Mälardalen University Licentiate Thesis
No.11

Transformation Methods for Off-line Schedules to Attributes for Fixed Priority Scheduling

Radu Dobrin

May 2003

MÄLARDALEN UNIVERSITY

Department of Computer Science and Engineering
Mälardalen University
Västerås, Sweden

ISBN 91–88834–01–8
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

[1]Paper A, B and C, ©by publisher.

Abstract

Off-line scheduling and fixed priority scheduling (FPS) are often considered as complementing and incompatible paradigms. A number of industrial applications demand temporal properties (predictability, jitter constraints, end-to-end deadlines, etc.) that are typically achieved by using off-line scheduling. The rigid off-line scheduling schemes used, however, do not provide for flexibility. FPS has been widely studied and used in a number of industrial applications, e.g., CAN bus, mostly due to its simple run-time scheduling and small overhead. It can provide more flexibility, but is limited with respect to predictability, as actual start and completion times of executions depend on run-time events.

In this work we show how off-line scheduling and FPS run-time scheduling can be combined to get the advantages of both – the capability to cope with complex timing constraints while providing run-time flexibility. The proposed approaches assume that a schedule for a set of tasks with complex constraints has been constructed off-line. We present methods to analyze the off-line schedule and derive FPS attributes, e.g., priorities, offsets, and periods, such that the runtime FPS execution matches the off-line schedule. The basic idea is to analyze the schedule and to derive task attributes for fixed priority scheduling. In some cases, i.e., when the off-line schedule can not be expressed directly by FPS, we split tasks into instances to obtain a new task set with consistent task attributes. Furthermore, we provide a method to keep the number of newly generated artifact tasks minimal.

Finally, we apply the proposed method to schedule messages with complex constraints on Controller Area Network (CAN). We analyze an off-line schedule constructed to solve complex constraints for messages, e.g., precedence, jitter or end-to-end deadlines, and we derive attributes, i.e., message identifiers, required by CAN's native protocol. At run time, the messages are transmitted and received within time intervals such that the original constraints are fulfilled.

Acknowledgements

I would like to thank my supervisor Gerhard Fohler at the Department of Computer Science and Engineering for his guidance and constant constructive feedback through the first part of my graduate education. Further more, I want to thank Damir Isovic and Tomas Lennvall for their help with reviewing the work presented in this thesis, and all my other colleagues at the Department of Computer Engineering for the pleasant time we have had during this period of time. Last, but definitely not least, I would like to thank my parents for their major contribution in my scholar education.

This work has been supported by ARTES.

Västerås, May 2003
Radu Dobrin

Contents

List of Figures

List of Publications

The following articles are included in this licentiate[2] thesis:

A *Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-line Schedules*, Radu Dobrin and Gerhard Fohler, In Proceedings of Conference on Real-Time Computer Systems and Applications, Korea, 2000.

B *Translating Off-line Schedules into Task Attributes for Fixed Priority Scheduling*, Radu Dobrin, Gerhard Fohler and Peter Puschner, In Proceedings of Real-Time Systems Symposium, London, UK, 2001.

C *Implementing Off-line Message Scheduling on Controller Area Network (CAN)*, Radu Dobrin and Gerhard Fohler, In Proceedings of Emerging Technologies and Factory Automation, Antibes, France, 2001.

[2]A licentiate degree is a Swedish graduate degree halfway between MSc and PhD.

Chapter 1

Introduction

1.1 Real-Time Systems

Real-time systems are computer systems in which the correctness of the system depends not only on the logical correctness of the computations performed, but also on which point in time the results are provided [1]. Delivering a result at a point in time beyond the latest possible, i.e., after it's deadline, may result to catastrophic consequences in *hard real-time systems*. Example of such systems are medical control equipment or vehicle control systems. On the other hand, in *soft real-time systems*, e.g., multimedia applications, a number of deadlines can be missed without serious consequences. In this work we will primarily focus on hard real-time systems.

A real-time system typically consist of a number of *resources* (e.g., one or several processors), a number of *tasks*, designed to fulfill a number of *timing constraints*, and a *scheduler* that assigns each task a fraction of the processor(s), according to a *scheduling policy*. Tasks are usually *periodic* or *non-periodic*. Periodic tasks consist of an infinite sequence of invocations, called *instances* or *jobs*. Non-periodic tasks are invoked by the occurrence of an event. The choice of tasks and scheduling policy is made to satisfy some original constraints imposed on the system. Tasks can have various parameters, such as period, deadline, priority, depending on the scheduling policy chosen to be used.

The scheduling policies are divided in *off-line*, [2, 3], and *on-line scheduling*, [4]. The main difference between the two is that, in off-line scheduling, the decision of which tasks to execute at which time point and on which processor

is made at the design stage, and, at run-time, the dispatcher selects which task to execute from scheduling tables. On the other hand, in on-line scheduling, all decisions are made at run-time depending on the task priorities and their arrival times. At each point in time, the task which is ready to execute and has the highest priority, is dispatched to execute. On-line scheduling is furthermore divided in *fixed*- and *dynamic-priority scheduling*, e.g., *rate monotonic* (RM) or *earliest deadline first* (EDF) [4].
A key issue in real-time systems is *predictability*, i.e., to be able to anticipate the behavior of the system *before* run-time and the guarantee that the system will behave as anticipated *at* run-time. At the same time, *run-time flexibility* is a desired feature, as not all run-time events can be completely accounted for in advance. Additionally, the choice of scheduling strategy in real-time systems is strongly related to the nature of the timing constraints which are to be fulfilled. As different scheduling schemes provide different levels of, e.g., predictability or flexibility, for the cost of a number of limitations, there is usually a trade-off between the ability to handle complex constraints and the level of flexibility provided by the selected scheduling strategy.
In this work, we present mechanisms to handle real-time, complex constraints, while providing predictability and run-time flexibility for the task executions. In particular, we want to handle complex constraints while exploiting the run-time advantages provided by FPS.

1.2 Complex Constraints

In this section we describe a number of constraints that are challenging to deal with in priority driven scheduling, e.g., FPS, while fairly easy to solve in off-line scheduling.
Jitter – The time interval between consecutive task executions is bounded by fixed values. In this case, the execution of consecutive instances of the same task has to fixed between pre-determined points in time. Consequently, in some cases, different instances of the same tasks must have different attributes, e.g., priorities, leading to inconsistencies in FPS.
Precedence –Tasks must execute in a pre-defined order, e.g., sampling and actuating tasks in real-time control systems. Work has been done to deal with precedence constraints in EDF [5] as well as precedence relations have been taken into account when performing the schedulability analysis in FPS [6]. However, the issue of attribute assignment for FPS is a challenging task as the task executions depend on the unpredictable run-time events.

Distribution –Tasks are allocated to different nodes, e.g., to achieve inter-node communication. Well used in, e.g., automotive and avionics industry, [7], together with FPS. Additionally, in some cases, tasks with precedence constraints have to be alocated to different nodes. Mapping of all these constraints to FPS attributes directly, may be a challenging task.

Instance separation –Usually demanded in control systems [8], e.g., to achieve synchronized sampling, control computations and actuating [9]. That may require the execution of different instances of the same task separated by different time intervals, potentially leading to FPS attribute inconsistencies.

1.3 Motivation

1.3.1 Off-line vs. Fixed Priority Scheduling (FPS)

Off-line scheduling, [3, 2], and fixed priority scheduling (FPS), [10, 11], are often considered as having incompatible paradigms, but complementing properties. FPS has been widely studied and used in a number of applications, mostly due its simple run-time scheduling, small overhead, and good flexibility for tasks with incompletely known attributes. Temporal analysis of FPS algorithms focuses on providing guarantees that all task instances will finish before their deadlines. However, additional constraints to FPS schemes require new schedulability tests, which may not have been developed yet, or may find the system unschedulable in the new configuration. Hence, the run-time flexibility provided by FPS comes at the expense of ability to handle multiple constraints, such as, jitter, instance separation or end-to-end deadlines.

Furthermore, FPS is widely used in a number of industrial applications involving network scheduling using Controller Area Network (CAN). Early results on message scheduling on CAN have been presented in [12] and [13], in which the authors focused on fixed priority scheduling based on work presented in [4] and [14]. An approach to time–triggered communication on controller area network has been presented in [15], while in [16], the authors presented an approach to enhance both event– and time–triggered communication in CAN. However, both approaches imply modifications to the native CAN protocol.

Priority assignment for FPS tasks has, for example, been studied in [10] and [11]. [17] studies the derivation of task attributes to meet overall constraints, e.g., demanded by control performance. Modifications to the basic scheme to handle semaphores [18], aperiodic tasks [19], static [20] and dynamic [21] offsets, and precedence constraints [22], have been presented.

Off-line, table driven, scheduling for time-triggered systems, [3], on the other hand, provides predictability, as all times for task executions are determined and known in advance. The off-line scheduler allocates tasks to the processors and resolves complex constraints by determining windows for tasks to execute in, and sequences, usually stored in scheduling tables. At run-time, the dispatcher, invoked at regular time intervals, selects which task to execute from the scheduling tables, ensuring tasks execution within the off-line determined windows and, thus, meet their constraints.
However, as all actions have to be planned before startup, run-time flexibility is lacking. The advantage of solving complex constraints comes at a price of limited run-time flexibility in terms of ability to handle tasks with incompletely known attributes, e.g., aperiodic or sporadic tasks.
Off-line scheduling for time triggered systems has, for example, been studied in [3, 2]. The choice of scheduling technique used in order to achieve different requirements has been well analyzed and discussed [23].
The purpose of this work is to provide methods to combine the advantages of both scheduling strategies, off-line scheduling and FPS, i.e., predictability and ability to solve complex constraints while flexibility at run-time. A method to transform off-line schedules into earliest deadline first tasks has been presented in [24].

1.3.2 Application domains

Instead of enhancing only either FPS or off-line scheduling alone, the methods provide for a combination, such that benefits of either scheme are accessible to the other. In particular, the presented methods provide solutions for the following scenarios:

Legacy Systems: Some safety critical applications, e.g., from the avionics domain [7], demand temporal partitioning of task executions or assertions not only about deadline being met, but restrictions on the actual times when task executions are performed. Typically, such applications are executed in time triggered architectures with off-line schedule construction. A move to fixed priority based systems has to ensure the specific demands will be met, which may be cumbersome applying standard FPS methods, as these concentrate on deadlines primarily.
The proposed methods transform these demands directly into attributes for tasks to be feasibly scheduled by FPS, pertaining the predictability provided by off-line schemes.

FPS systems with unresolved constraints: The feasibility test provided for FPS tasks defines the types of constraints which can be met. Additional constraints or combinations require modifications to existing tests or the development of new ones, which may not be available in limited time.
In addition, constraints demanded by complex applications, cannot be expressed generally. Control applications may require constraints on individual instances rather than fixed periods, reliability demands can enforce allocation and separation patterns, or engineering practice may require relations between system activities.
The proposed method resolves the need for developing of new specific tests for unusual constraints: first, a designer will apply known tests on the application under consideration. Should standard schemes prove to be incapable, the designer submits these tasks to an off-line scheduling scheme, which can use elaborate and general methods, such as search or constraint satisfaction, to provide a feasible off-line schedule. Then, by using the proposed methods, we derive attributes, such as period, priority, and offset for these tasks, such that they can be scheduled with FPS, while meeting the specific constraints.

Predictable flexibility: Off-line scheduling provides deterministic execution patterns for all tasks in the system, while FPS schemes provide flexibility for all tasks. Only few applications, however, will demand either determinism or flexibility uniformly for all activities in the system. Rather, only few selected tasks have tight restrictions on their executions, e.g., those sampling or actuating in a control system, with strict demands on jitter and variability, while a majority can execute flexibly.
The methods allows the amount of flexibility at runtime to be set off-line in a predictable way by including restrictions on task execution as input to the transformation algorithm.

Off-line scheduling in Controller Area Network (CAN): Controller Area Network (CAN), has gained wide acceptance as a standard in a large number of industrial applications. The priority based message scheduling used in CAN has a number of advantages, some of the most important being the efficient bandwidth utilization, flexibility, simple implementation and small overhead. However, the increasing demands from the industrial applications leads to increased complexity imposed on the system.
On the other hand, off-line scheduling for time triggered systems provides determinism [3, 2], and, additionally, complex constraints can be solved off-line, but this scheduling strategy is not suitable for native CAN protocol.

By using the proposed methods, we transform off-line scheduled transmission schemes into sets of messages that can be feasibly scheduled on CAN without modifying the basic CAN mechanism.

1.4 Problem Formulation and Proposed Solutions

In this thesis we present work to combine FPS with off-line schedule construction.

Problem formulation First, an off-line schedule is constructed for a set of tasks to fulfill their complex constraints. Then, by analyzing the off-line schedule together with the original constraints, we derive FPS attributes, i.e., priorities, offsets, deadlines, such that the tasks, when scheduled by FPS, will execute flexibly, while fulfilling the same complex constraints of the original off-line scheduled tasks.

Proposed solutions FPS cannot reconstruct all schedules with periodic tasks with the same priorities for all instances directly. The constraints expressed via the off-line schedule may require that instances of a given set of tasks need to be executed in different order on different occasions. Hence, there not allways exist a valid FPS priority assignment that can achieve these different orders. Our methods detects such situations, and circumvents the problem by splitting a task into its instances. Then, the algorithm assigns different priorities to the newly generated “artifact” tasks, the former instances.
Key issues in resolving the priority conflicts are the number of artifact tasks created, and the number of priority levels. Depending on how the priority conflict is resolved, the number of resulting tasks may vary, e.g., splitting a task with a large number of instances over LCM would result in a large number of artifacts. The method presented in paper A [25], uses a constructive, heuristic approach, potentially creating large numbers of artifacts. In paper B [26] we present an approach that generates optimal solutions with an ILP-based algorithm. It does so by deriving priority inequalities, which are then resolved by integer linear programming. The result provided by the method is the task attributes for FPS that yield the minimum number of artifact tasks. By using an ILP solver for the derivation of priorities, additional demands, such as reducing the number of preemption levels, can be added by inclusion in the goal function.

Finally, we use the method described in paper B [26] to implement off-line scheduling in Controller Area Network (CAN). The description of the approach was presented in paper C [27]. The ILP-formulation is modified to ensure unique priorities for the messages, as required by the CAN protocol.
The off-line analysis we perform in the proposed method is simplified when applied to CAN, due to the CAN properties which we can take advantage of:

- The message length is constant in CAN – This fact ease the off-line analysis we perform in our methods, as it avoids situations where the order of transmission may change as a consequence of variable transmission time. In task scheduling we assume task executing for worst case execution time (WCET) while the actual execution time at run-time is most likely much less.
- CAN scheduling is non-preemptive – In preemptive task scheduling, the execution order may change, due to variations in the execution times. In CAN this problem no longer exist since the message length is constant and no preemptions may occur.

