TRITA-MMK 2002:6
ISSN 1400-1179
ISRN KTH/MMK/R--02/6-SE

Enforcing Temporal Constraints in Embedded Control Systems

By

Kristian Sandström

Stockholm 2002

Doctoral Thesis
Mechatronics Laboratory
Department of Machine Design
Royal Institute of Technology, KTH
S-100 44 Stockholm Sweden

Mechatronics Lab **Department of Machine Design** **Royal Institute of Technology** **S-100 44 Stockholm, Sweden**	TRITA-MMK 2002:6 ISSN 1400-1179 ISRN KTH/MMK/R—02/6—SE
	Document type Doctoral Thesis — *Date* 2002-04-19
Author(s) Kristian Sandström	*Supervisor(s)* Christer Norström, Martin Törngren, and Jan Wikander
Title Enforcing Temporal Constraints in Embedded Control Systems	*Sponsor(s)* Mälardalens University, KK-foundation

Abstract

Computer control systems are embedded in a large and growing group of products, ranging from consumer entertainment products to large airliners. Products such as automotive vehicles, aircraft, and industrial robots are equipped with advanced computer control systems and have high requirements of reliable and safe operation. A common property of these systems is that the computer systems are becoming increasingly more complex due to the inclusion of more functionality. At the same time, the product cycles are becoming shorter leading to requirements of shorter time to market. To meet this challenging task, the development of computer control systems must be a well-defined and controlled engineering process. One important part in reaching this goal is to find methods for dealing with the complexity of computer systems.

A computer control system is typically realized by a set of concurrent activities with inter-dependencies that have to meet a set of pre defined temporal constraints. Because of this, it is difficult to know in advance if the implementation of a design will meet its temporal constraints. Furthermore, it is hard to foresee the consequence of introducing alterations or additional functionality in a system. For a method to be useful it also has to capture all the relevant aspects of the application domain, and in the domain of embedded control systems this includes the ability to express and enforce the temporal constraints of control activities. Moreover, the computer system is a heterogeneous system with many responsibilities. Hence, methods for embedded control systems should support not only control activities but also activities related to e.g., human-machine interaction and communication.

The work presented in this thesis contributes with methods for enforcing temporal constraints in embedded control systems. The results include an industrial case study pointing out limitations in classic real-time models and giving indications of engineering needs. Moreover, a method is presented for pre-run-time scheduling of periodic control activities under the interference of sporadic interrupts. Furthermore, a method is presented for enforcement of complex temporal constraints using standard priority based real-time operating systems. Finally, the work includes a method for management of communication resources in distributed systems.

Keywords Embedded systems, real-time system, control systems, task model, scheduling, temporal constraints.	*Language* English

ACKNOWLEDGEMENTS

First I would like to thank Christer Norström for an inspiring co-operation, for his dedication, and for his support. Furthermore, I would also like to thank Professor Jan Wikander, and Professor Martin Törngren for their support and for sharing their knowledge of mechatronics.

I would also like to thank the ret of my colleagues for making the daily work motivating and fun. I would especially like to thank Henrik Thane, Anders Wall, Jukka Mäki-Turja, and Roger Hassel. Moreover, I highly appreciate the review efforts made by Christer Norström, Hans Hansson, Henrik Thane, Jan Wikander, and Anders Wall.

Harriet Ekwall has provided invaluable support in all sorts of practical issues.

Finally, I am especially grateful for the strong support given by my fiancée Camilla and the daily smiles delivered by my daughter Lina.

Västerås, 5 mars 2002

Kristian Sandström

THESIS CONTENTS

This thesis consists of a summary with original research papers appended. These are listed below and will be referred to in the text as papers A to D.

A **Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry**
Christer Norström, Mikael Gustafsson, Kristian Sandström, Jukka Mäki-Turja, Nils-Erik Bånkestad. In Eight IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, Washington, US, April 2001. IEEE Computer Society.
Kristian Sandström had part in all parts of the work presented.

B **Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System**
Kristian Sandström, Christer Norström, Gerhard Fohler. In Proceedings of the fifth International Conferance on Real-Time Computing Systems and Applications, October 1998. IEEE Computer Society.
Kristian Sandström and Christer Norström did the main part of this work.

C **Managing Temporal Constraints in Control Systems**
Kristian Sandström and Christer Norström. MRTC Technical Report 02/45. The work is an extension of the work presented in: Managing Complex Temporal Requirements in Real-Time Control Systems, Kristian Sandström and Christer Norström, In Proceedings of 9th IEEE Conference and Workshop on Engineering of Computer-Based Systems, Sweden, April 2002. A shorter version of this technical report has been submitted to the 8th IEEE Real-Time and Embedded Technology and Applications Symposium, California, USA, September 2002.
Kristian Sandström provided the main part of this work.

