope of  $G(s)$  changes. The expressions in (12) and the resulting  $M(s)$  are depicted in Fig. 3.

In order to determine where the slope of  $M(s)$  can change, we now compute where the two expressions in (12) intersect:

$$\begin{aligned} \frac{2s}{5} - \frac{8}{5} &= \frac{s}{5} + \frac{22}{5} \\ \frac{s}{5} &= 6 \\ s &= 30. \end{aligned}$$

Therefore, for  $s \leq 30$ ,

$$G(s) = \frac{s}{5} + \frac{22}{5}, \quad (14)$$



and for  $s > 30$ ,

$$G(s) = \frac{2s}{5} - \frac{8}{5}. \quad (15)$$

We will first try to find a zero of  $M(s)$  assuming  $s \leq 30$ . In this case,

$$\begin{aligned} M(s) &= \{\text{By (9)}\} \\ &= G(s) + S(\tau) - s \\ &= \{\text{By (13) and (14)}\} \\ &= \frac{s}{5} + \frac{22}{5} + 10 - s \\ &= \{\text{Rearranging}\} \\ &= -\frac{4s}{5} + \frac{72}{5}. \end{aligned} \quad (16)$$

Setting  $M(s)$  to 0 in (16),

$$s = \left(\frac{72}{5}\right) \left(\frac{5}{4}\right) = 18. \quad (17)$$

$s = 18$  satisfies our assumption that  $s \leq 30$ , so 18 is indeed a zero of  $M(s)$ . As demonstrated in [4], the zero of  $M(s)$  is unique, so we do not need to consider  $s > 30$ . Therefore, by (4),

$$x_1 = x_2 = \frac{18 - 4}{2} = 7, \quad (18)$$

and by (3), the maximum response time of any job of  $\tau_1$  or  $\tau_2$  is  $0 + 7 + 4 = 11$ , which gives a maximum lateness bound of  $11 - 5 = 6$ . Also, by (4)

$$x_3 = \frac{18 - 8}{2} = 5, \quad (19)$$

and by (3), the maximum response time of any job of  $\tau_3$  is  $15 + 5 + 8 = 28$ , which gives a maximum lateness bound of  $28 - 20 = 8$ . These results are depicted graphically in Fig. 2(c).

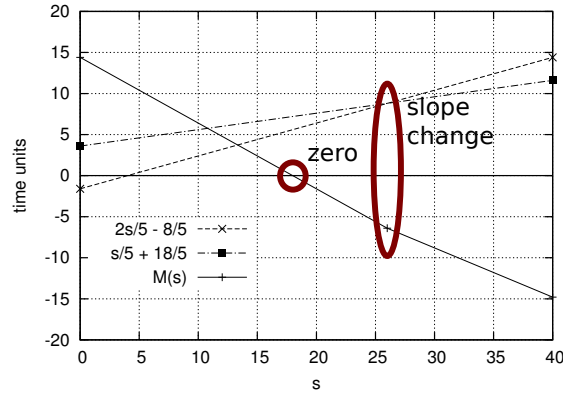
### 2.3 G-FL

In [3], the authors also propose G-FL, an algorithm that has the optimal maximum lateness bound over all GEL schedulers using CVA. Rather than using  $Y_i = T_i$ , G-FL uses  $Y_i = T_i - \frac{m-1}{m}C_i$ . Therefore, in our example system,

$$Y_1 = Y_2 = 5 - \frac{1}{2} \cdot 4 = 3 \quad (20)$$

and

$$Y_3 = 20 - \frac{1}{2} \cdot 8 = 16. \quad (21)$$



**Fig. 4.** Graph of expressions used to compute CVA bounds for G-FL.

However, by the PP Reduction Rule from Sec. 2.2, we can instead use  $Y_1 = Y_2 = 0$  and  $Y_3 = 13$  in the analysis. Here, we repeat the analysis from Sec. 2.2 as applied to G-FL.