An additional issue is that CAN scheduling require unique priorities. We solve the issue by simply adding an extra constraint to the ILP.
By using our method, we solve the issue of attribute assignment for a set of messages with complex constraints and schedule them on CAN while preserving the native CAN mechanism.

1.5 Results

This section summarizes the main contribution of each paper in the thesis.
The following papers are included in the thesis:

1.5.1 Paper A

“Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-line Schedules”, Radu Dobrin and Gerhard Fohler, In Proceedings of Conference on Real-Time Computer Systems and Applications, Korea, 2000

Summary In this paper, we present a method to combine off-line schedule construction with fixed priority scheduling by determining task attributes for the off-line scheduled tasks, such that the original schedule is reconstructed if the tasks are scheduled by FPS at run-time. The method analyzes an off-line schedule together with original task constraints to create sequences and windows of tasks. Priorities and offsets are set to ensure task orders in sequences and relations between windows. As FPS cannot reconstruct all schedules with periodic tasks, our algorithm can split tasks into several instances to achieve consistent task attributes. Lower priority tasks can be added for run-time use.

1.5.2 Paper B

"Translating Off-line Schedules into Task Attributes for Fixed Priority Scheduling ", Radu Dobrin, Peter Puschner and Gerhard Fohler, in Proceedings of Real-Time Systems Symposium, London, UK, 2001

Summary In this work we show how off-line scheduling and FPS run-time scheduling can be combined to get the advantages of both – the capability to cope with complex timing constraints and flexibility. We assume that a schedule for a set of tasks with complex constraints has been constructed off-line and we present a method to analyze the off-line schedule and derive an FPS task set with FPS attributes priority, offset, and period, such that the runtime FPS execution matches the off-line schedule. The proposed method analyzes the schedule and sets up inequality relations for the priorities of the tasks under FPS. Integer linear programming (ILP) is then used to find a FPS priority assignment that fulfills the relations. In case the priority relations for the tasks of the off-line schedule are not solvable we split tasks into the number of instances, to obtain a new task set with consistent task attributes. By using ILP, we can ensure that our schedule translation algorithm keeps the number of newly generated artifact tasks minimal.

1.5.3 Paper C

"Implementing Off-line Message Scheduling on Controller Area Network (CAN)", Radu Dobrin and Gerhard Fohler, in Proceedings of Emerging Technologies and Factory Automation, France 2001

Summary In this paper we apply the previously developed methods to take advantage of the benefits of off-line scheduling in controller area network (CAN). Assuming that a schedule, for a set of tasks transmitting messages on CAN, has been constructed off-line, we present a method that analyzes the off-line schedule and derives a set of periodic messages with fixed priorities, which can be scheduled on CAN. Based on the information provided by the off-line schedule, the method derives inequality relations between the priorities of the messages under FPS. In case the priority relations of the messages are not solvable, we split some messages into a number of artifacts, to obtain a new set of messages with consistent identifiers. We use integer linear programming to minimize the final number of messages.

1.6 Conclusions and Future Work

In this work we present methods that combine off-line schedule construction with fixed priority run-time scheduling. In particular, we want to take advantage of all benefits provided by off-line scheduling, while using fixed priority scheduling. We use off-line schedules to express complex constraints and predictability for selected tasks. Then, we derive attributes for tasks, such that, if applying FPS at run-time, the tasks will execute flexibly while fulfill their original constraints.

Thus, the methods solve issues arising from legacy systems, e.g., partition scheduling for avionics applications, and allows to handle constraints not covered by FPS feasibility tests, while using standard FPS at runtime. Also it provides for predictable flexibility, i.e., the restricted execution of selected tasks, e.g., for sampling and actuating in control systems, while enabling runtime flexibility for others. Furthermore, the method is applied to schedule messages with complex constraints on CAN. We use the information provided by an off-line schedule constructed to solve complex constraints and we derive attributes, i.e., message identifiers, required by CAN's native protocol. At run time, the messages are transmitted and received within time intervals such that the original constraints of the messages are fulfilled.

Our methods analyze an off-line schedule, constructed to solve complex constraints, and derives attributes for fixed priority scheduling such that the tasks, when scheduled by FPS, execute flexibly while fulfilling the original constraints. In certain cases, the method splits tasks into instances, creating artifact tasks, as not all off-line schedules can be expressed directly with FPS. A first, constructive approach, creates a potentially large amount of artifacts, while, later on, we use standard integer linear programing to solve priority inequalities derived from the off-line schedule and minimize the number of artifact tasks created. Finally, we assign offsets and periods to the task set provided by ILP in order to ensure the correct run-time execution.

In some cases, we have to perform additional splits, due to a violation of the

periodicity in the off-line schedule, which gives different offsets for different instances of the same task. By minimizing the number of artifact tasks, our method minimizes the number of offsets in the system as well.
Our methods does not introduce artifacts or reduce flexibility unless required by constraints: a set of FPS tasks, scheduled off-line according to FPS, and transformed by our method, executes in the same way as the original tasks.
To this point, we have concentrated on reconstructing the off-line schedule. Using the flexibility of the ILP solver, we can add objectives by inclusion in the goal function.
In this work, we assumed that all task dependencies have been resolved off-line. Future work will address the issue of task relations at run-time as well. Furthermore, we are currently investigating the possibility to improve FPS in terms of minimizing the number of preemptions yielded by an arbitrary set of tasks scheduled by FPS. At the same time, we aim to develop a common interface between original task constraints and the different scheduling strategies, i.e., off-line scheduling, FPS and EDF, to directly map complex constraints into attributes specific for each scheduling paradigm.

Bibliography

[1] J. A. Stankovic and K. Ramamritham. *IEEE Tutorial: Hard Real-Time Systems*. IEEE Computer Society Press, Washington, D.C., USA, 1988.

[2] H. Kopetz and G. Grunsteidl. TTP - a Protocol for Fault-Tolerant Real-Time Systems. *Computer*, 27(1):14–23, 1994.

[3] H. Kopetz. Why Time-Triggered Architectures will Succeed in Large Hard Real-Time Systems. In *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 2–9, 1995.

[4] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment. *Journ. of the ACM, 20, 1,* Jan. 1973.

[5] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-Time Tasks under Precedence Constraints. *Real-Time Systems Journal*, 2(3):181–194, Sept. 1990.

[6] J.C. Palencia and M. Gonzalez Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In *Proceedings of 20th IEEE Real-Time Systems Symposium*, 1999.

[7] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. ARINC Scheduling: Problem Definition. In *Proceedings of Real-Time Systems Symposium*, pages 165–169, 1994.

[8] M. Törngren. Fundamentals of Implementing Real-Time Control Applications in Distributed Computer Systems. *Real-Time Systems*, 1997.

[9] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham. Jitter Compensation for Real-Time Control Systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, Dec 2001.

[10] N.C. Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times. Technical report, Departament of Computer Science, University of York, 1991.

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

[12] K. Tindell, H. Hansson, and A.J. Wellings. Analizing Real-Time Communications: Controller Area Network (CAN). In *Proceedings of Real-Time Systems Symposium*, pages 259–263, Dec. 1994.

[13] K. Tindell, A. Burns, and A.J. Wellings. Calculating Controller Area Network (CAN) message response times. *Contr. Eng. Practice*, 3(8):1163–1169, 1995.

[14] J.Y-T. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2(4):237–250, Dec. 1982.

[15] G. Leen and D. Heffernan. Time-Triggered Controller Area Network. *Computing and Control Engineering*, 2001.

[16] L. Almeida, P. Pedreiras, and J. A. Fonseca. The FTT-CAN Protocol: Why and How. *IEEE Transactions on Industrial Electronics*, 49(6), Dec 2002.

[17] D. Seto, J.P. Lehoczky, and L. Sha. Task Period Selection and Schedulability in Real-Time Systems. In *Proceedings of Real-Time Systems Symposium*, pages 188–198, 1998.

[18] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: an Approach to Real-Time Synchronization. *IEEE Transactions on Computer*, 39(9):1175–1185, Sept 1990.

[19] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Tasks. *The Journal of Real-Time Systems*, 1989.

[20] K. Tindell. Adding Time Offsets to Schedulability Analysis. Technical report, Departament of Computer Science, University of York, January 1994.

[21] J.C. Palencia and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proceedings of 19th IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[22] M. Gonzalez Harbour and J.P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Varying Execution Priority. In *Proceedings of Real-Time Systems Symposium*, pages 116–128, Dec. 1991.

[23] J. Xu and D. L. Parnas. Priority Scheduling versus Pre-run-time Scheduling. *Real-Time Systems*, 2000.

[24] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Austria, Apr. 1994.

[25] R. Dobrin, Y. Özdemir, and G. Fohler. Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-line Schedules. In *Conference on Real-Time Computing Systems and Applications, Korea*, December 2000.

[26] R. Dobrin, P. Puschner, and G. Fohler. Translating Off-line Schedules into Task Attributes for Fixed Priority Scheduling. In *Proceedings of Real-Time Systems Symposium*, December 2001.

[27] R. Dobrin and G. Fohler. Implementing Off-line Scheduling in Controller Area Network (CAN). *Proc. of the 8th International Conference on Emerging Technologies and Factory Automation*, 2001.

Chapter 2

Paper A: Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-line Schedules

Radu Dobrin and Gerhard Fohler
In Proceedings of Conference on Real-Time Computer Sytems and Applications, Korea, 2000.

Abstract

A number of industrial applications advocate the use of time triggered approaches for reasons of predictability, distribution, and particular constraints such as jitter or end-to-end deadlines. The rigid offline scheduling schemes used for time triggered systems, however, do not provide for flexibility. Fixed priority scheduling can provide more flexibility, but is limited with respect to predictability, as actual times of executions depend on run-time events. In this paper, we present a method to combine off-line schedule construction with fixed priority scheduling: by determining task attributes for the off-line scheduled tasks, such that the original schedule is reconstructed if scheduled with FPS at run-time. It analyzes an off-line schedule together with original task constraints to create sequences and windows of tasks. Priorities and offsets are set to ensure task orders in sequences and relation between windows. As FPS cannot reconstruct all schedules with periodic tasks, our algorithm can split tasks into several instances to achieve consistent task attributes. Lower priority tasks can be added for run-time use.

2.1 Introduction

Fixed priority scheduling (FPS) has been widely studied and used in a number of applications, mostly due by its simple run-time scheduling and resulting small overhead. Modifications to the basic scheme to handle semaphores [1], aperiodic tasks [2], static [3] and dynamic [4] offsets, and precedence constraints [5], have been presented. Consequently, FPS enables good flexibility for tasks with incompletely known attributes. Temporal analysis of FPS algorithms focuses on meeting deadlines, i.e., guarantees that all instances of tasks will finish before their deadlines. The actual times of executions of tasks, however, are generally not known and depend largely on run-time events, compromising predictability.

Off-line scheduling for time-triggered systems, on the other hand, provides strong predictability, as all times for task executions are determined and known in advance. In addition, complex constraints can be solved off-line, such as distribution, end-to-end deadlines, precedence, jitter, or instance separation. All this is enabled at the expense of loosing run-time flexibility, as all actions have to be planned before.

In this paper, we present an algorithm to combine off-line schedule construction with fixed priority run-time scheduling. The resulting systems have a time-triggered base that is complemented with even-triggered on-line scheduling. This allows us to combine benefits of off-line scheduling, in particular a distributed system, complex, constrained tasks, and end-to-end deadlines, with online scheduling, which allows flexible task execution. A number of tasks are specified to execute predictable, while allowing flexibility for all others.

Our method works by transforming off-line scheduled tasks with their original constraints into tasks with attributes suited for fixed priority scheduling, i.e., periods, deadlines, and offsets, which will reenact the original offline schedule at runtime. It divides the off-line schedule and its tasks into windows and sequences, sets priorities to ensure execution orders within windows, and determines priorities and offsets to ensure orders and relations between windows. As FPS cannot reconstruct all schedules with periodic tasks, our algorithm can split tasks into several instances to achieve consistent task attributes. Tasks with lower priorities can be added for run-time scheduling.

Priority assignment for FPS tasks has been studied in, e.g., [6], [7], and [8] study the derivation of task attributes to meet a overall constraints, e.g., demanded by control performance. Instead of specific requirements, our algorithm takes an entire off-line schedule and all task requirements to determine task attributes. A method to transform off-line schedules into earliest deadline

first tasks has been presented in [9].
The rest of this paper is organized as follows: section 2.2 describes the problem and introduces items and terms used in the paper. Section 2.3 and 2.4 describe basic idea and algorithm, which is illustrated by an example in section 2.5. Summary and outlook in section 2.6 conclude the paper.

2.2 Problem Description

2.2.1 Offline Schedule

First, an off-line schedule is created for a set of tasks and constraints. While our method does not rely on a particular off-line scheduling algorithm, we have used the one described in [3] for our implementation and analysis. The schedule is usually created up to the least common multiple, LCM, of all task periods. LCM/T(T_i) instances of each task T_i with period T(T_i) will execute in the schedule.
The off-line scheduler resolves constraints such as distribution, end-to-end deadlines, precedence, etc, and creates scheduling tables for each node in the system, listing start- and finishing-times of all task executions. These scheduling tables are more fixed than required by the original constraints, so we can replace the exact start- and finishing-times of tasks with feasibility windows, taking the original constraints into account. A task receiving (sending) a message over the network, for example, has to start (finish) after (before) the scheduled transmission time, giving more leeway than the rigid scheduling table, defining release times (deadlines). Slot shifting [4] uses this method to transform off-line schedules into task for earliest deadline first scheduling.
Feasibility windows $WF(T_i^j)$ of each instance T_i^j of each task T_i, are derived from schedule and constraints transformed into earliest start times and deadlines.
The **earliest start time,** $est(T_i^j)$, of an instance T_i^j of a task T_i, is provided by the task constraints expressed in the offline schedule.
The **scheduled finishing time,** $finish(T_i^j)$ of an task instance is the time when T_i^j, is completing its execution according to the offline schedule.
The **scheduled start time,** $start(T_i^j)$, of an task instance is the time when T_i^j is starting its execution according to the offline schedule.
A **sequence** $S(t_k)$ consists of instances of tasks T_i^j ordered increasingly after theirs scheduled start times, such that

$$start(WF(T_i^j)) = t_k \vee start(WF(T_i^j)) < t_k \wedge finish(T_i^j) > t_k$$

according to the order of execution in the offline schedule. Additionally, we define:

$$\begin{aligned} first(S(t_k)) &= S1 = first\ task\ instance\ in\ the\ sequence\ S(t_k) \\ last(S(t_k)) &= SN = last\ task\ instance\ in\ S(t_k) \end{aligned}$$

Consequently:

$$\begin{aligned} S(t_k) &= \{Sn | Sn = T_i^j, where \\ &start(WF(T_i^j)) = t_k, or \\ &start(WF(T_i^j)) < t_k \wedge finish(T_i^j) > t_k\} \end{aligned}$$

Additionally:

$$start(S1) < start(S2) < \cdots < start(SN)$$

and

$$S1 = first(S(t_k))\ and\ SN = last(S(t_k))$$

2.2.2 Online Scheduling

At run-time, we want tasks to be scheduled according to fixed priority assignment. Our method assigns priorities, offsets, deadlines, periods. We refer to an *execution window*, $Wexec(T_i^j)$, of an instance T_i^j of a task T_i, as the time interval in which T_i^j will execute and complete if scheduled by FPS together with the other instances of the other tasks.