D **Frame Packing in Real-Time Communication**
Kristian Sandström, Christer Norström, Magnus Ahlmark. In proceedings of RTCSA 2000 Korea, December 2000. IEEE Computer Society
Kristian Sandström and Christer Norström provided the main part of the work.

THESIS SUMMARY

Contents

1 Introduction

The trend today and in the foreseeable future is the addition of more and increasingly complex functionality in embedded systems. In the development of Embedded Control Systems (ECS) a bulk of the total development time is spent on testing and maintenance. To successfully meet the challenge of developing more advanced products without increasing costs, the development of embedded control systems must put more emphasis on analysis and design. Although there are many methods for analysis and modelling of ECS described in the real-time research literature, few of them are actually used in industry. The reasons for this are manifold. The needs of industrial development processes are rarely covered by a single existing real-time method and integration of different methods can be complicated and may involve unsolved research problems. Also, some real-time methods need supporting tools in order to be useful in industry. Moreover, in developing these tools additional research problems are likely to be encountered when methods are deployed in industrial development processes. Finally, education is needed for engineers to be able to use the methods properly and efficiently.

The work provided in this thesis covers methods for use in the design and implementation of ECS. The focus is primarily on the specification, analysis, and realization of temporal constraints. The results are viewed with respect to both industrial and academic relevance.

The structure of this summary is as follows: Section 2 presents the background and motivation for this work. Section 3 presents the academic results and the industrial relevance and Section 4 provides a summary and gives directions for future work.

It should be noted that there is no overview of related work in this summary. Related work is presented in the papers that this thesis consists of.

2 Background and Motivation

The main goal for the work throughout this thesis has been to provide methods that aid the engineer in specifying and fulfilling the intended system behaviour for ECS. In this section, the author's view is given of what the requirements are for successful specification and verification of ECS. This section is divided into three sub sections; *Temporal constraints* describing constraints for ECS, *Temporal analysis and attribute assignment* treating the process of realising a desired behaviour, and *The Context of ECS* discussing the context in which methods for ECS should be valid.

2.1 Temporal Constraints

The development process for embedded control systems is similar to that of other computer based systems with respect to the high-level process elements of design, implementation, and verification. In detail there are differences adhering to the enforcement of temporal requirements of control systems. During the design of the computer control system the activities that will carry out the intended system behaviour are formed, as are the interactions between activities and the temporal constraints of these activities and interactions. If a system is going to be formally analysed a precise model for specifying temporal constraints is required. Common in many analysis models in real-time theory is the ability to specify deadlines and/or offsets for the execution of periodical activities. Although these mechanisms are appropriate for expressing constraints on many activities, some limitations arise when used for temporal constraints related to control design. In an industrial project in the automotive industry in which the author participated, the main objective was to make an automated tool for pre-run-time scheduling. During this project the observation was made that the classical task model used was the reason for why the tool did not relieve the engineer of the scheduling burden to the extent that was intended. The engineers had to express temporal constraints originating from control system design, and those constraints were quite hard to express using the basic deadlines and offsets that were provided. When using the classical model for expressing control constraints the engineers had to manually distribute offsets and deadlines over time to make the specified system schedulable. To understand why this was necessary, one should take a look at typical control constraints and their sources.

A key property of a control system implemented on a single CPU is that only one activity can be performed at any instant in time. Moreover, the processor architecture and the software implementation result in activities that have different execution time from one activation to another. The consequences of this include:

- That activities do not have strict periodicity due to interference from other activities.
- That time varying delays are introduced between the sampling of process data and the corresponding actuation of corrective data.
- That correlated sampling activities do not necessarily observe the process at the same point in time, if executed on the same CPU they cannot execute truly in parallel.

The consequences listed above can all be dealt with by control theory and design, but will put some constraints on the real-time system models and methods. The deviation from strict periodicity for example, can be accounted for in control design if the variations have known bounds. This will in turn put requirements on the expressiveness of the used real-time models. Constraints that come from control design include:

- Constraints on the variation of the period time of execution of activities (jitter). Deviation from exact periodic execution will affect, e.g., sampling. Since exact periodic execution is hard to accommodate (in computer systems) for a set of tasks with different period times, the ability to specify bounds for the variation is important.

- Constraints on maximum time from start of one activity to completion of another. The ability to specify bounds on the delay of control loops. In complex control systems, several control loops may interact and the period times for the different control loops might differ. This includes the ability to specify bounds on transactions where the tasks involved have different period times.
- Correlation constraints bounding the time between the executions of several concurrent activities. This type of constraint represents the behaviour of several concurrent activities that need to be synchronised in time, e.g., sampling of interrelated data. This constraint can be important in order to get a coherent picture of the state of the controlled process.
- Combinations of constraints, such as those listed above. In a control loop the variation of the period time of sampling and actuation may need to be bounded, at the same time as the time from sampling to actuation has a time limit. Furthermore, the entire control loop may be part of a larger control structure and needs to be synchronised with other activities.

