An Approach to Task Attribute Assignment for Uniprocessor Systems

I. Bate and A. Burns
Real-Time Systems Research Group
Department of Computer Science
University of York
York, United Kingdom
e-mail: {ijb,burns}@cs.york.ac.uk

Abstract

The purpose of this paper is to investigate the issues related to task attribute assignment on an individual processor. The majority of papers on fixed priority scheduling make the assumption that tasks have their attributes (deadline, period, offset and priority) pre-assigned. This makes priority assignment trivial. However in practice, the system's timing requirements are specified and it is expected that the task attributes are synthesised from these. This paper is to present work that has been developed to solve this problem.

1 Introduction

A significant challenge is to derive task attributes for a fixed priority schedule that meet the system's timing requirements in a way that can be understood by a non-specialist. An approach is proposed for task attribute assignment that caters for all the likely timing requirements of complex control systems imposed on the scheduler.

The issue of task attribute assignment for fixed priority scheduled systems is a subject that has received comparatively little attention compared to other areas of real-time research. Notable papers on the issue include, Gerber [1] and Yerrabilli [2]. These techniques whilst powerful, are difficult to justify, therefore a more straightforward techniques would be of benefit. However, the main barrier to the adoption of their work is that the approaches assume all attributes are changeable. For example, the fact that intermediate tasks of a transaction may have other functions outside of the transaction is ignored. For example, changing a task's period could cause problems.

A secondary aim of the task attribute approach is to try and ensure that only one set of analysis (i.e. task schedulability analysis) verifies all the timing characteristics of the system. The benefit of the secondary aim is that a separate verification tool is not needed for the transactions' timing requirements.

There are a number of parts to the paper. Section 2 provides a description of the system's timing requirements that are to be met. Section 3 derives analysis for proving transactions are met based on the task set's attributes. Section 4 investigates how transaction requirements may be met in a uniprocessor systems in two cases. The two cases are; when the transaction deadline is less than or equal to the tasks' (that form the transaction) periods, and when transaction deadline is greater than the tasks' periods. Section 5 investigates how a task's jitter requirement can be met. Section 6 investigates how tasks' separation requirements can be met. Finally, section 7 combines the product of the earlier sections to provide an overall approach to task attribute assignment.

2 The System's Timing Requirements

An evaluation of industrial practice shows there are two principal categories of timing requirements to be considered: those associated with tasks, and those associated with transactions.

The timing requirements for a task are:

1. *Period* - All tasks can be considered to have a period. Event triggered tasks are modelled as a periodic task whose period is equal to the task's minimum inter-arrival time.

2. *Deadline* - The deadline of a task is the maximum time allowed from the expected task release until the completion of the task execution.

3. *Jitter* - The jitter constraint for a task is the allowed variation of task completion from precise periodicity. Jitter constraints are normally placed on the outputs from embedded systems to ensure the occurrence of actions does not vary too much.

4. *Separation* - A particular sequence of tasks may have a minimum time separation enforced to prevent a catastrophic effect.

A sequence of tasks executed in a fixed order is referred to as a transaction. The timing requirements for a transaction are:

1. *Period* - Similar to a task, a transaction has a periodic requirement. It is not unusual for some of the tasks in the transaction to be executed at different rates. In these cases, the transaction period is equal to the least common multiple of the tasks' (that form the transaction) period. The reason the least common multiple is chosen, rather than a lower value, is that the tasks can only execute in the required precedence order this often.

2. *End-to-End Deadline* - Transactions normally have a requirement that all tasks are executed in a particular order within a given amount of time.

3. *Jitter* - Similar to a task, a transaction may have a jitter constraint.

A need for any scheduling policy is to have an approach to task attribute assignment that effectively deals with these requirements.

3 Calculating the Response Times of Transactions

The scheduling model for the work contained in this paper is a fixed priority scheduled system where tasks have their priorities assigned using the Deadline Monotonic Priority Ordering (DMPO). DMPO is where tasks with the shortest deadline have the highest priority. The scheduling model allows both periodic and sporadic tasks, however some of the system's timing requirements may logically only apply to periodic tasks, e.g. separation requirements.