2.2.3 Problem Statement

Given a set of tasks T_i , i=1,2,…,n with earliest start times $est(T_i)$, deadlines $dl(T_i)$, periods $T(T_i)$, offline scheduled, with constraints represented by feasibility windows, we want to find fixed priorities, fixed offsets and deadlines for these tasks, $P(T_i)$, $O(T_i)$, $dl(T_i)$, such that the execution windows of each task $Wexec(T_i)$ given by FPS will be contained within the respective feasibility window $WF(T_i)$ and the sequence ordering kept. In case of priority conflict, i.e., if assigning the same priorities to all instances of a task fail, we want to find priorities, offsets, and deadlines, such that each of these instances executes within its respective feasibility window for a small number of instances.

2.3 Overview – Basic Idea

2.3.1 Overview

Our method performs the following steps transforming the offline schedule into FPS tasks (figure 2.1).

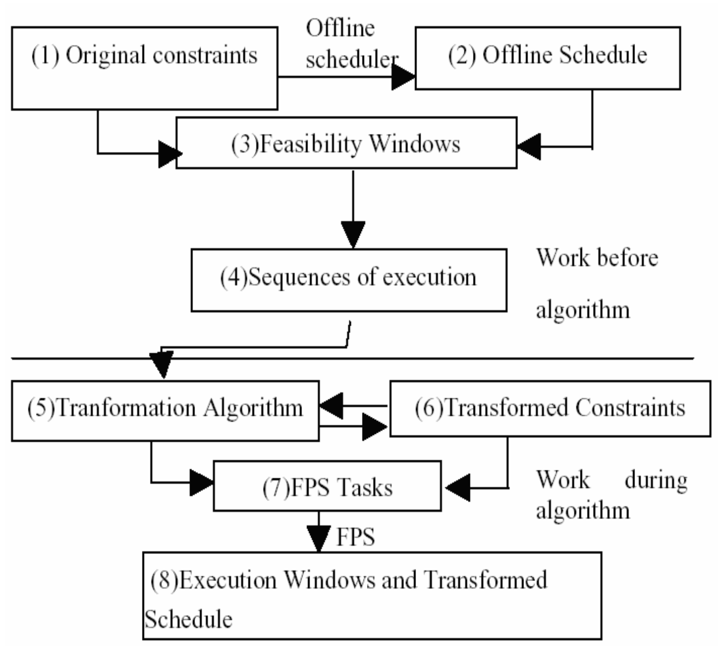


Figure 2.1: Method overview

1. Initially, tasks are given with their original constraints, and attributes, including worst case computation times, and periods.

2. A standard offline scheduling-algorithm constructs offline-scheduling tables with (1) as input.

3. Feasibility windows for each instance of each task are derived from scheduling tables and original task constraints

4. Sequences $S(t_k) \in WF(T_i^j)$ are now straightforward to derive from the feasibility windows.

5. and 6. perform the actual attribute assignment creating FPS task

7. Whenever a priority conflict arises during the priority settings, we split a task T_i into instances for different priorities.

8. Having the set of tasks with priorities, offsets, periods, deadlines, it is straightforward to schedule these using FPS.

2.3.2 Basic idea of attribute assignment algorithm

Our algorithm determines priorities by traversing the off-line schedule represented by the series of feasibility windows in increasing order of time (left to right). It sets priorities to reflect the positions of task instances in sequences, i.e., later position gives lower priority. The algorithm attempts to keep the same priorities for all instances of a task. This may, however, lead to an inconsistent assignment. Then the algorithm splits the task into instances and assigns new attributes for each.

2.4 Algorithm

2.4.1 Assumptions

Our method is based on the following assumptions:

- Task deadlines are less than the task periods.
- The computation-time of a task has a known, upper bound.
- Tasks execute as soon as they are ready and have the highest priority.
- Tasks are independent and fully preemptive. All dependencies have been resolved in the off-line schedule.
- A higher numerical value for priority represents higher priority.

2.4.2 Attribute assignments

It suffices to set priorities according to the sequence of tasks of a feasibility window, if it does not overlap with any other feasibility window. In the case of overlaps, however, the order in more than one sequence has to be taken into account for the priority assignment.
Our algorithm traverses the schedule in the sequence of feasibility windows and performs two operations: **set** and **check + split/update**. *Set* is performed on each first instance T_i^1 of each task T_i by assigning a priority to the task T_i according to the sequence $S(t_k)$, $t_k = Start(WF(T_i^1))$. For the rest of the instances of T_i we *check* if the priority assigned by the *set* operation will schedule T_i^j within its feasibility window, in the same position in the sequence $S(t_k)$ given by the offline schedule. It may happen, however, that the ordering of tasks may be different for some instances. These cases cannot be expressed directly with fixed priority assignment, leading to inconsistent priority assignment. Our algorithm detects this, and circumvents the problem by splitting a task into its instances by performing *split* and *update* on all the previous instances of T_i^j by updating offsets, periods and deadlines and by that, treating all instances of T_i as individual tasks.
So, the priorities of instances T_i^j of tasks T_i with $est(T_i^j) = start(WF(T_i^j))$ are based upon either:

- the priority of the first task instance in sequence $S(t_k)$ in this window or
- the priority of another task assigned to this window which was set in a previous step

Initially, no priorities are set: $\forall i, j,\ P(T_i^j) = NULL$, offsets are equals to $start(WF(T_i^j))$: $\forall i, j,\ O(T_i^j) = start(WF(T_i^j))$, and deadlines are set to the ends of respective feasibility windows: $\forall i, j,\ D(T_i^j) = end(WF(T_i^j))$. The priority of $T_i^j \in S(t_k) = [T1, T2, \ldots, Tn]$ must be set/checked to be in descending order: $P(T1) > P(T2) > \cdots > P(Tn)$.

2.4.3 Task attribute assignment algorithm

In this section, we present the pseudo code for the proposed method.
For each start of feasibility window t_k, $k \leq nr\ of\ feasibility\ windows$, $\forall i, j\ such\ that\ est(T_i^j) = t_k$:

1. if $j = 1$ (the first instance of task T_i)

(a) $if \; \exists \, T_m^n \in S(t_k) \; s.t. \; n > 1 \wedge (P(T_m^n) \neq NULL)$
($P(T_m^n)$ is the priority of T_m^n one instance in sequence has already pre-set priority, then **set** priorities in sequence accordingly)

i. if T_m^n is scheduled to execute after T_i^j (according o the sequence $S(t_k)$), then:
set $P(T_i^j) = P(T_m^n) + (x+1) * c$
(where x=nr. of executing tasks belonging to $WF(T_i^j)$ between T_i^j and T_m^n and c is a constant)

ii. else (T_m^n is scheduled to execute before T_i^j according to the sequence $S(t_k)$):
set $P(T_i^j) = P(T_m^n) - (x+1) * c$

(b) else ($\forall m, n$ s.t. $T_m^n \in S(t_k), n = 1$, i.e., we have only first instances in the sequence $S(t_k)$)
set $P(firstS(t_k)) = P(S1) = default = p$
set $P(T_i^j) = P(S1) - x * c$
(initial number priorities can be adjusted after assignment, $S1 = First(S(t_k)$ and x=nr. of executing tasks (with higher priority) in $S(t_k)$ before T_i^j)

2. else ($j > 1$, and $P(T_i^j) = P(T_i^{(j-1)}) = \cdots = P(T_i^1)$, e.g., priorities for all instances have already been set/checked in a previous step to the same value)
$\forall m, n, \; m \neq i, \; n \neq j$, such that $T_m^n \in S(T_k)$ and $P(T_m^n) \neq NULL$,
Check priority of T_i^j :

(a) if T_m^n is scheduled to execute after T_i^j (according to the sequence $S(t_k)$)

- if $P(T_i^j) > P(T_m^n) \Rightarrow$ **OK!**
- else ($P(T_i^j) < P(T_m^n) \Rightarrow$ **priority conflict!** $\Rightarrow$ **Split/Update** (section 2.4.4)

(b) if T_m^n is scheduled to execute before T_i^j(according to the sequence $S(t_k)$)

- if $P(T_i^j) < P(T_m^n) \Rightarrow$ **OK!**
- else ($P(T_i^j) > P(T_m^n) \Rightarrow$ **priority conflict!** $\Rightarrow$ **Split/Update** (section 2.4.4)

2.4.4 Priority assignment conflict – Instance splitting

Two instances, T_m and T_n^j, of two tasks T_m, T_n, have a priority-relation, $P(T_m^i) > P(T_n^j)$, in an feasibility window $WF(T_m^i) or WF(T_n^j)$. Now, suppose that at a later point in time the relation between these two priority assignments is contradicted: $P(T_m^{(i+k)}) < P(T_n^{(j+p)})$. Either $T_m^{(i+k)}$ must have a different priority than the previous instance T_m^i, or $T_n^{(j+p)}$ than T_n^j. When a priority assignment conflict arises between two instances T_m^i and T_n^j, we have to change the priority of one of the instances, which causes the conflict, i.e., T_m^i or T_n^j. One way to solve this problem is to split one of these two tasks instances. The result is that the split instances will be treated as unique tasks with their own attributes.
The selection of which task to split depends on the tasks period and whether it is possible to split this task. The first choice is to split the task with the largest period, because that gives the least number of new tasks (least number of instances). A task T_m can be split without risking further conflicts, if the earliest start time of the conflicting instance T_m^i is the start of the current feasibility window $WF(T_m^i)$.
If we split a task T_m whose instance T_m^i is released before the start of the current feasibility window, a priority change of T_m^i could lead to another conflict in one of its previous feasibility window where the tasks priority has already been checked in a previous step. Further on, the successive instances of T_m will be treated as first instances of new tasks.
When we split a task T_m, the attributes of all instances of T_m^k, $k = 1, 2, \ldots$ will be updated, with:

$$\begin{aligned} offset(T_m^k) &= (k-1) * period(T_m) \\ dl(T_m^k) &= k * period(T_m) + dl(T_m) \\ period(T_m^k) &= LCM \end{aligned}$$

2.5 Example

We illustrate the algorithm with an example. Lets assume that we have the following task-set: A(15,2), B(15,1), C(15,5), D(10,3) where T(Period, computation time).The original offline schedule for the taskset is shown in figure 2.2. The feasibility windows for the task instances starts at $t_1 = 0$, $t_2 = 10$, $t_3 = 15$, $t_4 = 20$. At time $t_1 = 0$ (table 2.1), the instances whose earliest start

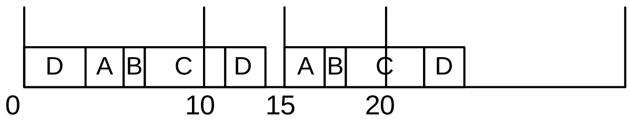


Figure 2.2: Off-line schedule

times are equal to t_1 are D1, A1, B1, C1. We set the default priority (in this case 100) on D and lower priorities of A, B and C according to the sequence $S(t_1) = D, A, B, C$.

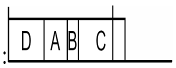
t_k	$est(T_i^j) = t_k$	$est(T_i^j) = t_k \& f(T_i^j) > t_k$	operation	status
0	D^1, A^1, B^1, C^1	NONE	setP(D)=100 setP(A)=99 setP(B)=98 setP(C)=97	default
	$S(t_1)$: 			

Table 2.1: Sequence at $t_1 = 0$

At time $t_2 = 10$, (table 2.2) the second instance of D has its earliest start time equal to t_2. Since D has already been assigned the priority P(D)=100 at time t_1, we check if this priority will schedule the second instance of D into its off-line scheduled position according to $S(t_2) = C, D$. In this case, since Cs already assigned priority, P(C)=97, is lower than P(D)=100, we have a priority conflict. We have to choose which task to split. The first candidate is C because the number of invocations of C during LCM, 2, is lower then the number of invocations of D, 3. Since Cs earliest start time for this instance is, however, $t_1 < est(D)$, we have to split and update D, creating three tasks D1, D2, D3. The previous instance of D, D1, is updated with T(D1)=LCM, O(D1)=est(D1)=0, dl(D1)= end(WF(D1)) =10, P(D1)=100. D2. The instance creating the conflict, D2, is treated as the first instance of an new task D2, with T(D2) = LCM, O(D2) = est(D2) = 10, dl(D2)= end(WF(D2))=20, and with a priority based on the priority of C, according to the sequence $S(t_2) = C, D2$, P(D2)=P(C)-1=96.
At time $t_4 = 20$ (table 2.3), we have the first instance of the new task D3 with its earliest start time equal to t_3. Since the task has no priority, we set P(D3) based on the priority of the C, according to the sequence $S(t_4) = C, D3$,

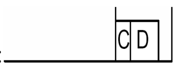
10	D^2	C^1	check P(D) split D	conflict!
	$S(t_2)$: 			

Table 2.2: Sequence at $t_2 = 10$

P(D3)=P(C)-1 =96.

15	A^2, B^2, C^2	NONE	check P(A) check P(B) check P(C)	OK! OK! OK!
20	D3	C^2	set P(D3)=96	

Table 2.3: Sequence at $t_4 = 20$

The final set of tasks provided by our algorithm is shown in table 2.4.