The constraints listed above are relative temporal constraints as opposed to deadline and offset constraints that for periodic activities generally are absolute. As an example, a jitter constraint could be expressed using a deadline and an offset, but there exist many different deadline offset combinations that fulfil one specific jitter constraint. In a system with many activities there could be an immense number of possible combinations for assigning deadlines and offsets, and it is hard to find a solution by hand if only a part of those combinations lead to a system that is schedulable. Hence, it can be difficult to transfer control constraints to a classic real-time model with deadlines and offsets. To aid the engineer developing control systems, methods and tools based on control constraints are needed.

2.2 Temporal analysis and attribute assignment

Disregarding the model used, activities specified during the design usually are implemented utilising some infrastructure, e.g., a real-time operating system and communication subsystems. To transfer information about the design to the implementation, the design information eventually has to be expressed using the infrastructure primitives, e.g., task and task attributes such as priorities. One method for assigning implementation primitives so that temporal constraints specified in the design phase are met is to make ad hoc assignments of the task attributes (e.g. priorities). The resulting system behaviour is then verified and changes in the attribute assignments are made iteratively until a satisfactory behaviour is achieved. In the design phase the verification can be done by real-time analysis methods and in later phases by testing. One problem with the manual assignment of task attributes is that it does not scale well, i.e., it is difficult to use for other than very small systems. Depending on the used run-time system, a number of more methodical approaches are available for making the attribute assignment. For pre run-time scheduled systems the synthesis is an integrated part of the scheduling phase where the execution windows of activities are constructed so that temporal constraints are met. For pre-emptive priority based run-time scheduled systems, methods like rate monotonic and deadline monotonic can be used to assist the engineer in assigning priorities of activities. However, as discussed earlier, ad hoc transformation of more complex temporal constraints to e.g., deadlines and offsets are not always practical. Therefore, methods are needed that either makes those transformations or make the synthesis directly from the complex temporal constraints. There exist few methods for making the synthesis from complex control constraints. Available methods for pre-run-time scheduling, e.g., [1][2][3], support control constraints but lack support for pre-emption, sporadic activities, and varying task execution times. For run-time scheduled systems there are methods with support for a sub set of the control constraints in section 2.1, e.g., [4][5].

Analysis of the temporal behaviour can both be used as part of methods for assigning task attributes and as a stand-alone tool. Besides conventional analysis methods for real-time systems based on a worst-case scenario, methods that could more accurately model the environment and the true behaviour of a control system would be beneficial. Since there is a cost in over estimating the resource requirements, the analysis should be as exact as possible. But, even if the analysis is exact it is not possible for most products to dimension the control system for scenarios that will occur at most a few times during the products lifetime. Instead temporal constraints could be guaranteed with some reliability and confidence. Potential methods could be based on quality of service for real-time systems and statistical measures for analysis. Furthermore, since the correct behaviour of an ECS is dependent on correct functional behaviour at the right point in time, methods for co-analysis of the temporal and the functional behaviour could add confidence in early analysis of a system. Potential methods include formal methods with models for representing system functionality and environmental behaviour as well as state machine based methods with support for temporal analysis.

2.3 The Context of Embedded Control Systems

To make methods for synthesis and analysis of control constraints useful to an engineer there are additional constraints that must be considered to make the methods applicable in real systems. The computer system in complex embedded control systems have to cater for a vide variety of services. This includes, human-machine interaction and interfacing with communication sub systems and other target specific hardware. The control system does not only have to share the same resources with other services. It also has to interact and exchange information with them. Hence, accounting for the behaviour of other computer activities has to be an integrated part of methods for temporal analysis of computer control systems. This fact leads to a number of issues to consider when developing methods for analysing and enforcing temporal constraints of embedded control systems:

- It is important to consider both periodic and sporadic activities in the same system. Periodic activities originate in e.g., control functionality whereas sporadic activities can reflect an event-triggered behaviour or interrupt generated from e.g., the communication sub system. Both of these types of activities will coexist in most control systems. However, these two types of activities are often treated differently and separately in real-time analysis.
- The architecture of contemporary CPU's and computers, as well as the algorithms in the software implementation, lead to execution times of activities that vary from one activation to another. This directly affects the real-time analysis since e.g., jitter will be a result of a varying execution time.
- Activities in computer control system interact and have shared resources. The consequence is that methods should include mechanisms for synchronisation of the execution of activities, also between activities with different period times. Furthermore, there should be mechanisms for managing shared resources, also between periodic and sporadic activities.