By (5),

$$S_1 = S_2 = 4 \cdot (1 - 0) = 4 \quad (22)$$

and

$$S_3 = 8 \cdot \left(1 - \frac{13}{20}\right) = \frac{14}{5}. \quad (23)$$

Thus, by (6):

$$\begin{aligned} G(s) &= \max \left\{ \frac{s-4}{2} \cdot \frac{4}{5} + 4 - 4, \frac{s-8}{2} \cdot \frac{8}{20} + 8 - \frac{14}{5} \right\} \\ &= \max \left\{ \frac{2s}{5} - \frac{8}{5}, \frac{s}{5} + \frac{18}{5} \right\}, \end{aligned} \quad (24)$$

and by (7),

$$S(\tau) = 4 + 4 + \frac{14}{5} = \frac{54}{5}. \quad (25)$$

The expressions in (24) and the resulting  $M(s)$  are displayed in Fig. 4. As before,  $M(s)$  can change slope only when  $G(s)$  changes slope, so we now compute where the two expressions in (24) intersect:

$$\begin{aligned} \frac{2s}{5} - \frac{8}{5} &= \frac{s}{5} + \frac{18}{5} \\ \frac{s}{5} &= \frac{26}{5} \\ s &= 26. \end{aligned}$$

Therefore, for  $s \leq 26$ ,

$$G(s) = \frac{s}{5} + \frac{18}{5}, \quad (26)$$

and for  $s > 26$ ,

$$G(s) = \frac{2s}{5} - \frac{8}{5}. \quad (27)$$

We will first try to find a zero of  $M(s)$  assuming  $s \leq 26$ . In this case,

$$\begin{aligned} M(s) &= \{\text{By (9)}\} \\ &G(s) + S(\tau) - s \\ &= \{\text{By (25) and (26)}\} \\ &\frac{s}{5} + \frac{18}{5} + \frac{54}{5} - s \\ &= \{\text{Rearranging}\} \\ &-\frac{4s}{5} + \frac{72}{5}. \end{aligned} \quad (28)$$

Compared to Sec. 2.2, notice that the increased  $S_3$  term in  $G(s)$  cancels out the increased  $S_3$  term in  $S(\tau)$ , so (16) and (28) are identical.

Setting  $M(s)$  to 0 in (28),

$$s = \left(\frac{72}{5}\right) \left(\frac{5}{4}\right) = 18. \quad (29)$$

$s = 18$  satisfies our assumption that  $s \leq 26$ , so 18 is indeed a zero of  $M(s)$ . As before, because the zero of  $M(s)$  must be unique, we do not need to consider  $s > 26$ . Therefore, by (4),

$$x_1 = x_2 = \frac{18 - 4}{2} = 7, \quad (30)$$

and by (3), the maximum response time of any job of  $\tau_1$  or  $\tau_2$  is  $0 + 7 + 4 = 11$ , which gives a maximum lateness bound of  $11 - 5 = 6$ . Also, by (4)

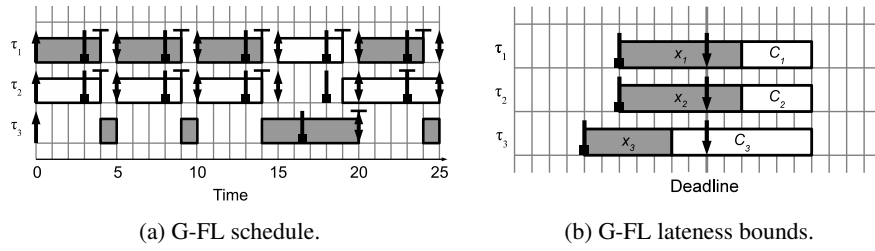
$$x_3 = \frac{18 - 8}{2} = 5. \quad (31)$$

By (3), the maximum response time of any job of  $\tau_3$  is  $13 + 5 + 8 = 26$ , which gives a maximum lateness bound of  $28 - 20 = 6$ . Thus, all tasks have the same lateness bound, as claimed.