$Task$	C	T	O	dl	priority
A	2	15	0	15	99
B	1	15	0	15	98
C	5	15	0	15	97
D1	3	30	0	10	100
D2	3	30	10	20	96
D3	3	30	20	30	96

Table 2.4: FPS tasks

2.6 Summary and Outlook

In this paper, we presented a method to combine off-line schedule construction with fixed priority run-time scheduling. It uses off-line schedules to express complex constraints and predictability for selected tasks, while providing flexibility for the remaining tasks and newly added ones at run-time. At first, it

analyses the off-line schedule together with the original task constraints and divides into windows and sequences. Then, it sets priorities and offsets to ensure task orders in sequences and relation between windows. The algorithm splits periodic tasks into several instances, as not all schedules can be expressed with FPS. Using fixed priority scheduling at run-time, the tasks with computed attributes will execute to *reenact* the original offline schedule. New, lower priority tasks can be added for run-time execution.
We have implemented the described methods and tested the results both via response time analysis and the original off-line schedule and original task constraints. Our algorithm determines task attributes trying to keep priorities for all instances of periodic tasks the same. This will lead to inconsistent priority assignments for some schedules, e.g., those created with a earliest deadline first strategy. Our algorithm attempts to resolve the arising priority conflicts by splitting the task into several instances with different priorities. We are currently investigating this issue further to try to reduce the number of instances with different attributes created.
We assume here, that task dependencies have been resolved off-line. Future work will address the issue of task relations at run-time as well.

Bibliography

[1] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: an Approach to Real-Time Synchronization. *IEEE Transactions on Computer*, 39(9):1175–1185, Sept 1990.

[2] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Tasks. *The Journal of Real-Time Systems*, 1989.

[3] K. Tindell. Adding Time Offsets to Schedulability Analysis. Technical report, Departament of Computer Science, University of York, January 1994.

[4] J.C. Palencia and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proceedings of 19th IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[5] M. Gonzalez Harbour and J.P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Varying Execution Priority. In *Proceedings of Real-Time Systems Symposium*, pages 116–128, Dec. 1991.

[6] N.C. Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times. Technical report, Departament of Computer Science, University of York, 1991.

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

[8] D. Seto, J.P. Lehoczky, and L. Sha. Task Period Selection and Schedulability in Real-Time Systems. In *Proceedings of Real-Time Systems Symposium*, pages 188–198, 1998.

[9] G. Fohler. Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems. In *Proc. 16th Real-time Systems Symposium*, Pisa, Italy, 1995.

Chapter 3

Paper B: Translating Off-line Schedules into Task Attributes for Fixed Priority Scheduling

Radu Dobrin, Gerhard Fohler and Peter Puschner
In Proceedings of Real-Time Systems Symposium, London, UK, 2001.

Abstract

Off-line scheduling and fixed priority scheduling (FPS) are often considered as complementing and incompatible paradigms. A number of industrial applications demand temporal properties (predictability, jitter constraints, end-to-end deadlines, etc.) that are typically achieved by using off-line scheduling. The rigid off-line scheduling schemes used, however, do not provide for flexibility. FPS has been widely studied and used in a number of applications, mostly due to its simple run-time scheduling, and small overhead. It can provide more flexibility, but is limited with respect to predictability, as actual start and completion times of execution depend on run-time events.

In this paper we show how off-line scheduling and FPS run-time scheduling can be combined to get the advantages of both – the capability to cope with complex timing constraints and flexibility. The paper assumes that a schedule for a set of tasks with complex constraints has been constructed off-line. It presents a method to analyze the off-line schedule and derive an FPS task set with FPS attributes priority, offset, and period, such that the runtime FPS execution matches the off-line schedule. It does so by analyzing the schedule and setting up inequality relations for the priorities of the tasks under FPS. Integer linear programming (ILP) is then used to find a FPS priority assignment that fulfills the relations. In case the priority relations for the tasks of the off-line schedule are not solvable we split tasks into the number of instances, to obtain a new task set with consistent task attributes. Our schedule translation algorithm keeps the number of newly generated artifact tasks minimal.

3.1 Introduction

Off-line scheduling and fixed priority scheduling (FPS) are often considered as having incompatible paradigms, but complementing properties. FPS has been widely studied and used in a number of applications, mostly due its simple run-time scheduling, small overhead, and good flexibility for tasks with incompletely known attributes. Modifications to the basic scheme to handle semaphores [1], aperiodic tasks [2], static [3] and dynamic [4] offsets, and precedence constraints [5], have been presented. Temporal analysis of FPS algorithms focuses on providing guarantees that all instances of tasks will finish before their deadlines. The actual start and completion times of execution of tasks, however, are generally not known and depend largely on run-time events, compromising predictability.
Off-line scheduling for time-triggered systems, on the other hand, provides determinism, as all times for task executions are determined and known in advance. In addition, complex constraints can be solved off-line, such as distribution, end-to-end deadlines, precedence, jitter, or instance separation. As all actions have to be planned before startup, run-time flexibility is lacking.
In this paper we present work to combine FPS with off-line schedule construction. It assumes a schedule has been constructed off-line for a set of tasks to meet their complex constraints. Our method takes the schedule and assigns the FPS attributes priority, offset, and period, to the tasks, such that their runtime FPS execution matches the off-line schedule. It does so by deriving priority inequalities, which are then resolved by integer linear programming.
Instead of enhancing only either FPS or off-line scheduling alone, our method provides for a combination, such that benefits of either scheme are accessible to the other. In particular, the presented method provides solutions for the following scenarios:

Legacy Systems: Some safety critical applications, e.g., from the avionics domain [6], demand temporal partitioning of task executions or assertions not only about deadline being met, but restrictions on the actual times when task executions are performed. Typically, such applications are executed in time triggered architectures with off-line schedule construction. A move to fixed priority based systems has to ensure the specific demands will be met, which may be cumbersome applying standard FPS methods, as these concentrate on deadlines primarily.
Our method transforms these demands directly into attributes for tasks to be feasibly scheduled with FPS, pertaining the predictability provided by off-line

schemes.

FPS systems with unresolved constraints: The feasibility test provided for FPS tasks defines the types of constraints which can be met. Additional constraints or combinations require modifications to existing tests or the development of new ones, which may not be available in limited time.
In addition, constraints demanded by complex applications, cannot be expressed generally: Control applications may require constraints on individual instances rather than fixed periods, reliability demands can enforce allocation and separation patterns, or engineering practice may require relations between system activities.
Our method resolves the need for developing of new specific tests for unusual constraints: first, a designer will apply known tests on the application under consideration. Should standard schemes prove to be incapable, the designer submits these tasks to an off-line scheduling scheme, which can use elaborate and general methods, such as search or constraint satisfaction, to provide a feasible off-line schedule. Then, our method derives attributes period, priority, and offset for these tasks, such that they can be scheduled with FPS, while meeting the specific constraints.

Predictable flexibility: Off-line scheduling provides deterministic execution patterns for all tasks in the system, while FPS schemes provide flexibility for all tasks. Only few applications, however, will demand either determinism or flexibility uniformly for all activities in the system. Rather, only few selected tasks have tight restrictions on their executions, e.g., those sampling or actuating in a control system, with strict demands on jitter and variability, while a majority can execute flexibly.
Our method allows the amount of flexibility at runtime to be set off-line in a predictable way by including restrictions on task execution as input to the transformation algorithm.
FPS cannot reconstruct all schedules with periodic tasks with the same priorities for all instances directly. The constraints expressed via the off-line schedule may require that instances of a given set of tasks need to be executed in different order on different occasions. Obviously, there exist no valid FPS priority assignment that can achieve these different orders. Our algorithm detects such situations, and circumvents the problem by splitting a task into its instances. Then, the algorithm assigns different priorities to the newly generated ”artifact” tasks, the former instances.

Key issues in resolving the priority conflicts are the number of artifact tasks created, and the number of priority levels. Depending on how the priority conflict is resolved, the number of resulting tasks may vary, depending on the periods of the split tasks. Our algorithm minimizes the number of artifact tasks. By using an ILP solver for the derivation of priorities, additional demands such as reducing number of preemptions levels can be added by inclusion in the goal function.
Priority assignment for FPS tasks has, for example, been studied in [7] and [8]. [9] study the derivation of task attributes to meet overall constraints, e.g., demanded by control performance. Instead of specific requirements, our algorithm takes an entire off-line schedule and all task requirements to determine task attributes. A method to transform off-line schedules into earliest deadline first tasks has been presented in [10]. A related paper [11] deals with priority assignment for off-line schedules. It uses a constructive, heuristic approach, potentially creating large numbers of artifacts, while the approach presented here generates optimal solutions with an ILP-based algorithm.
The paper is organized as follows: in Section 3.2 we describe the rationale and how our method relates off-line scheduling and FPS. In Section 3.3 we describe the problem together with the basic idea and the algorithm, which is illustrated by an example in Section 3.4. In Section 3.5 we present conclusions and further work, and we conclude the paper with proofs in appendix 3.7.

3.2 Off-line Schedules

In this section, we discuss the rationale of our method and position its functionality with respect to application timing constraints, off-line scheduler, and FPS online execution of tasks. We discuss the complexity reduction of the NP hard scheduling problem with general constraints achieved by the off-line scheduler and how the new method provides for a selective choice of degree of online flexibility of the resulting FPS tasks.

Before starting the discussion, we introduce the term *target windows*.

3.2.1 Target windows

We define the target window of as task is the interval of time in which the instance will execute and complete at run-time. For example, the target window of a task scheduled by the RM algorithm will be the period of the task. The target window of a task scheduled off-line will consist of the time slots that the off-line scheduler assigned to the task.

3.2.2 Time triggered system operation and schedule construction

We assume a distributed system of stand-alone computers connected via a shared network. Tasks are allocated to these nodes, communicate across the system, and are demanded to fulfill complex constraints, such as precedence, end-to-end deadlines, and jitter. The off-line scheduler allocates tasks to nodes and resolves complex constraints by determining windows for tasks to execute in, and sequences, usually stored in scheduling tables. The resulting off-line schedule is one feasible, likely suboptimal solution. At run-time, a simple dispatcher selects which task to execute from the scheduling tables, ensuring tasks execute within the windows and thus meet their constraints.
This way, the complexity of the original scheduling problem is reduced off-line, which allows for elaborate methods, improvement of results and modifications in the failure case.

3.2.3 FPS reenaction of off-line schedules

For off-line tasks, typically, the runtime dispatcher is invoked in each node at regular time intervals, *slots*, and performs table lookup for task selection. While being simple, this is not the only way to ensure tasks execute in the windows, from now on called *target windows*, and in the order computed by the off-line scheduler.
In this paper, we propose to use standard fixed priority scheduling instead: by deriving priorities and offsets for tasks in such a way that tasks execute within their target windows and fulfill the precedence requirements when scheduled by FPS, the system will reenact the off-line schedule. Thus, the advantages of deterministic off-line scheduling can be combined with FPS scheduling at runtime.

3.2.4 Increased runtime flexibility

By modifying target windows, flexibility can be increased at runtime, while keeping the original constraints. Target windows created by the offline scheduler resolve two types of constraints: *temporal*, e.g., start of periods, end-to-end deadlines, sending or receipt of messages over the network, or jitter, in-

stance separation, etc, and *order of task execution*, as determined by the offline scheduler, e.g., for data flow processing, precedence, mutual exclusion. The offline scheduler resolves both types of constraints by assigning absolute points in time for the execution of all tasks. While the times for temporal constraints have to be kept, e.g., a task cannot execute after its deadline, order constraints are relative, i.e., tasks can execute earlier provided the execution order of the schedule is maintained. The offline schedule, however, prevents tasks from executing at a different time, even if resources become available earlier, e.g., by early termination of a task, i.e., the schedule is over constrained.
We provide an execution pattern which is more flexible than interpreting an offline schedule but nevertheless guarantees to meet the given constraints. We propose to join target windows of order constrained tasks which have the same temporal constraints. These tasks form chains according to their order inside these new target windows. Thus, we can exploit more flexibility while maintaining the constraints resolved by the offline schedule.

3.2.5 Selectively reduced runtime flexibility

While desirable in general, additional flexibility may be harmfull for some tasks, e.g., those sampling and actuating in a control system. For such tasks, the deterministic execution provided by the offline schedule has to be pertained. We propose to prevent the joining of target windows or to reduce the length of some windows selectively to keep the strict execution behaviour of selected tasks, while providing flexibility for the rest. Thus, our methods allows the amount of run-time flexibility of a task to be set offline in a predictable way.

3.3 Attribute assignment algorithm

We present a method which determines attributes for tasks assigned to target windows and associated chains such that, if executed according to fixed priority scheduling, they will execute inside their target window and obey the order constraint of the task chains.

3.3.1 Problem Definition

Given a set of off-line scheduled tasks, $Original\_Tasks$ where

$$Original\_Tasks = \{T_1, T_2, \ldots, T_n\}$$

with constraints represented by target windows, $TW_i,\ i = 1, 2, \ldots$, we want to transform them into a set of tasks, *FPS_Tasks* where

$$FPS\_Tasks = \{\overline{T_1}, \overline{T_2}, \ldots, \overline{T_m}\},$$

with attributes suitable for FPS, i.e., $prio(\overline{T_i})$, $o(\overline{T_i})$, $dl(\overline{T_i})$, $p(\overline{T_i})$, such that:

1. Each instance of each task $\overline{T_i}$ will execute at run time inside its target window
2. The order of execution enforced by the original task constraints is preserved

if the tasks $\overline{T_i} \in \{FPS\_Tasks\}$ are scheduled by FPS.

3.3.2 Algorithm Overview

As input to our method we have:
Taskset $T_i \in \{Original\_Tasks\},\ i = 1, 2, \ldots$, with constraints expressed in:

- Off-line schedule, up to LCM, expressing the original task constraints, that gives off-line scheduled start- and finishing times for each instance T_i^j of each task $T_i,\ st(T_i^j),\ ft(T_i^j)$
- Target windows, $TW_n,\ n = 1, 2, \ldots$, that gives earliest start times and deadlines for each instance T_i^j of each task T_i:
  $est(T_i^j) = begin(TW_n) = begin(TW(T_i^j)),\ and$
  $dl(T_i^j) = end(TW_n) = end(TW(T_i^j))$

We start with target windows and sequences of instances. We translate order constraints into priority constraints between the new FP tasks.
We may not be able to find a FPS schedule with the same number of tasks as the original one, but we may have to create new tasks by splitting some of the original off-line tasks. The resulting number of FPS tasks is to be minimized.