Many embedded control systems are distributed systems with temporal constraints involving activities executing on different nodes and communication over a communication bus. Methods for analysis and synthesis can cover the complete system or the system can be divided into sub parts that are treated separately. If the system is treated in parts, the temporal constraints spanning several nodes must be partitioned into separate constraints for each node and bus. Two reasons for dividing the analysis and synthesis into sub parts are reduction of complexity and separate handling of system parts by sub contractors. For systems where the complexity can be managed and it is possible for one actor to control the details of the entire system it is possible to make analysis and synthesis that cover the entire system at once. For this, many methods for scheduling activities on a processor can, with modifications, be applied in planning resource usage on communication busses. However, there are additional matters to consider. In embedded control systems most data (signals) sent between activities are quite small in terms of bits and to handle them individually in bus communication would cause a large overhead. Therefore, several signals are allocated to a larger container, a frame. However, the transmission of the frame still has to meet the most severe temporal constraint of the contained signals. There are several criteria that could be considered when allocating signals to frames, e.g., resulting use of bandwidth and resulting number of frames sent to a specific node. To manage the resource usage of the bus alone is a hard task in complex systems. It could prove very hard to succeed with global system resource management, without trying to limit the complexity.

In the development of embedded control systems it is important that engineers have access to methods and tools that support the hard and complicated task of designing and implementing cooperative activities with complex temporal constraints. It is also important that the methods incorporate knowledge from both control engineering and computer engineering, in order to be a successful aid in design of embedded computer control systems. Furthermore, methods should manage real world characteristics originating from e.g., the heterogeneous nature of embedded control systems.

3 Focus and Aim

The main goal of this thesis is to provide methods and knowledge that assist the engineer in the design and implementation of embedded control systems. In more detail, the methods should give engineers support for describing the desired temporal behaviour of activities in a computer control system. More over, the methods should assist the engineer in transferring the described temporal behaviour to an implementation and to verify that the implementation meets the specified behaviour. The methods should also add ease to the management of temporal constraints in the process of maintenance and further development of control systems. Finally, it is important that the heterogeneous nature of embedded control systems is reflected in the methods by providing support for periodic control activities in a computer system consisting of a variety of activities such as human-machine interaction and networking.

4 Results and Contributions

In this section the results of the thesis is organised according to academic results, educational results, and the industrial relevance of the results.

4.1 Academic results

A key paper in the thesis is paper A, which treats the experiences from a project within the automotive industry. From the work presented in this paper several problems were identified and solved, mainly in paper B: *Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System* and paper C: *Managing Complex Temporal Requirements - A Method for Assigning Priorities and Offsets in Fixed Priority Systems*. The work provided in Paper B presents a method for fulfilling temporal constraints in heterogeneous embedded systems by integrating handling of sporadic activities with pre-run-time scheduled periodic tasks. Paper C presents a method with stronger support for control functionality by management of complex temporal constraints for use with standard priority based real-time operating systems. Furthermore, in paper B problems related to the implementation of the communication system is dealt with, whereas paper D proposes a method for managing communication resources in distributed systems.

4.1.1 Paper A: Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry

This paper discusses some findings and conclusions from introducing real-time techniques in the development of computer control systems for automotive vehicles.

Some of the most important findings can be summarised as:

1. Introduction of solid real-time theory provides valuable benefits, including early error detection and less verification efforts.
2. Real-time theory should be transferred in the form of tools, methods, and education.
3. Real-time theory has poor support for some control induced temporal constraints.
4. Results from real-time theory has to be adapted to be useful in an industrial context, the adaptation includes research issues such as providing useful information to the engineer when the system is not schedulable.

4.1.2 Paper B: Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System

This paper proposes a method for handling sporadic interrupts that pre-empts pre-run-time scheduled tasks. The motivation for this work came from introducing a pre-run-time scheduler in the development of control systems for automotive vehicles. In the system the interrupt load caused by the communication sub system was high and could not be handled by the pre-run-time scheduled system by simple ad hoc methods. The problem was solved by developing a method that accounts for interrupts in the pre-run-time scheduling process. The technique uses results for analysis of fixed priority tasks by incorporating the analysis into a heuristic tree search algorithm for pre-run-time scheduling.

This work contributes with an efficient and safe method to account for sporadic interrupts when creating a pre-run-time schedule. Furthermore, the method will add no additional run-time overhead.

4.1.3 Paper C: Managing Complex Temporal Requirements - A Method for Assigning Priorities and Offsets in Fixed Priority Systems

The paper presents a method for assigning task attributes with the objective to meet complex temporal constraints for applications running on a standard real-time operating system. Using this method, complex temporal constraints originating from control design can be expressed with a precise syntax and semantics. Based on specifications of the temporal constraints, task offsets and priorities are derived so that the constraints are met.

