Managing Complex Temporal Requirements in Real-Time Control Systems

Kristian Sandström and Christer Norström
Mälardalen Real-Time Research Center, Department of Computer Engineering
Mälardalen University, Västerås, Sweden
ksm@mdh.se

Abstract

Design and implementation of motion control applications includes the transition from control design to real-time system implementation. To make this transition smooth, the specification model for the real-time system should allow also for temporal requirements other than deadlines, e.g., deviation from nominal period time of an activity, end-to-end timing constraints, temporal correlation between different sampling tasks and constraints on temporal variations in output. Many real-time systems in industry today are based on pre-emptive priority based run-time systems, and hence, the temporal requirements should be fulfilled by correctly assigning attributes such as priorities and offsets to the tasks executing in such systems. Assigning priorities and offsets in order to fulfill complex temporal requirements originating from control design and computer system design is a hard task that should be supported by powerful methods and tools. In this paper we propose a method, which by assigning priorities and offsets to tasks guarantees that complex timing constraints can be met. In addition to the complex timing constraints, the method supports sporadic tasks, shared resources, and varying execution times of tasks. We present the idea and the implementation, which is based on a genetic algorithm, and illustrate the method by an example.

1. Introduction

To successfully design and implement motion control applications, such as robots, vehicle/trucks, and mobile machinery, in distributed computer systems there is a need to make a smooth and predictable transition from the design of a control system to its implementation in the computer system. One important prerequisite to accomplish this for real-time systems is to appropriately derive and model application timing requirements [1]. Moreover, these requirements must be translated into timing constraints that are suitable for implementation, thereby providing means for interaction between control and computer engineers. The timing constraints in the control design cannot be directly mapped to attributes of a real-time system, such as priorities, period times, deadlines and offsets of tasks. Assigning the attributes of the tasks so that the complex timing constraints derived from the control design are fulfilled is a non-trivial problem. Typical complex timing constraints are tolerances on sampling periods, end-to-end timing constraints, temporal correlation between different sampling tasks, and constraints on temporal variations in output.

The aim of this paper is to show how these complex timing constraints can be mapped to attributes of periodic tasks running on standard pre-emptive priority based multitasking real-time operating systems, as for example WxWorks provided by Windriver, in such a way that the timing constraints are fulfilled. In order to guarantee the behaviour of a control system subject to complex timing constraints, one must also consider that execution times of activities in most cases varies. Varying execution times will directly affect e.g., constraints on maximum deviation from a nominal period time.

Bate and Burns [2], propose a related method for assigning offsets and priorities to a fixed priority preemptive task set. They define a specification model that allows for expressing similar constraints as defined in Section 2 of this paper. However, their method does not consider the use of shared resources between sporadic and periodic tasks, which would require dealing with some

sort of semaphore mechanism. Furthermore, attribute assignment for dealing with constraints on period time variation is managed using a heuristic algorithm with local optimization that, according to the authors, can lead to unschedulable systems. For this reason it is difficult to extend the method to incorporate the additional constraints we would like to consider. Several researchers have approached the same problem by generating off-line schedules [3][4][5], and thereby do not support use of a standard priority-based RTOS. Furthermore, they do not support pre-emption, sporadic activities, and varying task execution times. Additionally, in [6] the authors presents a design methodology for real-time systems with end-to-end timing constraints, temporal correlation between different sampling tasks and constraints on temporal variations in output. The methodology derives period times, deadlines, and offsets for the tasks. However, the task model does not agree with a standard priority-based RTOS and constraints on period time variation cannot be expressed. The method assumes that task execution times are static.

The motivation for the work provided in this paper was mainly achieved from our participation in a real industrial project where we used a specification model with support for periodic tasks, deadlines, precedence relationships, mutual exclusions, and offsets [7]. By using this model we could express all timing constraints required by the application. However, the designer had to manually translate the timing constraints into attributes of the used model and consequently the designer acted as a pre-scheduler. This is possible for simple systems, but in systems with many such requirements it becomes very difficult to assign these attributes manually. Even if the designer succeeds in finding a feasible mapping, we get a maintenance problem [7].