A G-FL schedule of the example system is depicted in Fig. 5(a), and its associated analysis is depicted in Fig. 5(b). Although all analytical bounds are identical, due to the pessimism of the bounds the actual lateness is not necessarily identical.

## 2.4 Linear Programming

Next, we show how CVA can be formulated as a LP. Such a formulation allows system designers to optimize lateness bounds for arbitrary (linear) optimization objectives by adjusting PPs. We show how to use this LP formulation of CVA to minimize average lateness, and to minimize average lateness while maintaining the same maximum



**Fig. 5.** Illustration of the G-FL schedule (a) and lateness bounds (b). Note that the deadlines are aligned in (b) to depict how lateness compares among tasks.

lateness as G-FL. Furthermore, using this LP formulation, average lateness can be minimized while ensuring that application-specific lateness tolerances are all satisfied (if it possible under CVA).

Recall that under CVA, to find the lateness bound of a task  $\tau_i$ ,  $x_i = v_i(s)$  must be computed using Equations (4)-(8). We show how these equations can be reformulated as constraints in a LP that minimizes average lateness (or possibly other objectives). We assume that  $x_i$ ,  $Y_i$ ,  $S_i$ ,  $S(\tau)$ ,  $G(s)$ , and  $s$  are variables, and we also introduce the auxiliary variables  $b$  and  $z_i$ . All other values are constants (*i.e.*,  $U_i$ ,  $C_i$ , and  $m$ ).

**Constraint Set 1** The linear constraints corresponding to (4) are given by

$$\forall i : x_i = \frac{s - C_i}{m}.$$

**Constraint Set 2** The linear constraints corresponding to (5) are given by

$$\forall i : S_i \geq 0; \quad S_i \geq C_i(1 - Y_i/T_i).$$

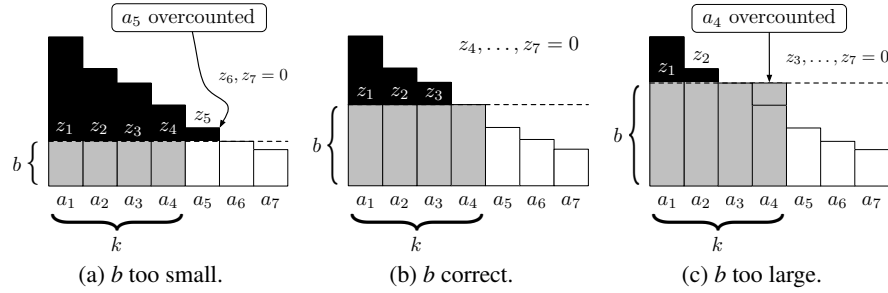
The two constraints in Constraint Set 2 model the two terms of the max in (5). If  $C_i(1 - Y_i/T_i) \leq 0$ , then the constraint  $S_i \geq 0$  ensures that the value of  $S_i$  is correct. If  $C_i(1 - Y_i/T_i) \geq 0$ , then the constraint  $S_i \geq C_i(1 - Y_i/T_i)$  ensures the value of  $S_i$  is correct.

The next equation to be linearized, (6), is less straightforward. We first show how to minimize the sum of the largest  $k$  elements in a set  $A = \{a_1, \dots, a_n\}$  using only linear constraints, by an approach similar to one presented in [8]. The intuition behind this approach is shown in Fig. 6. This figure corresponds to the following linear program

$$\begin{aligned} &\text{Minimize : } G \\ &\text{Subject to : } G = kb + \sum_{i=1}^n z_i, \\ &\quad z_i \geq 0 \quad \forall i, \\ &\quad z_i \geq a_i - b \quad \forall i, \end{aligned}$$

in which  $G$ ,  $b$ , and  $z_i$  are variables and both  $k$  and  $a_i$  are constants. In Fig. 6, the term  $kb$  corresponds to the gray-shaded area, and  $\sum_{i=1}^n z_i$  corresponds to the black-shaded area. When  $G$  is minimized, it is equal to the sum of the largest  $k$  elements in  $A$ . This is achieved when  $z_i = 0$  for each element  $a_i$  that is not one of the  $k$  largest elements in  $A$ , and  $b$  is at most the  $k^{th}$  largest element in  $A$ , as is shown in Fig. 6 (b).