The constraints that can be expressed include; correlation constraints between concurrent sampling tasks (or actuation tasks), jitter constraints on period times, and latency constraints from sampling to actuation. Furthermore, except for periodic tasks, sporadic tasks can be specified as well as shared resources between sporadic and periodic tasks. The model also handles varying execution times of tasks. The method uses genetic algorithms to search for an optimal assignment of priorities and offsets for a task set. Simulations indicate the appropriateness of the method and the effectiveness compared to related work.

This work contributes with a strict syntax and semantics for specification of complex temporal constraints for control systems and a method for enforcement of complex temporal constraints using standard priority based real-time operating systems. Compared to methods proposed in [4][5] the method in presented in this paper can handle a larger set of constraints, e.g., both jitter and correlation constraints. Moreover methods in [1][2][3] do not handle sporadic tasks, shared resources between periodic and sporadic tasks, and varying execution times of tasks.

4.1.4 Paper D: Frame Packing in Real-Time Communication

This paper describes a method for allocating data shared by activities over a network into frames handled by the communication subsystem. In embedded systems the size of most data sent between activities on different nodes are quite small compared to the available data containers, i.e., frames, for the bus. Even if there are small frames available the transmission overhead increases as the size of the frame decreases. As a result, signals from several activities have to be allocated to the same frame in order to reach an acceptable performance of the communication subsystem. The signals have different temporal constraints and the allocation will therefore affect the bus utilisation. In fact, finding frame sizes and making proper allocation of signals to frames so that the bus utilisation is minimised is a NP-hard problem. The method presented allows fast allocation of very large signal sets with good result with respect to bus bandwidth utilisation. The method is based on approximation algorithms and simulations are used to verify the appropriateness of the algorithms.

The work presented in this paper makes contributions by formulating the packing problem, showing that the packing problem is NP-hard, presenting a simple and effective heuristic for frame packing, and demonstrating the effectiveness of the algorithm on realistically sized problems derived from the automotive industry.

4.1.5 Tools

To aid the development and evaluation of methods for managing temporal constraints in ECS two software tools have been developed.

GenECS is a tool for generating simulated system specifications for embedded control systems. The tool generates task sets with temporal constraints according to a predefined profile. It is possible to control the distribution of values generated for attributes such as the worst and best case execution times, deadlines, jitter constraints for period times, and utilisation for sporadic and periodic tasks.

TemporalControl is a tool for assigning attributes for pre-emptive priority based real-time operating systems. Given a specification of a task set with temporal constraints the tool assigns priorities and offsets to the tasks so that the constraints will be met during run-time. The task set can include both periodic and sporadic tasks with synchronisation constraints and shared resources. Temporal constraints that can be solved include jitter, correlation, and latency.

An earlier version of the tools where developed as part of a master thesis conducted by Johnnie Blom under supervision of Kristian Sandström.

4.2 Educational results

The knowledge gained during the work of the thesis has been exploited for many educational purposes. As the most prominent the following are mentioned:

- The development of course material and lab exercises for a basic course in real-time systems at Mälardalen University.
- Development and realisation of several instances of a large-scale project course in distributed real-time systems. Research results have been applied in the course, e.g., in a project developing design and analysis tools for embedded real-time systems.
- A compendium in real-time systems used at Mälardalen University, Uppsala University, and Dalarna University.

4.3 Industrial relevance

Throughout the work of this thesis there have been an exchange of knowledge with industrial partners. Methods have been tested in industrial cases and research issues have been extracted from problems found in industrial projects. Furthermore, several courses in real-time systems have been given to industry. In the rest of this section, the more extensive co-operations are presented.

4.3.1 Technology transfers

In a project with Volvo Construction Equipment Components a real-time design model and analysis was introduced in the development of their computer control system. The model has evolved and is still in use in the development of the embedded control system for construction equipment. From this project a lot of problems were identified, and some of them are solved in this thesis.

In a master thesis [6], methods for allocation of signals to frames and scheduling of frames on the network where developed. The work was conducted in cooperation with Volcano Communication Technologies. A simulation environment was developed and the results have later been used in an industrial project.

4.3.2 Prototypes

As a commercial spin-off, a prototype tool for pre run-time scheduling, the Configuration Compiler, was developed. The prototype was developed for use at Volvo Construction Equipment Components and is now incorporated in the tool set for the commercial real-time operating system Rubus provided by Arcticus Systems.

5 Conclusion and future work

In order to develop and maintain embedded control systems of higher quality in shorter time, it is important that methods and tools support the complex task of computer control system development. Tools and methods that help the engineer to formulate and enforce the complex temporal constraints put on control systems are needed. Furthermore, since most embedded control systems are heterogeneous, support for systems consisting of subsystems with different characteristics are required.