A prime driver for the task attribute assignment is to represent all the system's timing requirements as task attributes, i.e. if all the tasks' deadline are met then all the system's timing requirements are met. Independent of this aim, it is still useful to have analysis available that allows transactions' requirements to be verified. It has been shown that it is only necessary to check the one instance of the transaction immediately after the critical instant and only in the case where all tasks execute for their worst-case time [3].

Consider the timing requirements presented in Figure 1. Figure 1 illustrates a task set consisting of three tasks (A, B and C), and a single transaction requirement across all the tasks. The periods and deadlines of tasks A, B and C are 50, 100 and 50 respectively. The transaction has a period of 100 with a deadline of 75. Unless otherwise specified, all the tasks are initially given a deadline equal to their period.

There are two phases to the calculation of the transaction's response time, which are: establishing the particular

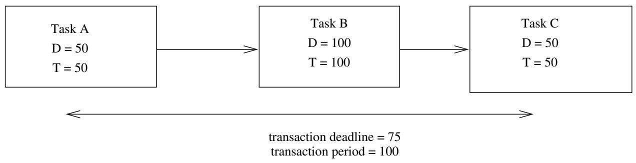

Figure 1: Timing Requirements for a Transaction

release of each task in the transaction, and the completion time for the releases of interest. This is carried out by starting with the first task and working through the tasks in the defined precedence order. Equation (1) can be used for calculating the task instance that is relevant to the transaction.

$$n_t = \left\lceil \frac{(n_{t-1} - 1)T_{t-1} + R_{t-1} - R_t + T_t}{T_t} \right\rceil \tag{1}$$

where t is the t^{th} task (assuming tasks are ordered according to precedence), in the transaction,
n_t is the instance of the t^{th} task, where $n_t \in \mathbb{N}$
n_1 is 1, and
R_t is the worst-case response time of task t.

The worst-case response time of task t, $R_t^{n_t}$, of the transaction's critical instant (i.e. time zero) is given in equation (2). When calculating transactions' response times, an assumption is made that the worst-case response time of a task is equal to the task's deadline. A proof of equations (1) and (2) is contained in [3].

$$R_t^{n_t} = (n_t - 1)T_t + R_t \tag{2}$$

where $R_t^{n_t}$ is the worst-case response time of the n_t^{th} instance of task t.

Using an initial value of $n_1 = 1$, the response time of each task in the transaction can be calculated by starting with the first task and taking each task in turn. Consider the example in Figure 1,

$$\begin{aligned} n_1 &= 1 && (3)\\ R_1^1 &= R_1 = 50 && (4)\\ n_2 &= \left\lceil \frac{R_1 - R_2 + T_2}{T_2} \right\rceil = 1 && (5)\\ R_2^1 &= 100 && (6)\\ n_3 &= \left\lceil \frac{R_2^1 - R_3 + T_3}{T_3} \right\rceil = 3 && (7)\\ R_3^2 &= 150 && (8) \end{aligned}$$

$\therefore$ The worst-case response time for the transaction is 150.

Analysis for verifying transactions can be developed using this approach. However, preference would be given to a scheduling technique that eliminates the need for bespoke analysis other than task schedulability analysis. The approach to meeting the timing requirements (and providing evidence that they are met) is illustrated in Figure 2. There are basically two parts to this approach; the first involves deriving the task attributes (without knowledge of the tasks' worst-case execution times), and the second then uses the tasks' worst-case execution times to verify that the timing requirements are met. This paper provides the first part of the approach (i.e. the task attribute assignment), the second part (i.e. the schedulability analysis) is covered in [3] as well as numerous other papers.

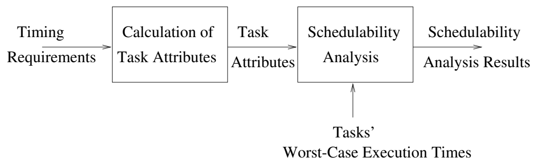

Figure 2: Diagram to Illustrate the Approach to Meeting the Timing Requirements

4 Meeting Transaction Deadlines in Uniprocessor Systems

The basic requirement for a transaction is that a sequence of tasks are executed in a specific order within a fixed amount of time. To achieve this requirement, the attributes that can be controlled are period, offset, deadline and priority. The attributes to be controlled can be further simplified by assigning priorities from deadlines using the DMPO.