In this work we use an enhanced specification model that support temporal dependencies between tasks, which relives the designer from the task of acting as a pre-scheduler. We will show that we can solve the problem of mapping a system described by this specification model to a run-time system model in an efficient way by using a genetic algorithm (GA). There are several reasons for using the GA approach. GA is a general optimisation method that has been used successfully for solving a wide variety of complex problems including scheduling, e.g., in [8][9][10][11]. It can also easily be extended to optimise on other attributes such as minimising the response time of handling an event. One of the most important properties of the GA is its ability to deliver a result that fulfils a subset of the timing constraints in cases where it is impossible to fulfil all constraints. This information is important since the designer then can get an indication of which constraints that can not be fulfilled and thereby simplify the re-modelling of the application. Although not all timing constraints are fulfilled, the application requirements may in some cases still be fulfilled since the robustness of the control design can tolerate deviations from the specification. However, this has to be verified by control analysis.

Thus, the contributions of this paper are:

- A specification model for describing systems with complex timing constraints.
- A synthesis algorithm that assigns priorities and offsets to tasks to fulfil the timing constraints given our specification model.
- An illustrative example.

The rest of this paper is organised as follow. Section 2 describes the used system model. The method for attribute assignment is covered in Section 3 followed by an example in Section 4. Finally we conclude the paper in Section 5.

2. System model

The system model is divided into two parts. The first part specifies the required behaviour of the run-time system and the second part is a definition of the specification model used to express the constraints of the task set.

2.1. Run-time system model

The basic model for the run-time system is a priority based, pre-emptive run-time system with shared resources protected by semaphores conforming to the priority ceiling protocol. Furthermore, the run-time system should provide a mechanism to enforce phasing between tasks i.e., offsets, and the ability to periodically release tasks with some predefined resolution, e.g., the operating system tick. These required features exist in many RTOS and if not, it is quite easy to construct these mechanisms from existing RTOS primitives.

The run-time system may also support prioritised sporadic activities.

2.2. Specification model

The specification model defines the information that has to be specified for each periodic and sporadic task, as well as the constraints that can be expressed on a task set. A periodic task is defined by its worst-case execution time, best-case execution time, and nominal period time. The nominal period time is the desired rate at which the task should be executed.

The following constraints can be expressed for and between periodic tasks:

- Period time variation – the maximum allowed deviation from the nominal period time. An upper (V_h) and lower bound (V_l) on the time between two consecutive executions of a task.
- Precedence – constraint specifying the execution order between two tasks
- Relative deadline – deadline relative the actual start time, i.e., the instant when the task execute its first instruction, of an instance of the task
- Correlation – constraint on the maximum time between executions of two or more parallel tasks
- Distance – constraint of a minimum distance between the executions of two tasks.
- Latency – constraint specifying a maximum allowed distance between the start of one task and the completion of another task.
- Shared recourses – specification of the tasks that uses the semaphores and times for the tasks critical sections.

Periodic tasks may have a varying execution time and phasing relative each other and hence the start time and completion time for a task can vary. This must be considered when finding an attribute assignment meeting the constraints for a task set. Therefore the analysis of a task set is performed for all instances over the least common multiple (*lcm)* of tasks period times. This is necessary since calculation of the worst- and best-case start time and completion time without considering all instances would be too pessimistic. As an example, consider the task t_1 and t_2 depicted in Figure 1. The tasks have a period time of T and only one constraint, a precedence constraint between t_1 and t_2. Assume a completion time of t_1 equal to 3, and a start time of t_2 equal to 4 in one period and a completion time of t_1 equal to 2, and a start time of t_2 equal to 2 in the next period. The latest completion of t_1 relative the period is 3 and the earliest start relative the period for t_2 is 2, i.e., the precedence is violated considering only the task timing while if the separate instances are considered one can see that precedence is achieved between t_1 and t_2.

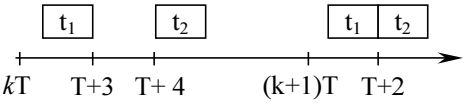

Figure 1. The execution of the two tasks t_1 and t_2.