This thesis proposes methods for specification, analysis, and synthesis of temporal constraints for embedded control systems. Furthermore, methods are provided that are adapted for the reality of embedded systems with a mix of control and computer system related functionality. The contributions are:

- An industrial case study pointing out limitations in classic real-time models and giving indications of engineering needs.
- A strict syntax and semantics for specification of complex temporal constraints for control systems.
- Scheduling off periodic control activities under the interference of interrupts.
- A method for enforcement of complex temporal constraints using standard priority based real-time operating systems.
- A method for management of communication resources in distributed systems.

Several tools and prototypes have been developed to validate the results and some of the prototypes have been transferred to industry and are in successful use today.

In future work the author would like to investigate how the methods can be better adapted to practical operating conditions of a system. The method for attribute assignment presented in this thesis assigns attributes based on a safe analysis using a system model that will represent the outer boundaries, with respect to the temporal behaviour of the real system. The analysis will often provide over estimations if compared to how the real system will behave. The main part of this pessimism in the analysis comes from the assumption that sporadic activities will be activated with their maximum frequency and with a worst-case phasing relative the periodic activities. In the real system, this is in many cases unlikely to occur and surely in some cases it cannot. For many products housing embedded control systems, the price of having absolute certainty, with respect to resource usage, cannot be justified. Instead the system could be dimensioned to meet its temporal constraints with some probability within some confidence interval. One possible method for achieving this property for the method presented in paper C would be to base the analysis part of the genetic algorithm on samples taken from simulations.

Another possible direction for future research is to look at methods for use earlier in the development process and how they link to the methods presented in this thesis. Since, an undetected error made in early development is much more expensive than an error made late it is important to have early measures of the possibility to meet functional and temporal constraints. It is also important to find a coherent methodology that connects methods throughout the development process from architectural analysis to implementation and maintenance.

6 References

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

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

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

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

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

[6] Ahlmark M. Local Interconnect Network (LIN) - Packaging and Scheduling. Master Thesis, Department of Computer Engineering, Mälardalens University, June 2000.

PUBLICATIONS

Managing Complex Temporal Requirements in Real-Time Control Systems
Kristian Sandström and Christer Norström. In proceedings of 9th IEEE Conference on Engineering of Computer-Based Systems, Sweden, April 2002.

Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry
Christer Norström, Mikael Gustafsson, Kristian Sandström, Jukka Mäki-Turja, Nils-Erik Bånkestad. In Eigth IEEE International Conference and Workshop on the Engineering of Compute-Based Systems Washington, US, April 2001. IEEE Computer Society.

Verifying Temporal Constraints on Data in Multi-Rate Transactions
Anders Wall, Kristian Sandström, Jukka Mäki-Turja, Christer Norström. In proceedings of RTCSA 2000 Korea , December 2000. IEEE Computer Society.

Frame Packing in Real-Time Communication
Kristian Sandström, Christer Norström, Magnus Ahlmark. In proceedings of RTCSA 2000 Korea , December 2000. IEEE Computer Society.

Findings from introducing state-of-the-art real-time techniques in vehicle industry
Christer Norström, Mikael Gustaffson, Kristian Sandström, Jukka Mäki-Turja, Nils-Erik Bånkestad. In industrial session of the 12th Euromicro Conference on Real-Time Systems, Stockholm, Sweden, June 2000.

Modeling and Scheduling of Control Systems
Kristian Sandström. Licentiate Thesis, ISSN 1400-1179l, Department of Machine Elements, The Royal Institute of Technology, Sweden, 1999.

Constructive Feedback turns Failure into Success for Pre-Scheduled Systems
Kristian Sandström, Christer Norström. In Swedish National Real-Time Conference SNART'99, August 1999.

Towards Efficient Analysis of Interrupts in Real-Time Systems
Jukka Mäki-Turja, Gerhard Fohler, Kristian Sandström. In work in progress, 11th EUROMICRO Conference on Real-Time Systems, York, England, May 1999.

Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System
Kristian Sandström, Christer Norström, Gerhard Fohler. In Proceedings of the fifth International Conferance on Real-Time Computing Systems and Applications,pages 158-165, October 1998. IEEE Computer Society.

An Overview of RTT: A Design Framework for Real-Time Systems
Christer Norström, Jukka Mäki-Turja, Kjell Post, Mikael Gustaffson, Jan Gustafsson, Kristian Sandström, Ellus Brorson. Journal of Parallel and Distributed Computing, no 36, August 1996.

A Graphical Design Environment for Development of Object-Oriented Hard Real-Time Systems
Christer Eriksson, Roger Hassel, Lennart Myrehed, and Kristian Sandström. TOOLS Europé 95, Paris, France, Mars 1995. Published in TOOLS 16 by Prentice Hall, ISBN 0-13-443128-6.

Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry

by

Christer Norström, Mikael Gustafsson*, Kristian Sandström, Jukka Mäki-Turja, and Nils-Erik Bånkestad**

Mälardalen Real-Time Research Centre, Department of Computer Engineering, Mälardalen University, Västerås, Sweden
*TietoEnator ArosTech AB, Västerås, Sweden
** Volvo Construction Equipment Components AB, Eskilstuna, Sweden

In Eigth IEEE International Conference and Workshop on the Engineering of Compute-Based Systems, Washington, USA, April 2001. IEEE Computer Society.

Experiences from Introducing State-of-the-art Real-Time Techniques in the Automotive Industry

Christer Norström, Mikael Gustafsson, Kristian Sandström, Jukka Mäki-Turja, and Nils-Erik Bånkestad

Mälardalen Real-Time Research Centre, Department of Computer Engineering, Mälardalen University, Västerås, Sweden
*TietoEnator ArosTech AB, Västerås, Sweden
** Volvo Construction Equipment Components AB, Eskilstuna, Sweden

Abstract

The use of state-of-the-art real-time techniques in industry is still rare. The reason for this is three-folded: (1) the lack of commercially available tools, (2) the lack of methodologies that considers real-time throughout the complete development process, and (3) the lack of competence in real-time theory among industrial practitioners.

In this paper we present a case study of introducing state-of-the-art real-time techniques in industry. The case study was done as a collaboration between Mälardalen University and the industrial partners Volvo Construction Equipment AB (VCE) and TietoEnator ArosTech. VCE develops computer control systems for construction equipment vehicles, such as wheel loaders, graders, and articulated haulers. TietoEnator ArosTech is a firm of consultants with expertise competence in the area of embedded real-time systems.

We will present both the used methodology and the findings from introducing this methodology in an industrial project. The methodology emphasis is on introducing timing requirements early in the design of a system and it relies on the use of a well defined design language. We will present our findings categorized into methodological aspects, technology transfer, and technical aspects. The main result reported can be summarized as “people, not paper, transfer technology”.

1 Introduction

Development of complex embedded systems is a growing area, i.e., we see more and more applications that are dependent on the use of embedded computers. Examples include highly complex systems, such as medical control equipment, mobile phones, and vehicle control systems.

Most of the embedded systems can also be characterised as real-time systems, which means that their correct function is dependent on both correct functional results and that the results are produced at the correct time.

The increased complexity of these systems leads to increasing demands on issues such as requirements engineering, high level design, early error detection, productivity, integration, verification, and maintenance. This calls for methods and models that enable a controlled and structured way of working during the complete life cycle of the system [Kal88].

There exist many design methods for real-time systems like, UML-RT and HRT-HOOD. However, these methods often concentrate on the logical and structural decomposition rather than focusing on the temporal behaviour. The temporal behaviour is often added on top. This is not so strange since these methods are based on general software development methods that are not focusing on embedded real-time systems. Furthermore, these methods have no, or limited, support for high level timing analysis and do not provide support for automatic mapping from the design to a resource structure. This often leads to a semantic gap between the design and the implementation, that is, the code and design description may not describe the same version of the system. Thus, classic problems during integration may occur, such as erroneous synchronisation and communication interfaces and that the system is hard to maintain.

Therefore, we have developed a model and method focused on the real-time properties of a system. The key property of the model and method is specification of a high level design that includes the specification of temporal constraints, communication and synchronisation. Furthermore, the model and method supports formal verification of these properties, early system integration, and efficient testing.

The aim of this paper is to briefly present this model and method, as well as our findings from introducing and using them in an industrial project. This project was performed as a cooperation between Mälardalen University, Volvo Construction Equipment AB (VCE) and TietoEnator ArosTech.

VCE has had onboard electronics since 1981 for specific functionality. Currently more and more functionality is provided by the computer control system. This has led to an increased number of people involved in the development of each product, and thus the need for better development methods and tools.

This was the motivation for the university to participate in the development of a new computer control system for the next generation wheel loaders. Since a complete new architecture was to be developed we were given the opportunity to introduce new technologies and methods.

Many functions are similar in different vehicles and therefore it would be a desired property to be able to reuse existing solutions. This was the starting point for defining a new architecture that could be used for all types of future construction equipment. Hence, the result of this project will act as a basis for extracting a product line architecture [Bos00]. However, the latter step is outside the scope of this paper.

Thus, this paper is focused on presenting our findings from introducing state of the art real-time technology in an industrial project. The validity of these findings is based on a single, but extensive, case study of one industrial project. Some of the findings are strengthened by similar results in other industrial projects that also have utilized state of the art real-time technology [Cas98, Mel98].

The outline of the paper is as follows: Section 2 presents briefly the characterization of the application. The design language used is described in Section 3. Section 4 presents briefly the tool that maps the design to a resource structure. Thereafter, in Section 5, the development methodology is presented. In Section 6 we present our findings categorized into findings related to methodological aspects, technology transfer, and technical aspects. Finally, in Section 7 some conclusions are given.