The proposal is that the necessary timing requirements can be handled by just setting deadlines and offsets to appropriate values. The system timing requirements are then verified by simply proving the task's offsets are enforced in the scheduler and using schedulability analysis to show the task's deadlines are met.

The approach to meeting the transaction requirements is based on reducing the task deadlines in a systematic manner so that the task deadlines are reduced by the minimum possible. The technique is presented through a number of examples building up towards a general-purpose algorithm. An assumption is made that the sum of the worst-case execution times of the tasks is less than or equal to the transaction's deadline.

To demonstrate these approaches work, time-lines are produced as shown in Figure 3. The purpose of the time-lines is to illustrate the worst-case response times for tasks and transactions. The time-line depicts the worst-case situation for both tasks and transactions, which is a critical instant for the tasks and the tasks' response times are equal to their deadlines. The time-lines present the tasks' execution from the release of the first task of the transaction. The time-lines present enough instances of each task until the required precedence ordering has been achieved. In the case of the transaction requirement in Figure 1, Figure 3 shows task A executing followed by task B followed by task C. It is not deemed necessary to present instances of each task after the instance corresponding to the transaction, e.g. only one release of task A is shown.

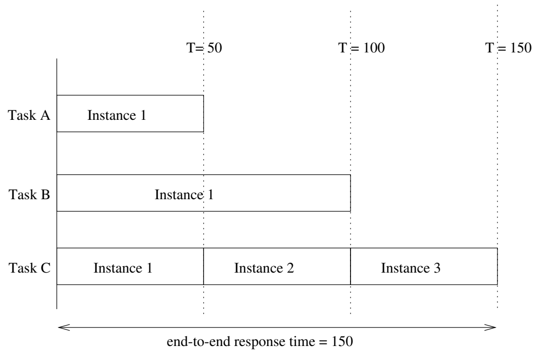

Figure 3: A Time-Line for the Transaction Illustrated in Figure 1

When producing the time-lines, which is effectively the same as analysing whether the transaction requirements are met, the following steps are taken:

1. Assume the first task in the transaction completes at its worst-case time, i.e. at its deadline.

2. Find the appropriate instance of the next (in terms of precedence) task in the transaction. An appropriate instance is one where the response time of the task is greater than the preceding task of the transaction.

   Rule: If a task has a lower priority than the preceding task and an identical (or later) release time, then the current release can be considered. Otherwise, the next release must be considered.

3. Assume the task completes at its worst-case time, i.e. at its deadline.

4. If the task is not the last of the transaction then return to step 2.

5. Test whether the response time of the transaction meets its deadline.

4.1 Task Attribute Assignment When the Transaction Deadline ≤ Transaction Period

The purpose of this section is to propose a strategy for assigning task attributes so that transaction requirements are met in the simplest of cases that is when the transactions' deadlines are less than or equal to their periods. The attribute to be controlled is the tasks' deadlines.

An important consideration when trying to meet the transaction's deadline is the relationship between tasks' deadlines. Consider two tasks *t-1* and *t*, task *t* is always released at the same time as task *t-1* and the tasks have deadlines such that $D_{t-1} < D_t$. Under these conditions, task *t* always follows task *t-1*, i.e. $n_t = n_{t-1}$ based on equation

(1). It is proven in [3] that these deadlines enforce the precedence relationship and that the transaction's response time is less than D_i - assuming the tasks meet their deadlines. The advantage of the deadlines conforming to this relationship is that the worst-case response time of each task in the transaction is kept to a minimum because the tasks are logically phased. Considering equation (2), the response time of the transaction is minimised if the value of n_i (where $i \in$ Tasks in the Transaction) is kept as small as possible. Therefore, the transaction deadline is more likely to be met. In addition, by dealing with deadlines in this way it is easier to justify the precedence ordering is met.

To demonstrate the approach to task attribute assignment, consider the timing requirement illustrated in Figure 1. The time-line in Figure 3 indicates the worst-case end-to-end response time for the task characteristics in Figure 1 is 150, which is outside our allowed band.