Phasing of tasks and the start- and completion time variation is incorporated into the model by describing, for each instance of a task during the *lcm*, the earliest start time (*est*), the latest start time (*lst*), the earliest completion time (*ect*), and the latest completion time (*lct*). The constraints and notation are defined below, where τ_i represents task *i* and τ_i^n represents instance *n* of task *i*.

est(τ_i^n) - earliest start time of τ_i^n .

lst(τ_i^n) - latest start time of τ_i^n .

ect(τ_i^n) - earliest completion time of τ_i^n .

lct(τ_i^n) - latest completion time of τ_i^n .

Precedence $<\tau_i, \tau_j>$ holds iff
$lct(\tau_i^n) \le est(\tau_j^n)$

Distance $<distance, \tau_i, \tau_j>$ holds iff
$est(\tau_j^n) - lct(\tau_i^n) \ge distance$

Period time variation $<V_h, V_l, \tau_i>$ holds iff
$lst(\tau_i^{n+1}) - est(\tau_i^n) \le V_h \wedge est(\tau_i^{n+1}) - lst(\tau_i^n) \ge V_l$

Relative deadline $<relativeDeadline, \tau_i>$ holds iff
$lct(\tau_i^n) - est(\tau_i^n) \le relativeDeadline$

Latency $<latency, \tau_i, \tau_j>$ holds iff

$$T_i = T_j \rightarrow \forall n(lct(\tau_i^n) \le est(\tau_j^n) \wedge lct(\tau_j^n) - est(\tau_i^n) \le latency)$$
$$T_i > T_j \rightarrow \forall n \exists m : lct(\tau_i^n) \le est(\tau_j^m) \wedge lct(\tau_j^m) - est(\tau_i^n) \le latency)$$
$$T_i < T_j \rightarrow \forall n \exists m : lct(\tau_i^m) \le est(\tau_j^n) \wedge lct(\tau_j^n) - est(\tau_i^m) \le latency$$

Correlation $<Correlation, \tau_i, \tau_{i+1}, \ldots, \tau_{i+m}>$ holds iff
$\forall j,k \in i..(i+m) : \left| lst(\tau_j^n) - est(\tau_k^n) \right| \le Correlation$

A sporadic task is specified by a worst-case execution time, a minimum inter-arrival time, and a deadline. Here, the best-case execution time is not considered, since the best case considering the entire task set is that the sporadic task is not activated at all at a given instance. The minimum inter-arrival time specifies the shortest possible time between two consecutive activations of the task and the deadline is relative the activation of the task. It is also possible for sporadic tasks to use semaphores that is shared with both sporadic and periodic tasks

3. Attribute Assignment

This section describes the algorithm for assigning priorities and offsets to the periodic tasks and priorities to sporadic tasks in order to meet the constraints specified for a task set. It is assumed that constraints are specified according to the model defined in the previous section. The heart of the attribute assignment is a genetic algorithm that assigns offsets and priorities, evaluates the

assignments, and incrementally finds new assignments, thereby gradually achieving the required system behaviour. The general idea of a GA is to let individuals in a population gradually improve by the mechanisms of natural selection. In this case the individuals consists of attribute assignments for a tasks set and the environment to master is the constraints put on that task set. An overview of the structure and operation of the genetic algorithm used is given below.

1. Initial *Population* – The algorithm initially makes a number of guesses about the assignment of priorities and offsets for the complete task set. A complete assignment for the entire task set is referred to as a *genome*.
2. Apply *Objective function* – The objective function calculates a goodness value for each genome, given how far the genome is from meeting the requirements. If the objective is reached, the algorithm has found a solution and is terminated.
3. Crossover – In this step parts of different genomes are combined to produce an offspring, i.e., a new genome built from two other genomes.
4. Mutation – Randomly alters a genome by e.g., by reassigning a priority in the genome by a random number.
5. Repeat from step 2, each iteration is referred to as a *generation*

An assignment of offsets and priorities for a task set is represented by a set of offset priority pairs for the periodic tasks and a priority for each sporadic task, e.g., a task set with periodic tasks t_1 to t_i and sporadic tasks st_1 to st_j is represented by the set g: {<$priority_1$, $offset_1$>,…,<$priority_i$, $offset_i$>,<$priority_1$>,…,<$priority_j$>}. The population of the genetic algorithm then consist of a number of such priority-offset sets $G = \{g_1, \ldots, g_n\}$.