Output: we are looking for a set of tasks, $\overline{T_i} \in \{FPS\ Tasks\}$, with:

- Priorities, $prio(\overline{T_i})$
- Offsets, $o(\overline{T_i})$
- Periods, $p(\overline{T_i}),\ where\ \overline{T_i} \in \{FPS\_Tasks\}$

3.3.3 Derivation of the Inequalities

Given the target windows derived from the original constraints and the off-line schedule and defined as $TW_n = \{T_i^j \mid est(T_i^j) = t_k = begin(TW_n)$, and $dl(T_i^j) = end(TW_n)\}$, where $begin(TW_n)$ and $end(TW_n)$ are the starting time and the end time of the n^{th} target window, we derive the *sequences of tasks* corresponding to the start of each target window. The derivation of the sequences is illustrated in Figure 3.1.

A **sequence of tasks** SEQ_k consists of task instances, ordered by increasing scheduled start times according to the off-line schedule. A sequence may contain instances T_i^j of tasks T_i such that $est(T_i^j) = t_k$, referred to as $\{current\ instances\}_{t_k}$, or instances T_s^q of tasks T_s from overlapping target windows such that $est(T_s^q) < t_k$ and $ft(T_s^q) > t_k$, which we refer to as $\{interfering\ instances\}_{t_k}$, where $t_k = begin(TW_n)$.

$$\begin{aligned} SEQ_k &= \{\{current\ instances\}_{t_k} \cup \\ &\quad \cup \{interfering\ instances\}_{t_k}\}_{ordered} = \\ &= <\overset{k}{S}_1, \overset{k}{S}_2, \ldots, \overset{k}{S}_N> \end{aligned}$$

Where:

- $\{current\ instances\}_{t_k} = \{T_i^j \mid est(T_i^j) = t_k\}$
- $\{interfering\ instances\}_{t_k} = \{T_s^q \mid est(T_s^q) < t_k \wedge ft(T_s^q) > t_k\}$
- $first(SEQ_k) = \overset{k}{S}_1 = first\ task\ instance\ in\ SEQ_k$
- $last(SEQ_k) = \overset{k}{S}_N = last\ task\ instance\ in\ SEQ_k$

The priority assignment has to preserve the execution order expressed in the off-line schedule. Therefore, from each sequence of tasks SEQ_k, $k = 1, 2, \ldots$, we derive priority relations between the task instances within SEQ_k.

$$prio(\overset{k}{S}_1) > prio(\overset{k}{S}_2) > \ldots > prio(\overset{k}{S}_N)$$

The priority inequality system derived from the sequences of tasks, includes all task instances in the off-line schedule.

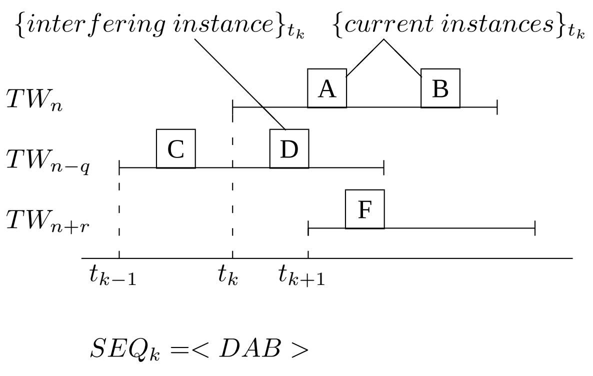


Figure 3.1: Sequence of tasks.

3.3.4 Attribute Assignment - Conflicts

Based on the order of execution expressed by the inequalities derived in Section 3.3.3, we derive attributes - priorities and offsets - for each task.
Our goal is to provide tasks with fixed offsets and fixed priorities. It may happen, however, that we have to assign different offsets/priorities to different instances of the same task, in order to reenact the off-line schedule at run time. These cases cannot be expressed directly with fixed priorities and fixed offsets and are the sources for *offset assignment conflicts* or *priority assignment conflicts*. In both cases, we split the conflicting task into instances such that, further on, each instance will be considered as an independent task with one instance during LCM.
By *offset assignment conflict* we mean that different instances of the same task have to be assigned different offsets in order to ensure the run-time execution of each one of them in the derived target window.

```
for $1 \leq i \leq nr\_of\_off - line\_scheduled\_tasks$
    for $1 \leq j \leq n,\ where\ n = nr\_of\_instances(T_i)$
        if: $begin(TW(T_i^j)) - (j-1)p(T_i) \neq$
            $\neq begin(TW(T_i^{j+1})) - j * p(T_i),$
        Then: $We\ split\ T_i\ into\ T_{i,1},\ T_{i,2}, \ldots, T_{i,n}$
```

The offset and period assignment will be described in section 3.3.6.

Priority assignment conflicts are detected after the derivation of the sequences, and occurs in the cases when two different instances of the same task have to be assigned different priorities in order to ensure the run-time execution of each one of them in the derived target window, and in the right position in the sequence the tasks belongs to. In this case, since a priority assignment involves more than one task, there is typically a choice of which task to split.
In our method, we split tasks that causes offset assignment conflicts into instances *before* deriving the sequences of instances. By that, we reduce the probability of priority assignment conflict eventually caused by the same tasks since the new created tasks will have only one instance during LCM.
We illustrate the issues with an example. Assume that we have the off-line schedule in Figure 3.2 expressing the original constraints of the following taskset: A(5,1), B(10,3), C(20,8), (period, computation time). We assume that we have a precedence constraint between the $(4m+1)^{th}$ instance of A and the $(2m+1)^{th}$ instance of B, $A^{4m+1} \rightarrow B^{2m+1}$, $m = 0, 1, 2, \ldots$, and a precedence constraint between the $(2n+2)^{th}$ instance of B and the $(4n+3)^{th}$ instance of A, $B^{2(n+1)} \rightarrow A^{4n+3}$, $n = 0, 1, 2, \ldots$. The time points

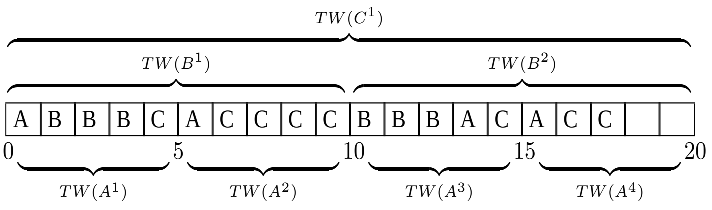


Figure 3.2: Example 1: Off-line Schedule and Target Windows

$t_1 = 0,\ t_2 = 5,\ t_3 = 10,\ t_4 = 15$ marks the beginning of the target windows (Figure 3.2), i.e., $TW_1 = TW(A^1) = [0,5]$, $TW_2 = TW(B^1) = [0,10]$, $TW_3 = TW(C^1) = [0,20]$, $TW_4 = TW(A^2) = [5,10]$, etc. The priority relations (inequalities) between the tasks of each target window, are shown in Figure 3.3. In this example, we can easily see that we do not have any offset assignment conflicts, since the target windows of the instances of the same task begins at the same point in time relative to the task period. According to the sequence $SEQ_1 =< A^1, B^1, C^1 >$, task A must be assigned a higher priority then task B. On the other hand, according to the sequence $SEQ_3 =< B^2, A^3, C^1 >$, task B must be assigned higher priority then A. In this case we have a cycle of priority inequalities that has to be solved:

$$prio(A^1) > prio(B^1) > \ldots > prio(B^2) > prio(A^3)$$

k	t_k	$\left\{ \begin{array}{c} current \\ inst. \end{array} \right\}_{t_k}$	$\left\{ \begin{array}{c} intf. \\ inst. \end{array} \right\}_{t_k}$	SEQ_k	Priority inequalities
1	0	A^1, B^1, C^1	None	A^1, B^1, C^1	$prio(A^1) > prio(B^1)$ $prio(B^1) > prio(C^1)$
2	5	A^2	C^1	A^2, C^1	$prio(A^2) > prio(C^1)$
3	10	A^3, B^2	C^1	B^2, A^3, C^1	$prio(B^2) > prio(A^3)$ $prio(A^3) > prio(C^1)$
4	15	A^4	C^1	A^4, C^1	$prio(A^4) > prio(C^1)$

Figure 3.3: Example 1: Sequences of tasks

We solve this issue by splitting the task with the inconsistent priority assignment into a number of new periodic tasks with different priorities. The instances of the new tasks comprise all instances of the original task. Since a priority assignment conflict involves more than one task, like in our example, there is typically the choice of which task to split. Our goal is to find the splits which yield the smallest number of FPS tasks. In our example, we have to break the chain by splitting either A or B into instances and considering each one of these instances as individual tasks. Depending on the number of instances of A and B during LCM, the choice of the task to be split influences the number of artifact tasks created.
In order to minimize the number of artifact tasks, we create an integer linear programming problem from the derived system of priority inequalities to first identify which instances to split, if any, and to derive priorities for the resulting FPS tasks. The flexibility of the ILP solver allows for simple inclusion of other criteria via goal functions.
In section 3.3.6 we present a complete solution for our example.

3.3.5 ILP Problem Representation

A linear programming (LP) problem consists of a linear goal function in a number of variables and a set of linear inequality relations of the variables. LP solving searches a value assignment for all variables (solution) that optimizes (minimizes or maximizes) the given goal function under the given constraints. If the values of a solution have to be integral the problem is called an integer linear programming (ILP) problem.
The aim of the given attribute assignment problem is to find a task set, i.e., a minimum number of tasks together with their priorities, that fulfills the priority relations of the sequences of the schedule. As mentioned above, each task of

the task set is either one of the original tasks or an artifact task created from one of the instances of an original task selected for splitting.
The problem is translated into an ILP problem, because we are only interested in integral priority assignments and solutions. In the ILP problem the goal function G to be minimized computes the number of tasks to be used in the FPS scheduler

$$G = N + \sum_{i=1}^{N} (k_i - 1) * b_i,$$

where N is the number of tasks in the off-line schedule, k_i is the number of instances of task T_i, and b_i is a binary integral variable that indicates if T_i needs to be split into its instances.
The constraints of the ILP problem reflect the restrictions on the task priorities as imposed by scheduling problem. The priority relations of the original tasks of the off-line schedule form the basis for the ILP constraints. To account for the case of priority conflicts, i.e., when tasks have to be split, the constraints between the original tasks are extended to include the constraints of the artifact tasks. Thus each priority relation $prio(\overset{k}{S}_l) > prio(\overset{k}{S}_{l+1})$ with $\overset{k}{S}_l = T_i^j$ and $\overset{k}{S}_{l+1} = T_p^q$ is translated into an ILP constraint

$$p_i + p_i^j > p_p + p_p^q,$$

where the variables p_i and p_p stand for the priorities of the FPS tasks representing the original tasks T_i and T_p, respectively, and p_i^j, p_p^q stand for the priorities of the artifact tasks T_i^j and T_p^q (in case it is necessary to split the off-line tasks). Although this may look like a constraint between four tasks (T_i, T_i^j, T_p, T_p^q) it is in fact a constraint between two tasks – for each task only its original (T_i resp. T_p) or its artefact tasks (T_i^j resp. T_p^q) can exist in the FPS schedule. A further set of constraints for each off-line task T_i ensure that only either the original tasks or its artefact tasks are assigned valid priorities (greater than 0) by the ILP solver. All other priorities are set to zero.

$$\begin{aligned} p_i &\leq (1 - b_i) * M \\ \forall j : p_i^j &\leq b_i * M \end{aligned}$$

In these constraints M is a large number, larger than the total number of instances in the off-line schedule. The variable b_i for task T_i, which also occurs in the goal function, is the binary variable that indicates if T_i has to be split, i.e., b_i allows only a task or its artifact tasks to assume valid priorities. Since the goal function associates a penalty equal to the number of instances of T_i for each b_i that has to be set to 1, the ILP problem indeed searches for a solution that produces a minimum number of task splits. The constraints on the variables b_i complete the ILP constraints: $b_i \leq 1$.

The solution of the ILP problem yields the total number of tasks as the result of the goal function. The values of the variables p_i and p_i^j for each task represent a priority assignment for tasks and artifact tasks that satisfies the priority relations of the scheduling problem. If $p_i > 0$ or $p_i^j > 0$ then the respective task T_i or T_i^j exists in the FPS schedule and its priority is p_i or p_i^j, respectively. If a variable (p_i or p_i^j) has been assigned the value zero the task/artifact task is not included into the FPS schedule, i.e., $FPS\_tasks = \{T_i : p_i > 0\} \cup \{T_i^j : p_i^j > 0\}$ and for each task in FPS_tasks the priority $prio(T_i)$ is the value of the priority variable of the corresponding task/artifact task, i.e., p_i or p_i^j.

In our example (Example 1), the solution provided by the solver is:

$$
\begin{aligned}
& b_A = b_C = 0 \\
& b_B = 1, \; meaning\ that\ task\ B\ is\ to\ be\ split \\
& p_A = 3 \\
& p_B^1 = 2 \\
& p_B^2 = 4 \\
& p_C = 1
\end{aligned}
$$

3.3.6 Periods and Offsets

Since the priorities of the FP tasks have been assigned by the LP-solver, we can now focus on the assignment of periods and offsets. Now we have a set of tasks with priorities, *FPS_tasks*, produced by the LP-solver, consisting of a subset of the original taskset, $\{orig\_tasks\} \subseteq \{Original\ Tasks\}$, and a set of artifact tasks, $\{art\_tasks\}$, $FPS\_tasks = \{orig\_tasks \cup art\_tasks\}$. Based on the information provided by the LP-solver, we assign periods and offsets to each task in $\overline{T_i} \in \{FPS\_Tasks\}$, in order to ensure the run time execution within their respective target windows, as following:

$\overline{T_i}$	p	c	o	prio
A	5	1	0	3
B1	20	3	0	2
B2	20	3	10	4
C	20	8	0	1

Figure 3.4: Example 1: FPS tasks.

$$\begin{aligned} for \quad & 1 \leq i \leq nr\_of\_tasks\_in(FPS\_tasks) \\ & p(\overline{T_i}) = \frac{LCM}{nr\_of\_instances(\overline{T_i})} \\ & o(\overline{T_i}) = begin(TW(\overline{T_i^1})) \end{aligned}$$

The final set of tasks, derived from the original off-line scheduled tasks in example 1, by performing the steps described in Sections 3.3.4, 3.3.5 and 3.3.6, are illustrated in Figure 3.4. The highest value represents the highest priority.