Tasks' deadlines are assigned starting with the last task in the transaction and working backwards towards the first task. By reducing the deadline of tasks A and B to 50-2Δ (where Δ = 1 clock cycle) and 50-Δ respectively, the worst-case response time of the transaction becomes 50 and the requirement is met. The time-line in Figure 4 demonstrates the requirement is met.

An important constraint is that deadlines can only ever be decreased because increasing the deadline could affect the ability to meet other requirements. Therefore, if the original deadline of task A was less than $50 - 2\Delta$, then task A's deadline would not be increased to $50 - 2\Delta$. However, the transaction would still be met.

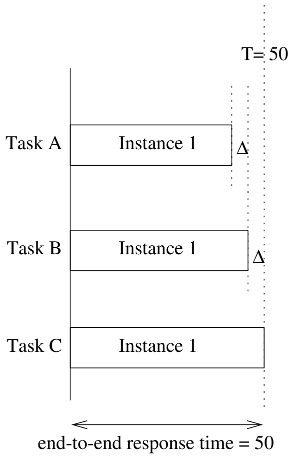

Figure 4: A Time-Line for the Transaction in Figure 1

It has been argued in [3] that the task assignment algorithm presented in this section is optimal in the case where the transaction deadline is less than the transaction period. However in our experience, the computational model considered in this section is too restrictive because a transaction's deadlines is often greater than their period. The following section is to relax this restriction.

4.2 Task Attribute Assignment For Arbitrary Transaction Deadlines

The aim of this section is to develop an approach to task attribute assignment for cases where transaction deadlines have an arbitrary value. Again, the aim is to use deadline assignment instead of priority assignment to avoid the need to separately analyse whether transaction requirements are met.

A transaction deadline could be dealt with using the approach in section 4.1. However, the ability to schedule the task set may be unnecessarily reduced by making the deadlines less than needed. The reason is the transaction deadline would be constrained so it has to be less than or equal to its period. A potential solution to this problem is to increase the deadline of the last task in the transaction to the value of the deadline of the transaction. Then, the task deadlines would be assigned so that precedence is achieved as described in section 4.1. The transaction's deadlines would then be verified using the schedulability analysis for arbitrary deadlines discussed in [4]. There are two problems with adopting this approach, which are; this form of timing analysis is pessimistic [5], and the tasks may have other constraints, e.g. other transactions, preventing the deadlines being increased. Therefore, a more appropriate approach is sought.

A characteristic of the requirements is a task's period could be longer than the period of the task that follows it in the transaction. In this case, it may not be necessary or possible to reduce all the deadlines so that perfect precedence between the tasks of the transaction is maintained. For example, consider the timing requirements in Figure 5. The requirements of the transaction are an iteration rate of 100 and an end-to-end deadline of 150. The periods of tasks A, B, C and D are 50, 100, 100 and 50 respectively. In this case, task B cannot always follow task A because task B has a slower update rate.

Figure 6 shows that without altering tasks' deadlines, the requirement is not met because the transaction's response time is 250. By the previous approach, in section 4.1, the transaction requirements could be met as shown in Figure 7. The task attributes illustrated in Figure 7 are sub-optimal because the timing requirements could be met with larger deadlines as shown in Figure 8. Having larger deadlines means the tasks have a lower priority. The benefit of this is that the impact of the change in deadlines on the rest of the system is minimised and the tasks are more likely to be schedulable.

The task attributes of Figure 7 can be generated using Algorithm 1. The algorithm does not simply rely on the tasks' deadlines being reduced so that perfect precedence is maintained. Instead, while the transaction requirement is not met, the highest deadline is reduced by Δ, and if