The objective function calculates start times and completion times for the task set and derive a single value used for sorting different genomes by their closeness to the optimum, where the representation of optimum is defined by the genetic algorithm, e.g., the higher value the closer to optimal. The deviations from the requirements for a task set, using the offsets and priorities of a given genome, are calculated by rearranging the formulas earlier described in Section 2. For example, deviation from the distance constraint is calculated by reformulating $lct(\tau_j^n) - est(\tau_i^n) \geq dist$ as $dist - (lct(\tau_j^n) - est(\tau_i^n))$. The objective value is then expressed as a percentage of the allowed deviation, e.g., $dist - (lct(\tau_j^n) - est(\tau_i^n)) \;/\; dist$. This value is then divided by the number of instances of the task during an *lcm*. The division by *dist* and the number of instances is done in order to normalise the value against other constraints so that not too strong emphasis is put on some constraints. The objective value for a genome is the sum of the normalised values calculated for each constraint.

The analysis performed to calculate the earliest and latest start times and completion times for the instances of the task set can be divided in to two cases: 1) The earliest start time and completion time is calculated disregarding the sporadic tasks, using the best-case execution times, and assuming that no tasks are blocked when using shared resources. 2) The latest start time and completion time is calculated considering interference from sporadic tasks, using the worst-case execution time and assuming blocking. Common for both cases is that the execution of the instances of the periodic tasks is analysed by time wise stepping through the *lcm* of the periodic tasks. At each relevant point in time it is determined which task instance that should execute based on priority and offset information. By relevant points in time we mean the operating system tick and the completion of task execution. The analysis is safe but not always exact, this is the case when e.g., the earliest start time of one task instance can not occur when another task instance is experiencing its latest completion time. By necessity the analysis is a compromise between precision and calculation speed. Since an increased complexity of the calculation will decrease the number of calculations performed during a given amount of time, it will also decrease the number of generated and evaluated genomes of the GA, thereby possibly leading to less good assignments. The behaviour resulting from a given set of priority and offset pairs can be fully explored after delivery from the GA.

4. Example

In this example we assume an application for which constraints needed for implementation have been derived from a control design and expressed according to the specification model described in Section 2. The computer control system consists of 12 periodic tasks and 3 sporadic tasks resulting in a total CPU-utilisation of 42 %. Furthermore, the periodic tasks have 31 temporal constraints that should be fulfilled by assigning appropriate priorities and offsets to the tasks. The period times of the periodic tasks result in 54 instances, during an *lcm*, that have to be considered by the analysis. The example is intended to give the reader a glimpse of the non-trivial task of assigning priorities and offsets to task sets with temporal dependencies between tasks, even for small examples.

In Table 1 the periodic tasks are listed together with constraints on period time variation and relative deadline. All specified times are given in microseconds.

Task	Wcet	Bcet	Period time	V_l	V_h	Relative Deadline
S1	300	250	25000	24000	26000	1500
S2	300	250	25000	24000	26000	1500
S3	300	250	25000	24000	26000	1500
A1	700	600	25000	23000	27000	1500
A2	700	600	25000	23000	27000	1500
A3	700	600	25000	23000	27000	1500
D1	1400	900	50000	45000	55000	6500
D2	2100	2100	50000	45000	55000	5000
T1	370	370	10000	8000	11000	2000
T2	120	120	25000	24000	26000	400
DT1	300	300	10000	7000	14000	5000
DT2	800	800	50000	45000	53000	2000

Table 1: The example task set with constraints.

Figure 2 depicts additional constraints on some of the periodic task. The constraints include latency constraints both between task with the same period time and between tasks with different period times, correlation constraints, and distance constraints. Furthermore, tasks S1, S2, S3, and DT1 share a resource protected by a semaphore, where the semaphore is locked for times in the range of 20 to 90 microseconds.