Using this technique, we can formulate (6) as a set of linear constraints.



**Fig. 6.** Illustration of the auxiliary variables used to sum the largest  $k$  elements in the set  $A = \{a_1, \dots, a_n\}$ . The total of the gray and black shaded areas is equal to  $G$ . The gray areas correspond to  $kb$  while the black areas correspond to positive  $z_i$ 's. When  $G$  is minimized, as in (b),  $G$  is equal to the sum of the largest  $k$  elements in  $A$ . As is shown in (a) and (c), if  $b$  is too small or too large then  $G$  will be larger than the maximum  $k$  elements in  $A$ . Note that elements of  $A$  are depicted in sorted order only for visual clarity.

**Constraint Set 3** The linear constraints corresponding to (6) are given by

$$G(s) = b(m-1) + \sum_{\tau_i \in \tau} z_i,$$

$$\forall i : z_i \geq 0,$$

$$\forall i : z_i \geq x_i U_i + C_i - S_i - b.$$

In the optimization objectives we consider,  $G(s)$  is not itself explicitly minimized, as in the example linear program. However, the objective functions we consider minimize average lateness subject to some constraints, and because the lateness of all tasks is a function of  $G(s)$ ,  $G(s)$  is also minimized.<sup>1</sup>

**Constraint Set 4** The linear constraint corresponding to (7) is given by

$$S(\tau) = \sum_{\tau_i \in \tau} S_i.$$

**Constraint Set 5** The linear constraint corresponding to (9) is given by

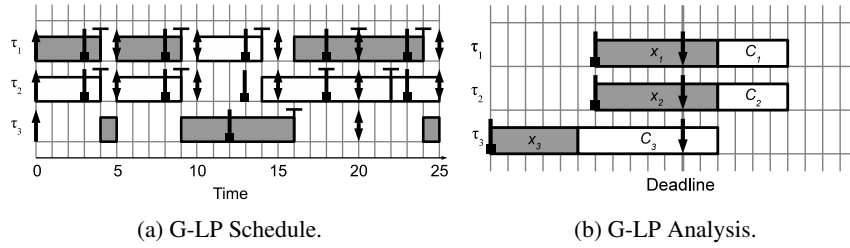
$$G(s) + S(\tau) = s.$$

Constraint Sets 1–5 model CVA through a set of linear constraints. Next we show how these constraints can be coupled with an optimization objective to find optimal priority point settings under CVA. We refer to schedulers produced by setting priority points using our LP formulation of CVA, as *G-LP*. First, we show how *G-LP* can be optimized to minimize average lateness.

**Minimizing Average Lateness.** The following linear program may be solved to minimize average lateness under CVA.

$$\begin{aligned} &\text{Minimize : } \sum_{\tau_i \in \tau} Y_i + x_i \\ &\text{Subject to : Constraint Sets 1–5} \end{aligned}$$

<sup>1</sup> If  $G(s)$  is not minimized, then the resulting lateness bounds are still correct, just not as tight.



**Fig. 7.** Illustration of the G-FL schedule (a) and lateness bounds (b). Note that the deadlines are aligned in (b) to depict how lateness compares among tasks.

Note that average lateness is given by  $\sum_{\tau_i \in \tau} (Y_i + x_i + C_i - T_i)/n$ , but  $C_i$ ,  $T_i$ , and  $n$  are all constants that are not necessary to include in the optimization objective. We denote G-LP optimized for average lateness as G-LP-AL.