3.3.7 Discussion

Our method does not introduce artifacts or reduce flexibility unless required by constraints: a set of FPS tasks, scheduled off-line according to FPS, and transformed by our method will execute in the same way as the original FPS tasks. The FPS tasks resulting from our method execute flexibly, unless prevented by reducing target windows for strict predictability for some tasks. Tasks can execute earlier if preceding tasks finish earlier than the assumed worst case execution time, or may even change order of execution, if tasks are not ready to run, provided the priority order is kept.

In some cases, we have to perform additional splits, due to violation of the periodicity in the off-line schedule, which gives different offsets for different instances of the same task. By minimizing the number of artifact tasks, our method minimizes the number of offsets in the system as well, since we don't change offsets unless we have to split tasks. By using ILP, we minimize the number of artifact tasks and, implicitly, offsets.

While our method is capable of deriving FPS tasks for general off-line schedules, the resulting task set and attributes may be awkward in extreme cases, e.g., the off-line schedule includes non periodic patterns or changes execution orders of tasks. Still, our method allows these to be re-enacted with standard FPS scheduling. We are currently investigating the inclusion of trade-offs in our algorithm if more straight forward FPS tasks are desired.
Note that our method allows, e.g., tasks to execute according to earliest deadline first order, although using FPS, by creating artifact tasks with different priorities. Thus, the properties of, in this case, EDF can be exploited in an FPS system. The resulting increase in utilization comes from the periods of the artifact tasks being set to LCM.
Target windows can be derived from off-line schedules directly, without further knowledge about the original timing constraints. In that case, the off-line schedule will be re-enacted exactly by the FPS tasks, providing the same determinism. The resulting assignment, however, will lead to inflexible schedules and inefficient attributes.
We envision the proposed method to be complemented by a run-time enforcement mechanism, such as a watch-dog, to ensure tasks do not overrun their budgets.
On-line tasks with lower priority can easily be added to the fixed priority schedule, while an on-line acceptance test can be performed on the higher priority sporadic or aperiodic tasks.

3.4 Example

We illustrate the ability of the method to deal with tasks with complex constraints, with an example. We assume we have the taskset described in Figure 3.5 and the earliest start times and the deadlines of the off-line tasks, are equal to the start and end of the periods. The precedence relations are:A→B→C; D→E; F→G; F→C, and it takes one time slot to send a message between 2 nodes. Additionally, we want the FPS execution of task A to be fixed between $(est(A) + 2)$ and $(est(A) + 4)$. The off-line schedule and the target windows are illustrated in Figure 3.6.
The priority inequalities between the instances are derived in the same way as in the example presented in Section 3.3 and shown in Figure 3.7. From the inequalities, we can see that we have a number of priority assignment conflicts, i.e., $prio(E^1) > prio(B^1) > prio(C^1) > prio(D^2)$ resulting from SEQ_3 corresponding to $t_3 = 5$, and $prio(D^2) > prio(E^2)$ from SEQ_4, meaning that

$Task$	p	c	Node
A	15	2	0
B	15	1	0
C	15	5	0
D	10	3	0
E	10	2	0
F	15	3	1
G	15	4	1

Figure 3.5: Example 2: Off-line Tasks

we have a cycle of inequalities consisting of $prio(E^1) > prio(B^1) > \ldots > prio(E^2)$. Another cycle of inequalities is given by SEQ_1, SEQ_2, SEQ_3 and SEQ_4: $prio(D^1) > prio(E^1) > \ldots > prio(C^1) > prio(D^2)$. We formulate our goal function in the same way we did in Section 3.3.5, in order to minimize the number of artifact tasks, and LP provide us information about which task(s) to split, in this case D and E, and priorities for the final FPS tasks. Finally we assign offsets and periods to the FPS tasks in order to ensure the execution within the target windows (Figure 3.8). The schedule obtained by scheduling the final taskset by FPS is illustrated in Figure 3.9.

3.5 Conclusions and future work

In this paper we have presented a method that combines off-line schedule construction with fixed priority run-time scheduling. We use off-line schedules and target windows to express complex constraints and predictability for selected tasks, and derive attributes for tasks, such that if applying FPS at run-time, the tasks will execute within the specified target windows and fulfill the original constraints.

Thus, the method solves issues arising from legacy systems, e.g., partition scheduling for avionics applications, allows to handle constraints not covered by FPS feasibility tests, while using standard FPS at runtime. Also it provides for predictable flexibility, i.e., the restricted execution of selected tasks, e.g., for sampling and actuating in control systems, while enabling runtime flexibility for others.

Our method analyzes the off-line schedule and the target windows and derives

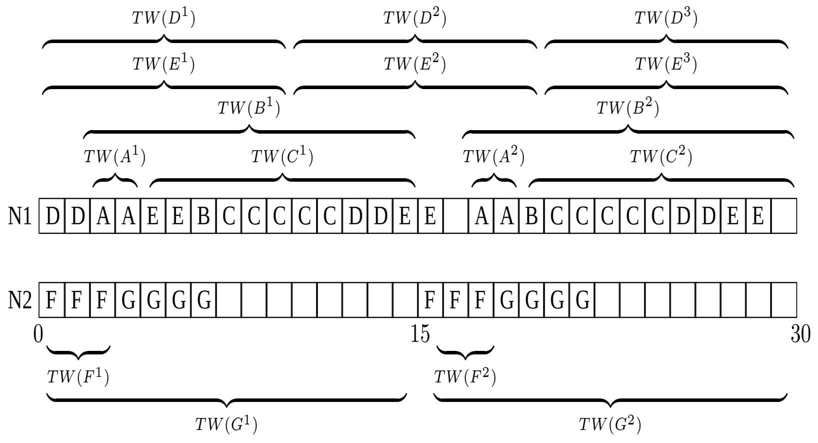


Figure 3.6: Example 2: Off-line Schedule and Target Windows

priority relations between task instances, expressed in a set of inequalities. In certain cases, the method splits tasks into instances, creating artifact tasks, as not all off-line schedules can be expressed directly with FPS. We use standard integer linear programing to solve the priority inequalities and minimize the number of artifact tasks created. Finally, we assign offsets and periods to the task set provided by ILP in order to ensure the correct run-time execution within the derived target windows.

In some cases, we have to perform additional splits, due to a violation of the periodicity in the off-line schedule, which gives different offsets for different instances of the same task. By minimizing the number of artifact tasks, our method minimizes the number of offsets in the system as well. The number of artifact tasks and offsets can be decreased by reducing target windows, if the resulting loss in flexibility is acceptable.

Our method does not introduce artifacts or reduce flexibility unless required by constraints: a set of FPS tasks, scheduled off-line according to FPS, and transformed by our method executes in the same way as the original tasks.

To this point, we have concentrated on reconstructing the off-line schedule. Using the flexibility of the ILP solver, we can add objectives by inclusion in the goal function. Currently, we are investigating providing for trade-offs to reduce the number of preemption and priority levels.

Furthermore, we assume that all task dependencies have been resolved off-line.

t_k	Node	$\left\{\begin{array}{c} current \\ inst. \end{array}\right\}_{t_k}$	$\left\{\begin{array}{c} intf. \\ inst. \end{array}\right\}_{t_k}$	SEQ_k	inequalities
0	0	D^1, E^1	None	D^1, E^1	$prio(D^1) > prio(E^1)$
	1	F^1, G^1	None	F^1, G^1	$prio(F^1) > prio(G^1)$
2	0	A^1, B^1	E^1	A^1, E^1, B^1	$prio(A^1) > prio(E^1)$
					$prio(E^1) > prio(B^1)$
4	0	C^1	E^1, B^1	E^1, B^1, C^1	$prio(E^1) > prio(B^1)$
					$prio(B^1) > prio(C^1)$
10	0	D^2, E^2	C^1	C^1, D^2, E^2	$prio(C^1) > prio(D^2)$
					$prio(D^2) > prio(E^2)$
15	0	None	E^2	E^2	-
	1	F^2, G^2	None	F^2, G^2	$prio(F^2) > prio(G^2)$
17	0	A^2, B^2	None	A^2, B^2	$prio(A^2) > prio(B^2)$
20	0	C^2, D^3, E^3	None	C^2, D^3, E^3	$prio(C^2) > prio(D^3)$
					$prio(D^3) > prio(E^3)$

Figure 3.7: Example 2: Inequalities

Future work will address the issue of task relations at run-time as well.

3.6 Acknowledgements

The authors wish to express their gratitude to Iain Bate, Alan Burns, Krithi Ramamritham, and Guillem Bernat for useful discussions and to the reviewers for their helpful comments on the paper.

3.7 Proofs

We prove the method in two steps. First, we prove that FPS tasks will meet their deadlines, and second, that we preserve the order of execution enforced by the task constraints expressed in the off-line schedule.

3.7.1 Proof 1

Theorem: Any instance T_i^j of any task T_i, produced by the method described in Section 3.3, execute at run-time within its derived target window, if scheduled by FPS together with all other FP tasks. In fact it complete its execution before it's off-line scheduled finishing time, $R(T_i^j) \leq ft(T_i^j)$.

T_i	p	c	o	prio
A	15	2	2	5 (highest)
B	15	1	2	3
C	15	5	4	2
D1	30	2	0	5
D2	30	2	10	1
D3	30	2	20	1
E1	30	2	0	4
E2	30	2	10	0
E3	30	2	20	0
F	15	3	0	1
G	15	4	0	0 (lowest)

Figure 3.8: Example 2: FPS Tasks

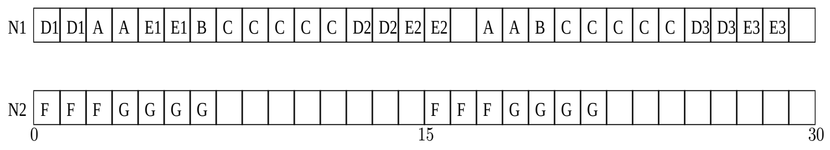

N1
D1 D1 A A E1 E1 B C C C C C D2 D2 E2 E2 A A B C C C C C D3 D3 E3 E3
N2
F F F G G G G F F F G G G G
0
15
30

Figure 3.9: Example 2: FP Schedule

Proof: We want to prove that
$\forall t_k,\ SEQ_k =< \overset{k}{S}_1, \overset{k}{S}_2, \ldots, last(SEQ_k) >$:

$$\begin{aligned} R(\overset{k}{S}_i) \quad &\leq \quad ft(\overset{k}{S}_i) \leq dl(\overset{k}{S}_i) = end(TW(T_i^j)), \\ &\qquad \forall i \in [1, nr\_of\_tasks\_in\_SEQ_k] \end{aligned}$$

where $R(\overset{k}{S}_i) = est(\overset{k}{S}_i) + c(\overset{k}{S}_i) + \sum_{\forall j \in hp(i)} c(T_j)$.
T_j is a task instance belonging to either SEQ_k or $SEQ_{k+n},\ n \geq 1$. All task instances with earliest start times less or equal to t_k are included in SEQ_k as either $current\_instances_{t_k}$ or $interfering\_instances_{tk}$. However, there might be an interference from task instances belonging to 'later' sequences SEQ_{k+n} that are not taken into account when deriving the priority inequalities

corresponding to SEQ_k.
What do we know?

1. $\forall\, TW_n,\ \forall\, T_i^j \in TW_n,$
   $est(T_i^j) = begin(TW_n)\ and\ dl(T_i^j) = end(TW_n)$

2. $\forall SEQ_k,\ \forall\, i \in [1, nr\_of\_tasks\_in\_SEQ_k],$
   $t_k + c(\overset{k}{S}_i) + c(\overset{k}{S}_{i-1}) + \ldots + c(\overset{k}{S}_1) \leq ft(\overset{end}{S}_i),$
   (from the off-line schedule)

3. $\forall SEQ_k,$
   $prio(\overset{k}{S}_1) > prio(\overset{k}{S}_2) > \ldots > prio(last(SEQ_k)),$
   (given by ILP)

4. $\forall i \in [1, nr\_of\_tasks\_in\_SEQ_k],$
   $ft(\overset{k}{S}_i) \leq end(TW(\overset{k}{S}_i)),$
   (given by the off-line schedule)

We use induction.

(a) We prove it for the 'last' sequence in the off-line schedule, SEQ_{end} corresponding to the time t_{end} since in this case we don't have interference from task instances belonging to 'later' sequences:

$$\forall\, i,\ \overset{end}{S}_i \in SEQ_{end},\ R(\overset{end}{S}_i) \leq ft(\overset{end}{S}_i)$$

Proof:

$$SEQ_{end} = <\overset{end}{S}_1,\ \overset{end}{S}_2, \ldots, last(SEQ_{end})>$$

$\forall i \in [1, nr\_of\_instances\_in\_SEQ_{end}].$
From (2) $\Rightarrow t_{end} + c(\overset{end}{S}_i) + c(\overset{end}{S}_{i-1}) + \ldots + c(\overset{end}{S}_1) \leq ft(\overset{end}{S}_i)$
From (3): $c(\overset{end}{S}_1) + c(\overset{end}{S}_2) + \ldots + c(\overset{end}{S}_{i-1}) = \sum_{\forall j \in hp(i)} c(T_j)$, since there is no interference from task instances belonging to 'later' sequences.
$\Rightarrow t_{end} + c(\overset{end}{S}_i) + \sum_{\forall j \in hp(i)} c(T_j) \leq ft(\overset{end}{S}_i)$
Additionally we know that $est(\overset{end}{S}_i) \leq t_{end}$ (from the definition of the sequences):

$est(\overset{end}{S}_i) + c(\overset{end}{S}_i) + \sum_{\forall j \in hp(i)} c(T_j) \leq ft(\overset{end}{S}_i)$

$\Rightarrow \underline{R(\overset{end}{S}_i) \leq ft(\overset{end}{S}_i)}$, meaning that $\overset{end}{S}_i$ complete its execution before its off-line scheduled finishing time, and implicitly, before the end of its target window (from (1) and (4)).

(b) We assume $SEQ_k =< \overset{k}{S}_1, \ldots, last(SEQ_k) > \Rightarrow \; R(\overset{k}{S}_i) \leq ft(\overset{k}{S}_i)$, $\forall i \in [1, nr\_of\_inst.\_in\_SEQ_k]$

We prove SEQ_{k-1}:

$$\forall\, i,\; \overset{k-1}{S}_i \in SEQ_{k-1},\; R(\overset{k-1}{S}_i) \leq ft(\overset{k-1}{S}_i)$$

We have two cases:

Case1: $SEQ_k \cap SEQ_{k-1} = \varnothing \Rightarrow$ No interference $\Rightarrow$ same proof as in (a).