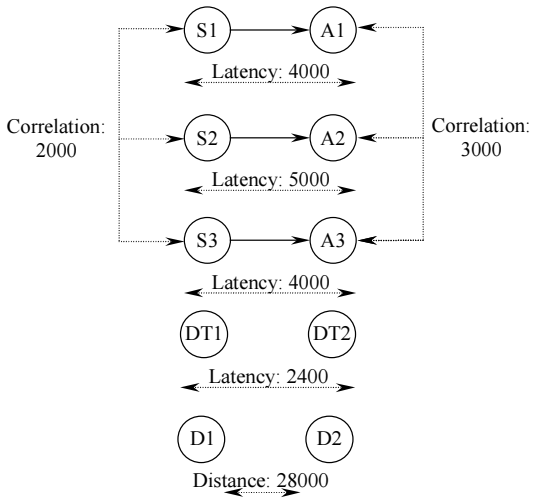

Figure 2. Additional constraints on the periodic tasks.

Table 2 show the specification of the three sporadic tasks with minimal inter-arrival time, worst-case execution time and deadline relative activation for each task.

Task	Min Inter-arrival Time	Wcet	Deadline
SP1	1000	45	500
SP2	3000	150	2700
SP3	50000	2000	35000

Table 2. The sporadic tasks of the example.

Given the specification above, the GA tries to find an attribute assignment in the run-time model such that the execution of the task set fulfils the constraints. Table 3 illustrates the progress of priority and offset assignment with regard to the latency constraint between task S1 and A1, the correlation between S1, S2, and S3, and the deadline of sporadic task SP1. The numbers in the table reflect approximate values, for the constraints, resulting from the assignment of priorities and offsets for generations 10 to 50 at intervals of 10, the second row display the specified constraints. The assignment of priorities and offsets are listed for each generation in table 4, at generation 50 all constraints are met. The total calculation time was approximately 7 seconds running on a 550 MHz Pentium.

	Latency S1,A1	Correlation S1-S3	Deadline SP1
Constraint	4000	2000	500
Generation 10	6800	3200	2950
Generation 20	2300	2150	7000
Generation 30	2045	1200	345
Generation 40	2045	1200	345
Generation 50	2045	1200	345

Table 3. The progress of the GA for generation 10 to 50.

	Periodic Task : Offset : Priority Sporadic Task : Priority		
Generation 10	S1:6000:15 A1:12000:14 D1:13000:13 T2:19000:2 SP1:11	S2:9000:3 A2:12000:14 D2:47000:5 DT1:7000:8 SP2:14	S3:6000:8 A3:8000:14 T1:7000:12 DT2:47000:8 SP3: 0
Generation 20	S1:6000:15 A1:8000:14 D1:13000:13 T2:9000:10 SP1:2	S2:8000:14 A2:8000:3 D2:47000:8 DT1:4000:0 SP2:11	S3:7000:11 A3:8000:9 T1:1000:3 DT2:4000:5 SP3:2
Generation 30	S1:6000:15 A1:8000:14 D1:13000:13 T2:9000:10 SP1:15	S2:5000:10 A2:8000:3 D2:47000:8 DT1:7000:5 SP2:13	S3:5000:8 A3:8000:9 T1:1000:3 DT2:29000:7 SP3:0
Generation 40	S1:6000:15 A1:8000:14 D1:4000:8 T2:19000:2 SP1:15	S2:5000:10 A2:8000:3 D2:47000:5 DT1:7000:12 SP2:13	S3:5000:8 A3:8000:14 T1:7000:8 DT2:47000:8 SP3:0
Generation 50	S1:6000:15 A1:8000:14 D1:4000:8 T2:7000:5 SP1:15	S2:5000:10 A2:8000:3 D2:47000:5 DT1:9000:10 SP2:13	S3:5000:8 A3:7000:12 T1:1000:3 DT2:29000:7 SP3:0

Table 4. The attributes for the tasks for each generation.

5. Conclusion

The problem of assigning priorities and offsets to tasks from a specification model supporting complex timing constraints is an important part in the implementation of real-time systems that consist of a number of periodic control activities executing with different frequencies while exchanging data. Such multirate control systems are for instance common in motion control algorithms. Sampled data control applications, in general, are real-time systems that are sensitive to deviations from nominal deterministic timing, i.e. the timing that normally is assumed in control design. Since an implementation of a computer control system inevitably introduces time-delays and time-variations, it is important to investigate the sensitivity of a control system to such "timing disturbances" during the control-engineering phase. Moreover, timing tolerances together with other timing requirements must be clearly communicated from the design phase (presumably carried out by control engineers) to the implementation phase (presumably carried out by computer engineers). Actual time-variations and delays should be fed back to the control design [12]. Many motion control applications are used in safety critical contexts, and/or environments where high reliability and availability are required. This emphasises the need for analysis of the correctness of the computer control system prior to implementation.