While G-LP-AL is optimal with respect to average lateness, as is shown experimentally in Sec. 3, the lateness of some tasks may be larger than the maximum lateness bound of G-FL, which we denote  $L_{max}$ . Next, we show how to optimize the average lateness of all tasks while maintaining a maximum lateness no greater than  $L_{max}$ .

*Minimizing Average Lateness from G-FL.* The following linear program may be solved to minimize the average lateness under CVA while maintaining the same maximum lateness bound as G-FL.

$$\begin{aligned} \text{Minimize : } & \sum_{\tau_i \in \tau} Y_i + x_i \\ \text{Subject to : } & \forall i : Y_i + x_i + C_i - T_i \leq L_{max},^2 \\ & \text{Constraint Sets 1-5} \end{aligned}$$

As before, the constants  $C_i$ ,  $T_i$  and  $n$  are omitted from the objective function. We denote the scheduler produced by this program as G-LP-FL.

For our previous example task system, both G-LP-AL and G-LP-FL set the priority point of  $\tau_3$  to  $Y_3 = 12$ , instead of  $Y_3 = 16$  as in G-FL. This priority point assignment improves the lateness bound of  $\tau_3$  from 6 under G-FL to 2 for G-LP. A schedule and a visual depiction of the analysis are given in Fig. 7. Table 1 summarizes the lateness bounds under different schedulers and analysis techniques for our running example. Note that, in general, G-LP-AL and G-LP-FL do not produce the same schedules, as will be evident in Sec. 3.

We note that the LP formulation of CVA can be used and extended to other optimization objectives, perhaps most notably, application-specific optimization objectives. For example, an LP solver can be used to assign PPs to ensure application-specific lateness tolerances are satisfied (if possible under CVA), or to maximize total system utility under some linear definitions of lateness-based utility.

<sup>2</sup> Application-specific per-task lateness tolerances could be used instead of  $L_{max}$ .

| Scheduler | Analysis              | $\tau_1$ | $\tau_2$ | $\tau_3$ |
|-----------|-----------------------|----------|----------|----------|
| G-EDF     | Devi and Anderson [2] | 6        | 6        | 10       |
| G-EDF     | CVA [3]               | 6        | 6        | 8        |
| G-FL      | CVA [3]               | 6        | 6        | 6        |
| G-LP-AL   | CVA [3] & LP          | 6        | 6        | 2        |
| G-LP-FL   | CVA [3] & LP          | 6        | 6        | 2        |

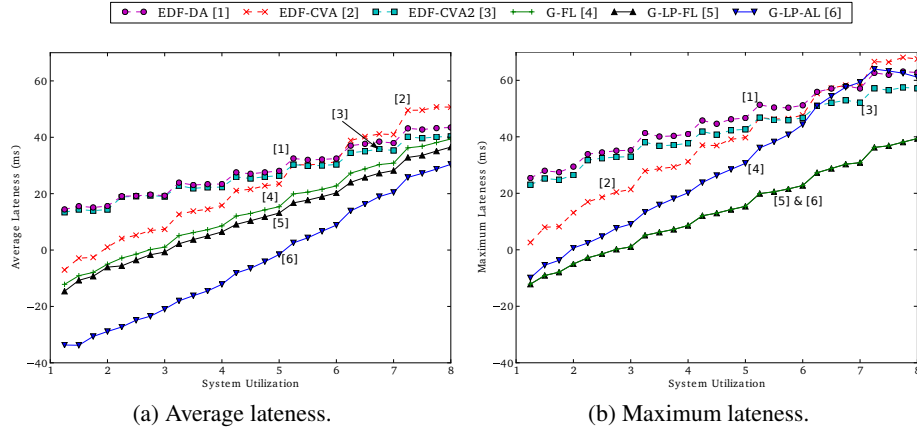
**Table 1.** Lateness bounds under different schedulers and analysis assumptions for the example task system  $\tau = \{(4, 5), (4, 5), (8, 20)\}$  on  $m = 2$  processors.