Case2: $SEQ_k \cap SEQ_{k-1} \neq \varnothing$, $SEQ_k \cap SEQ_{k-1} = common\_tasks$, where:

$common\_tasks = \{T^j \mid T^j \in SEQ_k \wedge T^j \in SEQ_{k-1} \wedge st(T^1) <$
$< st(T^2) < \ldots < st(T^m)\} =< T^1, T^2, ..., T^m >$

Then:

$SEQ_{k-1} =< \overset{k-1}{S}_1, \ldots, \overset{k-1}{S}_n > \cup \underbrace{< \overset{k-1}{S}_{n+1}, \ldots, last(SEQ_{k-1}) >}_{\text{common\_tasks}} =$

$=< \overset{k-1}{S}_1, \ldots, \overset{k-1}{S}_n, \underbrace{\overset{k-1}{S}_{n+1}, \ldots, last(SEQ_{k-1})}_{\text{common\_tasks}} >$

We know that:

$\forall T_i \in common\_tasks \subseteq SEQ_k \Rightarrow$

$R(T^i) \leq ft(T^i)$, (from the assumption), and:

$prio(\overset{k-1}{S}_i) > prio(T^j)$, $\forall i \in [1, n]$, $\forall T^j \in \{common\_tasks\}$, (from the priority assignment and (3)).

Then, $\forall i \in [1, n] : (from(2))$

$R(\overset{k-1}{S}_i) = \underbrace{t_{k-1}}_{\geq est(\overset{k-1}{S}_i)} + c(\overset{k-1}{S}_i) + \underbrace{c(\overset{k-1}{S}_{i-1}) + \ldots + c(\overset{k-1}{S}_1)}_{\sum_{\forall j \in hp(i)} c(T_j)} \leq ft(\overset{k-1}{S}_i)$

Hence:

$$R(\overset{k-1}{S}_i) = est(\overset{k-1}{S}_i) + c(\overset{k-1}{S}_i) + \sum_{\forall j \in hp(i)} c(T_j) \leq ft(\overset{k-1}{S}_i). \square$$

3.7.2 Proof 2

Theorem: If there is a precedence relation expressed in the original task constraints and/or the off-line schedule between any two instances T_m^i, T_n^j of any two tasks $T_m,\ T_n,\ T_m^i \rightarrow T_n^j$, then T_m^i execute before T_n^j when scheduling the FPS-task produced by the method described in 3.3 by FPS, assuming that the precedence requirement is fulfilled in the off-line schedule.

Proof: We prove that if there is an overlapping, in terms of time, between the target windows of the two instances, $TW(T_m^i)\ and\ TW(T_n^j)$, then T_m^i is assigned a higher priority then T_n^j, $prio(T_m^i) > prio(T_n^j)$. Additionally, we know that any instance of any FP task execute inside its target window if scheduled by FPS (3.7.1) and, if there is an overlapping between the two target windows, then the target window of T_m^i must begin before the beginning of the target window of T_n^j:

$$begin(TW(T_m^i)) \leq begin(TW(T_n^j))$$

We have two cases:

1. $begin(TW(T_m^i)) = begin(TW(T_n^j)) = t_k$
   then $T_m^i,\ T_n^j \in current\_instances_{t_k} \in SEQ_k$, and the off-line execution of T_m^i is before the off-line execution of T_n^j (from the off-line schedule). Then: $prio(T_m^i) > prio(T_n^j)$, since $prio(\overset{k}{S}_1) > prio(\overset{k}{S}_2) \ldots > prio(last(SEQ_k))$, where the off-line execution of $\overset{k}{S}_i$ is before the off-line execution of $\overset{k}{S}_{i+1},\ \forall i \in [1, nr\_of\_instances\_inSEQ_k]$.

2. $begin(TW(T_m^i)) = t_k < begin(TW(T_n^j)) = t_p$

   (a) $ft(T_m^i) \leq begin(TW(T_n^j)) = t_p$
   no interference. T_m^i finish its execution before the start of the target window of T_n^j.

   (b) $ft(T_m^i) > begin(TW(T_n^j)) = t_p$, then:
   $T_m^i \in \{interfering\_instances\}_{t_p} \in SEQ_p$,
   $T_n^j \in \{current\_instances\}_{t_p} \in SEQ_p$, and T_m^i is off-line scheduled to execute before T_n^j (from the off-line schedule). Then:
   $prio(T_m^i) > prio(T_n^j)$, since $prio(\overset{p}{S}_1) > prio(\overset{p}{S}_2) > \ldots >$

$> prio(last(SEQ_p))$, where $\overset{p}{S}_i$ is off-line scheduled to execute before $\overset{p}{S}_{i+1}$, $\forall i \in [1, nr\_of\_instances\_in(SEQ_p)]$.$\square$

Bibliography

[1] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols: an Approach to Real-Time Synchronization. *IEEE Transactions on Computer*, 39(9):1175–1185, Sept 1990.

[2] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Tasks. *The Journal of Real-Time Systems*, 1989.

[3] K. Tindell. Adding Time Offsets to Schedulability Analysis. Technical report, Departament of Computer Science, University of York, January 1994.

[4] J.C. Palencia and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proceedings of 19th IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[5] M. Gonzalez Harbour and J.P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Varying Execution Priority. In *Proceedings of Real-Time Systems Symposium*, pages 116–128, Dec. 1991.

[6] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. ARINC Scheduling: Problem Definition. In *Proceedings of Real-Time Systems Symposium*, pages 165–169, 1994.

[7] N.C. Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times. Technical report, Departament of Computer Science, University of York, 1991.

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

[9] D. Seto, J.P. Lehoczky, and L. Sha. Task Period Selection and Schedulability in Real-Time Systems. In *Proceedings of Real-Time Systems Symposium*, pages 188–198, 1998.

[10] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Austria, Apr. 1994.

[11] R. Dobrin, Y. Özdemir, and G. Fohler. Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-line Schedules. In *Conference on Real-Time Computing Systems and Applications, Korea*, December 2000.

Chapter 4

Paper C: Implementing Off-line Message Scheduling on Controller Area Network (CAN)

Radu Dobrin and Gerhard Fohler
In Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation, Antibes, France, 2001.

Abstract

Controller Area Network (CAN) is widely used in a number of industrial applications. The message scheduling on CAN is based on identifiers (ID) assigned to the messages according to their priorities. Fixed priority scheduling protocols (FPS) have a number of advantages, some of them being the efficient exploitation of the channel bandwidth, small overhead and the simple implementation. On the other hand, a number of industrial applications demand temporal properties that are typically achieved by using off-line scheduling. In addition, complex constraints can be solved off-line, such as distribution, end-to-end deadlines, precedence, jitter, or instance separation, but this scheduling strategy is not suitable for CAN.

In this paper we present a method that shows how off-line scheduled messages can be scheduled on CAN. The paper assumes that a schedule, for a set of tasks transmitting messages on CAN, has been constructed off-line. It presents a method that analyzes the off-line schedule and derives a set of periodic messages with fixed priorities, which can be scheduled on CAN. Based on the information provided by the off-line schedule, the method derives inequality relations between the priorities of the messages under FPS. In case the priority relations of the messages are not solvable, we split some messages into a number of artifacts, to obtain a new set of messages with consistent priorities. We use integer linear programming to minimize the final number of messages.

4.1 Introduction

Controller Area Network (CAN), has gained wider acceptance as a standard in a large number of industrial applications. The priority based message scheduling used in CAN has a number of advantages, some of the most important being the efficient bandwidth utilization, flexibility, simple implementation and small overhead. Early results on message scheduling on CAN have been presented in [1] and [2], in which the authors focused on fixed priority scheduling based on work presented in [3] and [4]. Later on, Zuberi [5] showed that static priority scheduling is not always the most suitable strategy. Earliest Deadline (EDF) can prove significantly better then fixed priority scheduling [6].

Off-line scheduling for time triggered systems, on the other hand, provides determinism [7], [8], and, additionally, complex constraints can be solved off-line, but this scheduling strategy is not suitable for CAN.

In this paper we present a method that transforms off-line scheduled transmission schemes into sets of messages that can be scheduled on CAN. It assumes a schedule has been constructed for a set of off-line scheduled messages to meet their complex constraints. Our method takes the off-line schedule with derived time intervals in which messages must be transmitted, from now on referred to as *Target Windows*, and assigns FPS attributes, (i.e., priorities) to the messages. It then, provides information about periods and offsets the messages have to be sent with by the sending nodes, such that the message transmission at run-time matches the off-line schedule. It does so by deriving priority inequalities, which are then resolved by integer linear programming.

FPS cannot reconstruct all schedules with periodic messages with the same priorities for all instances (invocations) directly. The constraints expressed via the off-line schedule may require different message sequences for invocations of the same message, as, e.g., by earliest deadline first, leading to inconsistent priority assignment. This phenomenon can be expressed as a cycle of inequalities. Our algorithm detects such situations, and circumvents the problem by splitting a message into its invocations. Then, the algorithm assigns different priorities to the newly generated ”artifact” messages, the former invocations.

Key issues in resolving the priority conflicts are the number of artifact messages created. Depending on where a priority conflict circle is ”broken”, the number may vary, depending on the periods of the split messages. Our algorithm minimizes the number of artifact messages by solving the priority inequalities with integer linear programming (ILP).

Priority assignment for FPS tasks has, for example, been studied in [9], [10] and [11]. [12] study the derivation of task attributes to meet a overall con-

straints, e.g., demanded by control performance. Instead of specific requirements, our algorithm takes an entire off-line schedule and all message requirements to determine message attributes. A method to transform off-line schedules into earliest deadline first tasks has been presented in [13]. A related paper [14] deals with priority assignment for off-line CPU scheduled tasks. It uses a constructive, heuristic approach, potentially creating large numbers of artifacts, while the approach presented here presents a general algorithm applied to message scheduling on CAN using ILP for optimum solutions.
The rest of the paper is organized as follows. In section 4.2 we give a brief overview of message scheduling on CAN. The method we are proposing is presented in section 4.3 and, then, illustrated with an example in section 4.4. Section 4.5 concludes the paper.

4.2 Controller Area Network (CAN) and message scheduling

CAN consists of the physical and data link layers. Each CAN frame consist of seven fields. In this paper we focus on the identifier field (ID). The identifier field may have two lengths: 11 bits, which is the standard format, and 29 bits, the extended format, and it controls message addressing and bus arbitration . In this paper we focus only on the former since the nodes can set message filters in order to receive only the identifiers they are interested in.
The nodes are connected via a wired OR (or wired AND) CAN bus. The time axis is divided in slots which must be larger or equal to the time it takes the signal to propagate back and forth the bus, $t = \frac{2L}{V}$, where L is the bus length and V is the propagation speed of the signal. When a node has to send a message, it calculates the message ID which may be based on the priority of the message. The message ID must be unique in order to prevent eventually ties. The message is then sent to the bus interface chips, which, further on, write the message ID on the bus, bit by bit, whenever the bus is idle at the beginning of a time slot. After writing a bit on the bus, the chip waits for the signal to propagate along the bus and, then, reads the bus. If the bit read is different from the bit sent, then there is another message on the bus with a higher priority, and the sending node aborts the transmission. Otherwise the node gets the right to send the message without being pre-empted.
In our paper, we assume a schedule has been constructed for a set of off-line messages. The proposed method transforms the off-line scheduled messages into as set of messages suitable for priority-based CAN message scheduling.

Since we do not want to assign any other attributes then priorities to the message ID's (e.g., periods and offsets), due to the restrictions enforced by the ID format, we rather provide information about attributes that have to be assigned by the programmer to the sending nodes and messages (periods, offsets and priorities) in order to ensure the run-time transmission of the messages according to the specifications expressed in the off-line schedule. Furthermore, we assume that the nodes are clock synchronized and the off-line schedule has been constructed by taking into account the increased bandwidth consumption due to the exchange of messages required by the time synchronization method.

4.3 Attribute assignment algorithm

4.3.1 Overview

Figure 4.1 gives an overview of the algorithm.
1) and 2) Initially, the off-line schedule table for a set of messages with constraints, is given.
3) Target windows for each invocation of each message are derived from the original message constraints and the off-line schedule.
4) Sequences are now straightforward to derive from the target windows and the transmission order expressed in the off-line schedule.
5) The analysis of each sequence provides a set of inequalities between priorities of invocations of different messages.
6) We use integer linear programming to solve the system of inequalities and the result is the final set of messages with fixed priorities.
Off-line schedule: The input to our method is the off-line schedule expressing the constraints specified for the messages to be sent on CAN. The schedule is usually created up to the least common multiple, LCM, of all message periods. We have $LCM/T(M_i)$ invocations of each message M_i with period $T(M_i)$ in the off-line schedule.
The off-line scheduler resolves constraints such as distribution, end-to-end deadlines, precedence, etc, and creates scheduling tables for each node in the system, listing start- and finishing-times of all message invocations. These scheduling tables are more fixed than required by the original constraints, so we can replace the exact sending- and receiving-times of messages with target windows, taking the original constraints into account.
Target windows $(TW(M_i^j))$ of each invocation M_i^j of each message M_i,are derived from the off-line schedule and the original constraints transformed into

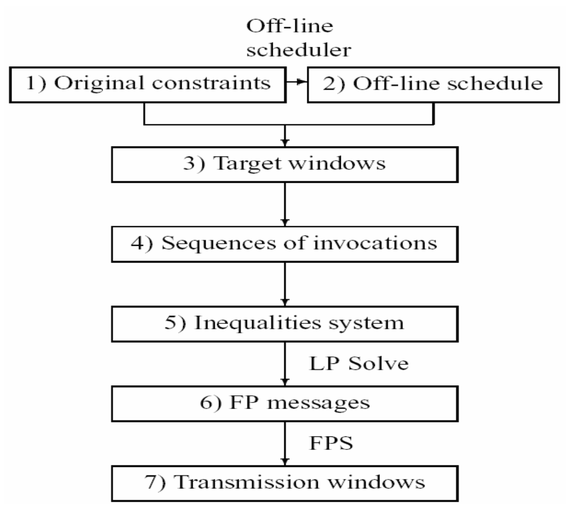


Figure 4.1: Algorithm overview.

earliest start times and deadlines.