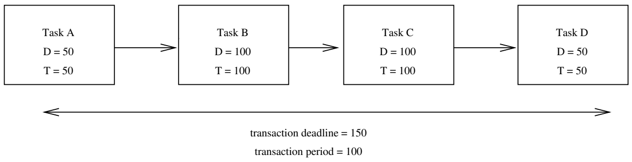

Figure 5: Timing Requirements for a Transaction

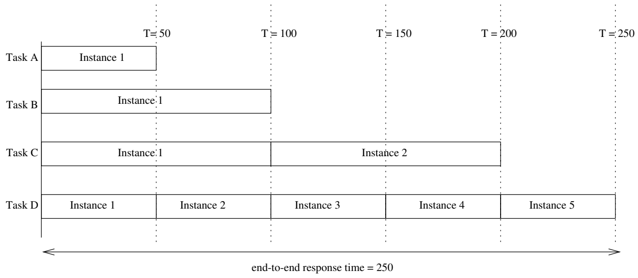

Figure 6: A Time Line for the Transaction in Figure 5

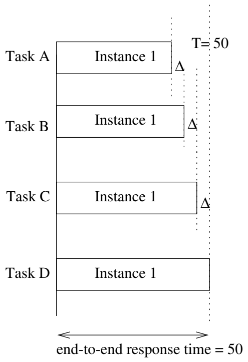

Figure 7: A Time Line for the Transaction in Figure 5

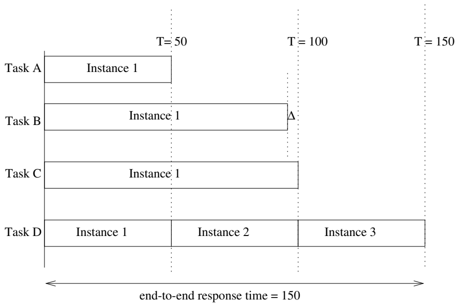

Figure 8: A Time Line For the Transaction in Figure 5

this causes any of the preceding sequence of tasks in the transaction to have the same deadline, then its deadline is also reduced by Δ. In the case of the task set in Figure 5, tasks B and C have their deadlines repeatedly reduced until the transaction's deadline requirement is met. The benefit of the approach in Algorithm 1 is that if the deadlines are larger, then there is a greater chance of a schedulable solution. The algorithm also contains a manipulation of the *TDAC - Task_Deadlines_Are_Changing* (that indicates whether the task deadlines have changed) flag. The use of this flag becomes apparent later in the paper.

Algorithm 1 - *Algorithm for the Generalised Approach*

for each task in the transaction
  if the following task has an equivalent deadline then
    reduce the task's deadline by Δ
    assign the value of false to the TDAC flag
  while the transaction deadline is not met
    assign the value of true to the TDAC flag
    take the longest deadline and reduce it by Δ
    for each task in the transaction
      if the following task has an equivalent deadline then
        reduce the task's deadline by Δ
        assign the value of false to the TDAC flag

4.3 Optimality of the Approach

With any approach it is important to ascertain whether it is optimal. If it is not optimal, then any deficiencies should be highlighted so that appropriate action can be taken where necessary. First impressions suggest that the approach may be optimal because the deadlines of the tasks are the maximum possible whilst still meeting the timing requirements. However, trying to prove optimality is difficult because of the effect of complex interactions with other tasks and transactions.

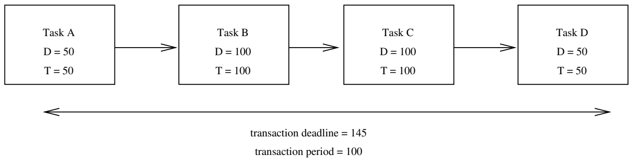

Figure 9: Timing Requirements For a Transaction

Consider the transaction requirement in Figure 9. Figure 10 presents the solution derived using the approach in Algorithm 1, whereas Figure 11 presents an alternative solution. It is impossible to judge which of these is the better solution without knowledge of all the tasks' execution times, priorities and deadlines. On the one hand, the task attributes in Figure 10 cause more interference through tasks B and C. While in Figure 11, tasks B and C cause less interference but task D causes more. Therefore, it can be stated that the approach in Algorithm 1 is not optimal.

In our experience using the task attribute assignment approach, no specific examples have been found where the issue raised in this section has caused problems. A case study on a real system is presented in [3] along with a discussion of what the designer could do if problems were encountered.