### 3 Experiments

In this section, we present experiments that demonstrate how G-LP can improve lateness bounds over existing scheduling algorithms. We generated random task sets using a similar experimental design as in previous studies (*e.g.*, [3]). We generated implicit-deadline task sets in which per-task utilizations were distributed either uniformly, bimodally, or exponentially. For task sets with uniformly distributed utilizations, per-task utilizations were chosen to be *light*, *medium* or *heavy*, which correspond to utilizations uniformly distributed in the range  $[0.001, 0.1]$ ,  $[0.1, 0.4]$ , or  $[0.5, 0.9]$ , respectively. For task systems with bimodally distributed utilizations, per-task utilizations were chosen from either  $[0.001, 0.5]$  or  $[0.5, 0.9]$  with respective probabilities of 8/9 and 1/9; 6/9 and 3/9; or 4/9 and 5/9. For task systems with exponentially distributed utilization, per-task utilizations were distributed exponentially with a mean of 0.10, 0.25 or 0.50. The periods of all tasks were generated using an integral uniform distribution between  $[3 \text{ ms}, 33 \text{ ms}]$ ,  $[10 \text{ ms}, 100 \text{ ms}]$  and  $[50 \text{ ms}, 250 \text{ ms}]$  for tasks with *short*, *moderate*, and *long* periods, respectively. We considered a system with  $m = 8$  processors, as clustered scheduling typically is preferable to global scheduling for larger processor counts [1]. For each per-task utilization and period distribution combination, 1,000 task sets were generated for each total system utilization value in  $\{1.25, 1.50, \dots, 8.0\}$  (we did not consider task systems with utilizations of one or less as they are schedulable on one processor).

For each generated task system, we evaluated lateness bounds under Devi and Anderson’s analysis of G-EDF [2] (EDF-DA), CVA analysis of G-EDF scheduling (EDF-CVA) using the PP Reduction Rule, CVA analysis of G-EDF using the alternative optimization rule in [4] (EDF-CVA2), G-FL, and G-LP when optimizing for average lateness subject to the constraint that no task has a lateness bound greater than it would have under G-FL (G-LP-FL), and G-LP when optimizing for average lateness without constraining maximum lateness (G-LP-AL). In Fig. 8, we show a results from a representative combination of per-task utilization and period distributions, medium and moderate, respectively. Note that the lateness bound results are analytical, and that in an actual schedule observed latenesses may be smaller.

**Observation 1.** EDF-CVA2 dominates EDF-DA with respect to both average and maximum lateness bounds. For task systems with small utilizations, the PP Reduction Rule of EDF-CVA outperforms the optimization in EDF-CVA2 [4];



**Fig. 8.** Average and maximum lateness bounds under the different analysis techniques and schedulers discussed for a system with moderate periods and uniform medium utilizations.

however, the converse is true for large utilizations. Furthermore, for large utilizations, EDF-DA can have lower lateness bounds than EDF-CVA.

This can be seen in insets (a) and (b) of Fig. 8. EDF-CVA2 always dominates EDF-DA, and EDF-CVA has slightly smaller lateness bounds than EDF-DA for task systems with total utilization less than six, while for utilizations greater than six, EDF-DA has better lateness bounds than EDF-CVA. Although the optimization from [4] in EDF-CVA2 performs very well for task systems with large utilizations, it is only applicable if  $Y_i = T_i$ , for all  $i$ . The PP Reduction Rule is applicable to any GEL scheduler.

**Observation 2.** All three of the GEL schedulers we considered with PPs different from G-EDF, namely, G-FL, G-LP-AL and G-LP-FL, had smaller average lateness bounds than G-EDF as determined via either CVA or Devi and Anderson's analysis.