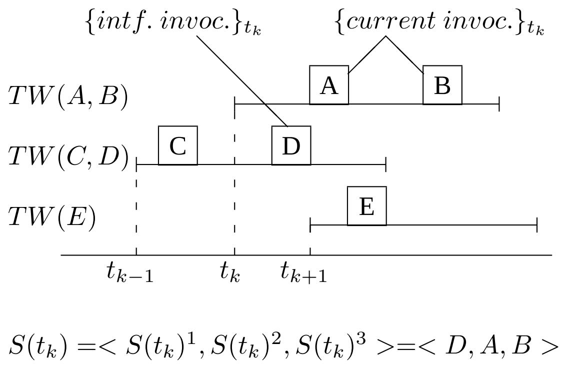
$$TW(M_i^j) = [t_m, t_n]$$

where

$$t_m = begin(TW(M_i^j))\ and\ t_n = end(TW(M_i^j))$$

The *earliest transmission start time*, $est(M_i^j)$, of an invocation M_i^j of a message M_i, is provided by the message constraints expressed in the off-line schedule. The *scheduled receiving time*, $srt(M_i^j)$, of an invocation M_i^j of a message M_i, is the time when M_i^j is received by the receiving node according to the off-line schedule. The *scheduled transmission start time*, $start(T_i^j)$, of an invocation M_i^j of a message M_i, is the time when M_i^j is sent on the bus, according to the off-line schedule.

A **sequence** $S(t_k)$ consists of invocations of messages M_i^j ordered by increasing scheduled transmission start times according to the off-line schedule. A sequence may contain invocations M_i^j such that $begin(TW(M_i^j)) = est(M_i^j) = t_k$, current invocations of $TW(M_i^j)$, and invocations M_p^q from overlapping target windows such that $est(M_p^q) < t_k$ and $start(M_p^q) > t_k$, interfering invocations of $TW(M_i^j)$. Additionally: $first(S(t_k)) = S(t_k)^1$ = first message invocation in the sequence $S(t_k)$, and $last(S(t_k)) = S(t_k)^N$ = last message invocation in $S(t_k)$. The derivation of a sequence corresponding to a time t_k is illustrated in figure 4.2. Note that, in figure 4.2, message 'E' is not included in the sequence corresponding to the time t_k, since it's earliest transmission start time is greater then t_k. 'E' will be instead a *current invocation* in the sequence corresponding to the time t_{k+1}.



$\{intf.\ invoc.\}_{t_k}$
$\{current\ invoc.\}_{t_k}$
$TW(A,B)$
A
B
$TW(C,D)$
C
D
$TW(E)$
E
t_{k-1}
t_k
t_{k+1}
$S(t_k) =< S(t_k)^1, S(t_k)^2, S(t_k)^3 >=< D, A, B >$

Figure 4.2: Sequence of messages.

We refer to an *transmission window*, $W_{trans}(M_i^j)$, of an invocation M_i^j of a message M_i, as the time interval in which M_i^j will be sent and received *at runtime*. We want to find fixed priorities, offsets, and deadlines such that the transmission window of each message invocation M_i^j, $W_{trans}(M_i^j)$, will be contained within the respective target window $TW(M_i^j)$, and transmission order specified off-line, kept.

4.3.2 Priority inequalities

Our algorithm derives relations (inequalities of priorities) among the invocations of the messages by traversing the off-line schedule represented by the series of target windows in increasing order of time. It determines priority inequalities between invocations according to the sequences $S(t_k)$ associated with target windows, such that:

$$P(S(t_k)^1) > P(S(t_k)^2) > \ldots > P(S(t_k)^N)$$

where

$$S(t_k)^1 = first(S(t_k))\ and\ S(t_k)^N = last(S(t_k))$$

Note that the inequalities have to take into account relations between priorities of invocations of the current target window and possibly interfering target windows.

4.3.3 Attribute assignment - conflicts

Our goal is to provide messages with fixed priorities periodically sent on the bus. It may happen, however, that we have to assign different priorities or/and offsets to different invocations of the same message in order to reenact the off-line schedule at run time. These cases cannot be expressed directly with fixed priorities and fixed offsets and are the sources for *offset assignment conflicts* or *priority assignment conflicts*. In both cases, we split the conflicting message into artifacts, such that, further on, each artifact will be considered an independent message, invoked only once during LCM. Thus we create a number of artifact messages equal to the number of invocations during LCM of the message to be split minus one (since the original message will be replaced by a number of messages equal to the number of its invocations).

By *offset assignment conflict* we mean that different invocations of the same message may have to be invoked at different points in time, relative to the sending task period, in order to ensure the run-time transmission of each one of them in the derived target window.

```
for $1 \leq i \leq nr\_of\_off - line\_sched\_messages$
    for $1 \leq j \leq n,\ where\ n = LCM/T(M_i)$
        if $begin(TW(M_i^j)) - (j-1) * T(M_i) \neq$
          $\neq begin(TW(M_i^{j+1})) - j * T(M_i)$,
          (where $T(M_i)$ is the period of the message $M_i$)
        then split $M_i$ into $M_{i,1}, M_{i,2}, \ldots, M_{i,n}$
```

By splitting M_i, we remove it from the original set of messages, *orig_messages*, and we insert $M_{i,1}$, $M_{i,2}$, ..., $M_{i,n}$ into *orig_messages*.

Priority assignment conflicts are detected after the derivation of the sequences, and occurs in the cases when two different invocations of the same task may have to be sent with different priorities in order to ensure the run-time transmission of each one of them in the derived target window, and in the right position in the sequence the message belongs to. In this case, since a priority assignment involves more than one message, there is typically a choice of which message to split.

In our method, we split messages that causes offset assignment conflicts into artifacts *before* deriving the sequences of invocations. By that, we reduce the probability of priority assignment conflict eventually caused by the same messages since the new created messages will be invoked only once during LCM.

4.3.4 Minimizing the final number of messages

In order to minimize the number of artifact messages, we create an integer linear programming problem from the derived system of priority inequalities to first identify which messages to split, if any, and to derive priorities for the resulting fixed priority messages. We aim for the minimum amount of artifact messages, and implicitly priorities, due to the limited amount of priorities available when scheduling messages on CAN.
The inequalities obtained from the execution order within the sequences, may form a circular chain of priority relations between messages invocations, e.g.,

$$P(M_i^j) > P(M_m^n) > \ldots > P(M_i^{j+k}) > \ldots > P(M_m^{n+q})$$

We use a higher value to represent a higher priority.
In this case we cannot assign the same priority to both invocations j and $(j+k)$ of M_i, nor to invocations n and $(n+q)$ of M_m. We have to break the chain by splitting either M_i or M_m into artifacts and considering each one of them as individual messages, which will result in a larger number of messages compared to the number of original off-line scheduled messages. We formulate a goal function for an integer linear programming solver to identify the minimum amount of messages with fixed priorities.

$$G = \#final\_msgs = \#orig\_msgs + \sum_{i=1}^{N}(|M_i| - 1) * b_i$$

where $\#final\_msgs$ is the number of final messages, $\#orig\_msgs$ is the number of original messages, $|M_i|$ = number of invocations of M_i in LCM and b_i is a boolean variable associated to each message M_i, $b_i \in \{0, 1\}$. b_i = 1 means that M_i needs to be split into $|M_i|$ messages. Additionally, the solver provides priority values for the messages (split or non-split).

At this point we have a set of messages with fixed priorities, *final_msgs*, produced by the LP-solver. Finally, we assign periods and offsets to each message (i.e., provide information about when the messages are to be sent by the controller interface) provided by the LP-solver in order to ensure the run time transmission of the messages within their respective target windows, as following:

$$\begin{aligned} for \quad & 1 \leq i \leq \#(final\_msgs) \\ & T(M_i) = \frac{LCM}{nr\_of\_invocations(M_i)} \\ & offset(M_i) = begin(TW(M_i^1)) \end{aligned}$$

4.4 Example

We illustrate the method with an example. Assume that we have the set of messages, sent from two nodes, shown in figure 4.3. Additionally we assume

Message	Node	message size	period (T)
A	1	1	5
B	2	3	10
C	1	4	20

Figure 4.3: Original set of messages

that we have a precedence constraint between the $(4m + 1)^{th}$ invocation of A and the $(2m + 1)^{th}$ invocation of B,

$$A^{4m+1} \rightarrow B^{2m+1}$$

where $m = 0, 1, 2, \ldots$, and a precedence constraint between the $(2n+2)^{th}$ invocation of B and the $(4n+3)^{th}$ invocation of A,

$$B^{2(n+1)} \rightarrow A^{4n+3}$$

where $n = 0, 1, 2, \ldots$.
The off-line schedule for the messages and the derived target windows are illustrated in figure 4.4.

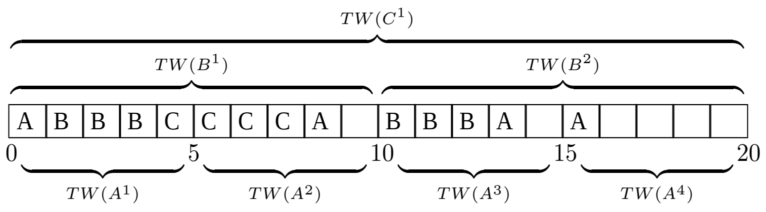


Figure 4.4: Off-line Scheduled Messages and Target Windows

The derivation of the inequalities, performed as described in section 4.3.2, is illustrated in the figure 4.5.

t_k	Message invocations	$S(t_k)$	inequalities
0	A^1, B^1, C^1	A^1, B^1, C^1	$P(A^1) > P(B^1)$ $P(B^1) > P(C^1)$
5	A^2	A^2	
10	A^3, B^2	B^2, A^3	$P(B^2) > P(A^3)$
15	A^4	A^4	

Figure 4.5: Inequalities

At time t_k=10, we have the inequality $P(B^2)>P(A^3)$ added to the relations obtained at t_1=0 and t_2=5: $P(A^1)>P(B^1)$, and $P(A^2)>P(C^2)$. That gives a circular chain of priorities that must be solved:

$$P(A^1) > P(B^1) > \ldots > P(B^2) > P(A^3)$$

In this case, we can either choose to split message A, or message B.
Splitting B will create two artifact messages, while splitting A will result in four. The integer linear programming solver with the goal function described in section 2.3 provides the solution:

- $b_A = b_C = 0$
- $b_B = 1$, meaning message B is to be split,
- $p_A = 2$
- $p_{B^1} = 3$
- $p_{B^2} = 1$
- $p_C = 4$
- $\#final\_msg = \#orig\_msg + \sum_{i=1}^{N}(|T_i| - 1) * b_i = 4$

After assigning offsets and periods to the artifact messages (i.e., B1 and B2), as described in section 4.3.3, the final set of messages is shown in figure 4.6. The lowest value represents the highest priority.

msg	node	msg size	T	offset	dl	prio
A	1	1	5	0	5	2
B1	2	3	20	0	10	3
B2	2	3	20	10	20	1
C	1	4	20	0	20	4

Figure 4.6: FP messages

4.5 Conclusions and future work

In this paper we have presented a method that shows how off-line scheduled messages can be scheduled on CAN. We use off-line schedules and target windows to express complex constraints and predictability for selected messages. We, then, derive attributes for the off-line scheduled messages, such that the messages will be transmitted within the specified target windows while fulfilling the original constraints, when scheduled on CAN.
Our method analyzes the off-line transmission scheme and the target windows and derives priority relations between the invocations of the messages, expressed in a set of inequalities. In certain cases, the method splits messages into instances, creating artifact messages with fixed priorities, as not all off-line schedules can be expressed directly with FPS. We use standard integer linear programing to solve the priority inequalities and minimize the number

of artifact messages created. Finally, offsets and periods can be assigned in the implementation, to the set of sending tasks provided by ILP in order to ensure the run-time transmission of the messages within the derived target windows.
In same cases, we may perform additional splits, due to violation of the periodicity in the off-line schedule, which gives different offsets at which different instances of the same message have to be sent. The number of artifact messages caused by offset assignment conflicts, could be decreased by reducing target windows, if the resulting loss in flexibility is acceptable. The priority inversion phenomenon, due to the non-preemption of message transmission, can be solved by modifying the start of the target windows of the messages with precedence relations considering the precision achieved in the global time synchronization.
Our method does not introduce artifacts or reduce flexibility unless required by constraints: the fixed priority messages provided by our method, with input consisting of a set of messages with fixed priorities, scheduled off-line according to FPS, will be transmitted within the derived target windows and in the off-line specified transmission order.
To this point, we have concentrated on reconstructing the off-line scheduled messages. Using the flexibility of the ILP solver, we can add objectives by inclusion in the goal function. Future work will address the issue of message relations at run-time as well.

4.6 Acknowledgements

The authors wish to express their gratitude to the reviewers for their useful comments on the paper.

Bibliography

[1] K. Tindell, H. Hansson, and A.J. Wellings. Analizing Real-Time Communications: Controller Area Network (CAN). In *Proceedings of Real-Time Systems Symposium*, pages 259–263, Dec. 1994.

[2] K. Tindell, A. Burns, and A.J. Wellings. Calculating Controller Area Network (CAN) message response times. *Contr. Eng. Practice*, 3(8):1163–1169, 1995.

[3] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment. *Journ. of the ACM, 20, 1*, Jan. 1973.

[4] J.Y-T. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2(4):237–250, Dec. 1982.

[5] K.M. Zuberi and K.G. Shin. Scheduling Messages on Controller Area Network for Real-Time CIM Applications. *IEEE Transactions on Robotics and Automation*, 13(2):310–314, Apr. 1997.

[6] M. Di Natale. Scheduling the CAN Bus with Earliest Deadline Techniques. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pages 259–268, Dec 2000.

[7] H. Kopetz. Why Time-Triggered Architectures will Succeed in Large Hard Real-Time Systems. In *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 2–9, 1995.

[8] H. Kopetz and G. Grunsteidl. TTP - a Protocol for Fault-Tolerant Real-Time Systems. *Computer*, 27(1):14–23, 1994.

[9] N.C. Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times. Technical report, Departament of Computer Science, University of York, 1991.

[10] N. Audsley, K. Tindell, and A. Burns. The End Of The Line For Static Cyclic Scheduling? In *Proceedings of the Fifth Euromicro Workshop on Real-Time Systems*, pages 36–41, 1993.

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

[12] D. Seto, J.P. Lehoczky, and L. Sha. Task Period Selection and Schedulability in Real-Time Systems. In *Proceedings of Real-Time Systems Symposium*, pages 188–198, 1998.

[13] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Austria, Apr. 1994.

[14] R. Dobrin, Y. Özdemir, and G. Fohler. Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-line Schedules. In *Conference on Real-Time Computing Systems and Applications, Korea*, December 2000.