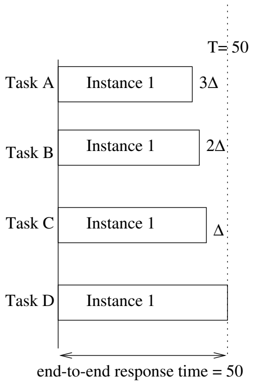

Figure 10: A Time Line for the Transaction in Figure 9

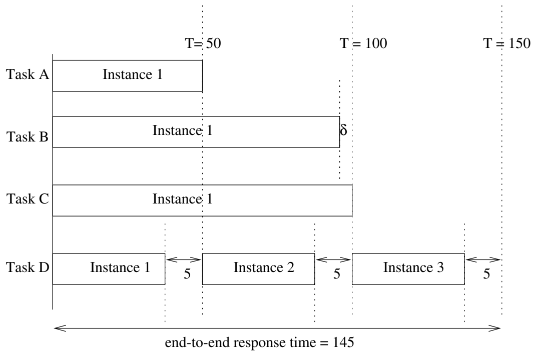

Figure 11: A Time Line for the Transaction in Figure 9

5 Jitter

The purpose of this section is to explore how jitter requirements can be handled by the manipulation of task attributes. An important constraint, and a frequent criticism of fixed priority scheduling is related to the control of jitter. A conventional model of fixed priority scheduling is said to suffer worse jitter than a cyclic scheduler [6].

A prime driver for this work is to control jitter (where it matters) and enable the schedulability analysis for tasks to also verify the jitter requirements. Therefore for similar reasons to those given in section 4.1, the jitter requirement should be handled using deadlines rather than priorities. If a task has a deadline equal to the jitter requirement, then the requirement is met because the window of allowed execution is constrained between the critical instant and the time allowed for the jitter requirement. Using this approach would be pessimistic since there is a minimum processing time for each task that adds no variability, i.e. the best case response time. Therefore, the deadline can be calculated using equation (9).

$$D_i = J_i + BCRT_i \tag{9}$$

where $BCRT_i$ is the best case response time of task i.

The best case response times of tasks can be generated in two stages. Initially, the value can be set to zero, this makes the task set harder to schedule since the deadlines are lower. Later, the value can be evolved as more results from analysis become available. The best case response times can be taken as the best case execution time or calculated using either exact analysis techniques, or the method proposed by Harbour, Garcia and Guiterrez [7].

Clearly it can be seen that having a deadline as defined in equation (9) constrains the variation of the task's computation time so that the jitter requirement is met. The fact that the choice of deadline controls the jitter means that the standard schedulability test also verifies the jitter requirement is met removing the need for extra analysis. However, there are problems with the approach based on equation (9). Two cases are illustrated.

Case 1

If many tasks have jitter requirements, then there may be an abnormally large number of tasks with short deadlines. Short deadlines means the tasks have to execute shortly after the critical instant, which could easily lead to an unschedulable solution.

Solution for Case 1

Phase the execution of tasks by spreading the execution using offsets. However, the use of offsets to phase execution should be avoided, where possible, due to the increased maintenance problems of having "slots". The maintenance problems of slots would be analogous to the problems of cyclic scheduling. □

Case 2

Another area of concern is related to transactions. A jitter requirement placed on the last task in the transaction would lead to all the preceding tasks having a shorter deadline, which could affect schedulability. For example, consider the task set illustrated in Figure 12, which represents a transaction requirement that includes a jitter requirement on the last task.

Using equation (9) to calculate the jitter requirement of task C results in a deadline of 6. Then applying the technique in Algorithm 1 leads to tasks A and B having a deadline of $6 - 2\Delta$ and $6 - \Delta$. The time-line in Figure 13 illustrates the execution of the tasks. The shaded area represents the allowed variation in task C's execution and the other time (1 unit) is the non-variant (equal to task C's best case response time) part of its execution. The problem with the solution in Figure 13 is that all the tasks have to execute within a tight deadline.