In earlier work we have had experiences of an industrial project where a more conventional specification model were used, supporting deadlines, precedence relationships, mutual exclusions, and offsets. Using this model the engineers had the complex and time consuming task of manual translation of constraints such as period time variation and multirate latency into attributes of the used model. Although this manual assignment of attributes is possible for smaller systems, we get a maintenance problem when introducing changes in the design. This problem can be handled by the introduction of more powerful means of expressing timing requirements for real-time control systems and by providing automated tools that help the engineer in assigning priorities and offsets.

In this paper we propose a method for fulfilling complex temporal requirements by assigning priorities and offsets to tasks running on a standard commercial RTOS using a genetic algorithm. Moreover, The use of a genetic algorithm brings with it the additional benefit to find near to optimal solutions when no solution exists that fulfils all the requirements. This is important from an engineering perspective, since the result can be used as input for remodelling of the application. The somewhat non-deterministic behaviour of genetic algorithms may be considered a drawback. However, the algorithm has been tested with good result on a substantial number of task sets [13].

References

[1] Törngren M. Fundamentals of implementing real-time control applications in Distributed computer systems. J. Of Real-Time Systems, 14, 219-250, Kluwer Academic Publishers, 1998.

[2] Bate I. and Burns A. An Approach to Task Attribute Assignment for Uniprocessor Systems. In Proc. 11th Euromicro Conference on Real-Time Systems (ECRTS99), York, England, June 9-11, 1999, IEEE Computer Society.

[3] Mok A. K., Tsou D., and De Rooij R. C. M. The MSP.RTL Real-Time Scheduler Synthesis Tool. In Proc. 17th IEEE Real-Time Systems Symposium, pp. 118-128. IEEE Computer Society.

[4] Würtz J. and Schild K. Scheduling of Time-Triggered Real-Time Systems, In Constraints, pp. 335-357, Oct, 2000. Kluwer Academic Publishers.

[5] Cheng S. T. and Agrawala A. K. Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints. Second International Workshop on Real-Time Computing Systems and Applications (RTCSA'95) October 25-27, 1995

[6] Gerber R., Saksena M, and Hong S. Guaranteeing Real-Time Requirements with Resource-Based Calibration of Periodic Processes. IEEE Transactions on Software Engineering, 21(7), July 1995.

[7] Norström C., Gustafsson M, Sandström K., Mäki-Turja J, Bånkestad N. Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry, In Proc. *Eigth IEEE International Conference and Workshop on the Engineering of Compute-Based Systems* Washington, US , April 2001. IEEE Computer Society

[8] Zomaya A. Y., Ward C., and Macey B. Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues. In IEEE Transaction on Parallel and Distributed Systems, VOL. 10, NO 8, August 1999.

[9] Corrêa R. C., Ferreira A., and Rebreyend P. Scheduling Multiprocessor Tasks with Genetic Algorithms. In IEEE Transactions on Parallel and Distributed systems, VOL 10, NO. 8, August 1999.

[10] Grajcar M. Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System. In Proc. 36th Design Automation Conference (DAC), p. 280-285, New Orleans, 1999. ACM Press.

[11] Faucou S., Déplanche A., and Beauvais J. Heuristic Techniques for Allocating and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems. In 3rd IEEE International Workshop on Factory Communication Systems, September 6, 2000. IEEE Industrial Electronics Society.

[12] Törngren M. Modelling and Design of Distributed Real-time Control Applications. PhD thesis, Dept. of Machine Design, The Royal Institute of Technology, Stockholm, Sweden, 1995.

[13] Sandström K. Evaluation of a Genetic Algorithm for RTOS Attribute Assignment. Technical Report, Mälardalen University, MRTC, October 2001.