These three scheduling algorithms optimize, with respect to CVA, either the average or maximum lateness of all tasks by moving PPs. Therefore, these algorithms should have smaller average lateness bounds than G-EDF. This can be observed in inset (a) of Fig. 8.

**Observation 3.** The maximum lateness bounds of G-LP-FL and G-FL are the same, but the average lateness bound of G-LP-FL is at worst the average lateness bound of G-FL.

Based on the constraints and the optimization objective of G-LP-FL, the average and maximum lateness bounds are provably no worse than G-FL. As is seen in inset (a) of Fig. 8, the improvement in average lateness in the task systems seen in our experiments was usually only a few ms.

**Observation 4.** Average lateness bounds were lower under G-LP-AL than under G-FL and G-LP-FL. This improvement in average lateness is made possible by allowing for increased maximum lateness.



As a result of the LP optimization, G-LP-AL is optimal with respect to average lateness under CVA. In inset (a) of Fig. 8, we see that the average lateness of G-LP-AL is always smaller than all other schedulers, often by 10-20ms or more. However, the maximum lateness bounds of G-LP-AL are larger than G-FL and G-LP-FL. In most observed cases, such as in Fig. 8 (b), lateness bounds of G-LP-AL were less than or commensurate with G-EDF lateness bounds as determined by either CVA or Devi and Anderson's analysis, though in some cases the maximum lateness was greater than G-EDF by 10-20ms. From these results, G-LP-AL may be practical in many applications.

We note that the lateness bounds of G-LP-AL in comparison to G-FL and G-LP-FL demonstrate that the LP solver has considerable flexibility in choosing priority points to optimize for certain lateness criteria. If some tasks have larger lateness tolerances than others, the PPs of the more tolerant tasks can be increased to improve the lateness bounds of the less tolerant tasks. This gives system designers much more flexibility to optimize the scheduler for application-specific needs.

## 4 Conclusions

In this paper, we surveyed existing schedulers and analysis techniques for soft real-time systems with bounded deadline lateness, and showed how to model compliant vector analysis (CVA) as a linear program. The linear formulation of CVA allows the priority points of tasks to be optimally (with respect to CVA) placed to minimize, for example, average or maximum lateness. Scheduling algorithms that result from moving priority points improve average lateness over existing algorithms. Furthermore, the linear formulation of CVA gives system designers flexibility to adapt the scheduling algorithm that is used to the lateness requirements of specific applications. We also generated random task systems and evaluated lateness bounds under a number of different scheduling algorithms and analysis techniques.

## References

1. B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, University of North Carolina, Chapel Hill, NC, 2011.
2. UmaMaheswari C. Devi and James H. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189, 2008.
3. Jeremy P. Erickson and James H. Anderson. Fair lateness scheduling: Reducing maximum lateness in G-EDF-like scheduling. In *ECRTS*, pages 3–12, 2012.
4. Jeremy P. Erickson, UmaMaheswari Devi, and Sanjoy K. Baruah. Improved tardiness bounds for global EDF. In *ECRTS*, pages 14–23, 2010.
5. Jeremy P. Erickson, Nan Guan, and Sanjoy K. Baruah. Tardiness bounds for global EDF with deadlines different from periods. In *OPODIS*, pages 286–301, 2010. Revised version at <http://cs.unc.edu/~jerickso/opodis2010-tardiness.pdf>.
6. Hennadiy Leontyev and James H. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*, 44(1):26–71, 2010.
7. Hennadiy Leontyev, Samarjit Chakraborty, and James H. Anderson. Multiprocessor extensions to real-time calculus. In *RTSS*, pages 410–421, 2009.
8. Włodzimierz Ogryczak and Arie Tamir. Minimizing the sum of the  $k$  largest functions in linear time. *Information Processing Letters*, 85(3):117–122, 2003.