Solution for Case 2

The solution to the problem is again the use of offsets. It is proposed that the solution demonstrated in Figure 14 is often better than the solution demonstrated in Figure 13. The basis of the solution is to constrain the variation in task C's execution at the end of the allowed execution time of

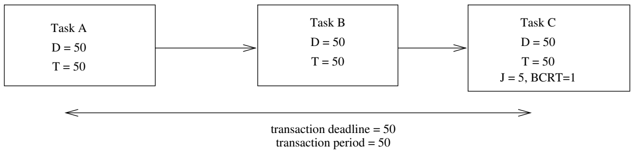

Figure 12: Timing Requirements For A Transaction

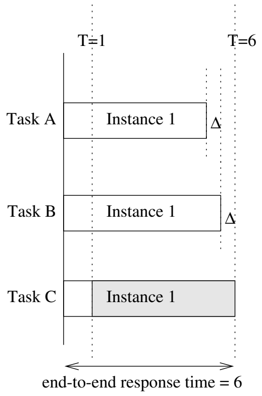

Figure 13: Time-line For the Transaction in Figure 12

the transaction by using an appropriate offset. The offset is calculated as shown in equation (10). The allowed execution time for task C is equal to the deadline calculated using equation (9). Precedence of the other tasks is enforced by making the other tasks execute before task C is released. Therefore, task B is given a deadline equal to the offset of task C and then Algorithm 1 is applied to the tasks preceding task C. The approach for choosing task B's deadline and task C's offset can be represented by Algorithm 2. □

$$\begin{aligned} O_C &= \text{transaction deadline} - \text{execution window of task C} \\ O_C &= \text{transaction deadline} - (J_C + BCRT_C) \end{aligned} \quad (10)$$

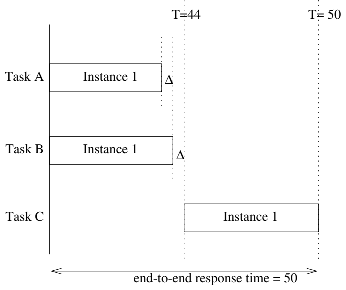

Figure 14: Time Line for the Transaction in Figure 12

It should be noted that if the transaction deadline is greater than its period, then the problem in Case 2 does not arise because Algorithm 1 does not unnecessarily enforce precedence in this case. Instead, the longest deadline is reduced until the requirements are met. Therefore, reducing task C's deadline to meet its jitter requirement does not necessarily lead to tasks A and B's deadlines being reduced to enforce the precedence.

Algorithm 2 - *Algorithm for Dealing With Jitter*

for each task (denoted i) in the system
 if task i has a jitter requirement then
  if task i is not the last task in the particular transaction then
   $D_i = J_i + BCRT_i$
  else
   if the transaction deadline $>$ *transaction period then*
    $D_i = J_i + BCRT_i$
   else
    if $D_i >$ *transaction deadline then*
     task deadline = transaction deadline
    $O_i = D_i - (J_i + BCRT_i)$
    for all preceding tasks (denoted j) in the transaction
     if transaction deadline is not met then
      if $D_j >$ *transaction deadline -* $(J_i + BCRT_i)$ *then*
       D_j *= transaction deadline -* $(J_i + BCRT_i)$

The approach defined in Algorithm 2 cannot be described as optimal. If a large number of tasks (specifically ones that are not part of the particular transaction) are already phased to execute during the time interval $[O_i, D_i]$, then the approach may lead to an unschedulable solution. In this circumstance an approach based entirely on equation (9) and Algorithm 1 may increase the likelihood of a schedulable solution. However, in practice this is unlikely to be the case. There is no optimal approach published for dealing with jitter.

6 Separation

The purpose of this section is to present a technique for assigning task attributes so that separation requirements are handled. For two tasks with a separation requirement (S_i) between them, the requirement can be satisfied by manipulating the offsets and deadlines of the two tasks. The technique gives the second of the two tasks an offset S_i from the deadline of the earlier task. This is expressed in equation (11).

$$O_i = S_i + D_{i-1} \quad (11)$$

where S_i is the separation requirement between two tasks,
task *i-1* precedes task i, and
D_{i-1} is the deadline of task *i-1*.

If the resultant offset is too great (i.e. $O_i + C_i > D_i$), then task *i* does not have a chance to meet its deadline. In this case, an appropriate metric for altering the deadline of the first task is required. An approach is presented in equation (12). The approach basically splits the available execution time between the two tasks. This is achieved by making the relative deadline (equal to the task's deadline minus its offset) of the preceding task equal to the relative deadline of the current task. The condition in equation (11) still has to be satisfied. The approach developed is represented in Algorithm 3. However, it should be noted that manual intervention may still be deemed necessary, dependence on the characteristics of the system.

$$D_{i-1} = \frac{D_i - S_i - O_{i-1}}{2} \tag{12}$$

Algorithm 3 - *Algorithm that Accounts for Separation*

for each task in the system
if a task (i) is to be separated from a task (i-1) then
$O_i = S_i + D_{i-1}$
if $(O_i + C_i) > D_i$ *then*
$D_{i-1} = \frac{D_i - S_i - O_{i-1}}{2}$
assign the value of true to the TDAC flag

7 Overall Task Attribute Assignment

Combining the various techniques discussed in this paper results in Algorithm 4. The overall method for priority assignment is not considered to be optimal because of some particularly obscure events that are discussed in this paper. For example, the case in section 4.3. However, use has shown that the approach is sufficient for the typical system timing requirements encountered.

Algorithm 4 - *Algorithm for Priority Assignment*

initialise TDAC flag to true
apply algorithm 2
while the TDAC flag is true
assign the value of false to the TDAC flag
for each transaction in the set of transactions
apply algorithm 1
apply algorithm 3
apply priorities to the tasks by DMPO
perform schedulability analysis of the task set

The only real problem encountered so far when using this approach has been caused by impractical requirements related to circular arguments. For instance consider the following two requirements, one requirement is task A has to precede task B and the other requirement is task B has to precede task A. Both requirements can not be resolved by Algorithm 1. In this case the Algorithm 1 would never obtain a result, with the deadlines tending towards $-\infty$. Clearly, there is no practical solution to this requirement, therefore it can be stated the requirement should not be allowed to exist. When implementing the algorithm, circularities should be recognised and flagged to the user so that appropriate manual intervention can take place.

8 Conclusions

This paper has addressed how attributes can be assigned to tasks that represent the system's timing requirements as priorities, offsets and deadlines. The three primary benefits of this work are:

1. the schedulability analysis assumes a critical instant verifies all the system's timing requirements,
2. the approach is easy to understand, and
3. the approach removes the need for complex mechanisms such as priority inheritance.

The approach derived has the significant advantage that the problems are dealt with off-line. This makes the implementation easier to produce and maintain, and it is easier to verify the precedence constraints are achieved. In most cases the approach is sufficient, except for a few obscure cases. An additional advantage is that our approach can be explained relatively straightforwardly to engineers and regulators, which helps the technology transfer process. Related to technology transfer, Rolls-Royce have successfully applied the technique to complex electronic engine controllers. The technique is subject to a pending patent [8].

References

[1] R. Gerber, S. Hong, and M. Sabsena, "Guaranteeing end-to-end timing constraints by calibrating intermediate processes," *IEEE Real-Time Systems Symposium*, 1994.

[2] R. Yerraballi, *Scalability in Real-Time Systems*. PhD thesis, Computer Science Department, Old Dominion University, August 1996.

[3] I. Bate, *Scheduling and Timing Analysis for Safety-Critical Systems*. PhD thesis, Department of Computer Science, University of York, November 1998.

[4] K. Tindell, *Holistic Scheduling Analysis for Distributed Hard Real-Time Systems*. No. YCS 197, Department of Computer Science, University of York, 1993.

[5] J. Gutierrez, J. Garcia, and M. Harbour, "On the schedulability analysis for distributed real-time systems," in *9th Euromicro Workshop on Real-Time Systems*, pp. 136–143, 1997.

[6] C. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives," *Real-Time Systems*, vol. 4, pp. 37–53, March 1992. Real-Time Syst. (Netherlands).

[7] J. Gutierrez, J. Garcia, and M. Harbour, "Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems," in *10th Euromicro Workshop on Real-Time Systems*, pp. 35–44, 1998.

[8] I. Bate and A. Burns, "Flexible scheduling for engine controllers," Tech. Rep. UK Patent Application Number 9710522.5 and US Patent Application Number 5,606,695, The Patent Office, May 1997.