2 Application characteristics

The application is a vehicle control system with high demands on safety, reliability, and timeliness. The hardware in the system consists of two nodes that are connected via redundant buses. The application contains tasks, running at different period times, which collaborate to perform certain control functions. The system contains about 80 tasks with well-defined functionality. Each node is very I/O intensive. The complete system has about 150 I/O channels connected to it.

The execution times of the tasks in the application range from about 10 μs to 1 millisecond. The application is, due to the construction of the hardware, interrupt intensive. Since this application has many interrupts, the effect of these interrupts cannot be neglected when scheduling the application tasks.

The worst-case utilization of the processors for the critical part is around 80%, divided into 35% for interrupts and 45% for application tasks. The spare capacity left is used by soft real-time tasks. At run-time, the spare capacity will be more than the remaining 20% if the load is less than the worst case.

The reason for the extensive use of interrupts is mainly due to the hardware design. The hardware could not be modified since it was already designed and certified when the software development started.

3 Design language

The design language should be simple with a few, but powerful, constructs with clearly defined syntax and semantics. The reason for this is twofold: 1) parts of the implementation can be automatically generated by tools and, 2) the traceability from specification to implementation is improved since it is easier to overcome the semantic gap between design specification and implementation. The design is tightly coupled to the implementation; it is easier to fix a bug by correcting the design than to just make a modification in the code, when tools generate parts of the implementation directly from the design specification.

Another important principle is the separation of concerns. A specific example in the language is to separate communication and synchronization constructs from the C-code. This gives advantages in verifying the temporal behavior of the system (analyzing or estimating the execution time of the code is easier since it is independent from other components of the system). The integration phase also becomes easier when the interaction and synchronization is specified and analyzed early in the design.

The most important contribution, however, is that temporal constraints are defined early in the development process, which enables an early temporal verification.

The key elements of the language in increasing order of granularity are:

- Application – defines the top level of a complete software system.
- Modes and mode transitions – defines a high level state machine.
- Transactions – describes the functionality in a mode.
- Interaction graphs – describes the interactions between tasks that make up a transaction.
- Tasks – the computational elements of the design language.

3.1 Application model

A classical way of attacking problems is by ”divide and conquer”, i.e., by decomposing the problem into more manageable sub-problems. This is done here by hierarchical decomposition, where an application is broken down into modes. A mode is an operational state of the application. Different modes contain different functionality. Each mode should only include the functionality that is needed for the desired behaviour. A picture of this hierarchy is shown in Figure 1.

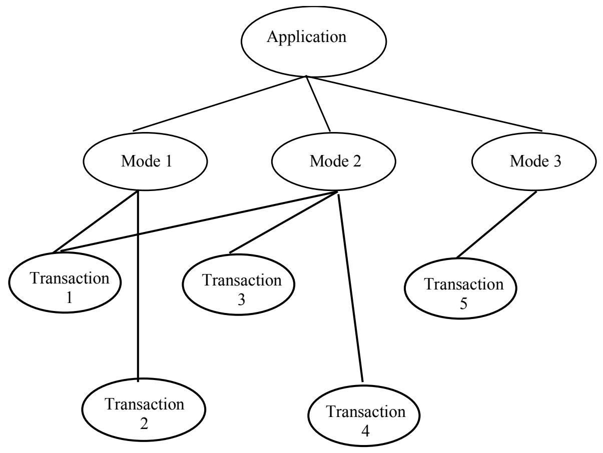


Figure 1: Application model

3.2 Modes and mode transitions

A mode describes specific functionality in a system state. If the functionality differs substantially from one state to another, one should separate them into two different modes. An example is the control system in a vehicle, which can have different functionality depending on the status of the vehicle. If the vehicle is fully functional, the control system is in full operating mode. If a severe error occurs the control system can take the vehicle into a reduced functionality, mode where only the most critical functions of the control system are provided, so that the vehicle can be taken for repair.

Modes in the system are described in a mode transition graph, comparable to a state transition graph, where all legal transitions between modes are depicted. An example mode transition graph for our vehicle is illustrated in Figure 2.

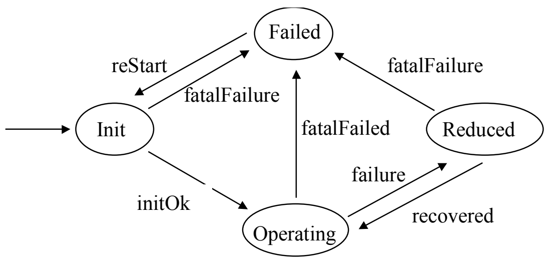


Figure 2: Mode transition graph for a control system for a vehicle

Why modes? In almost all application there is some kind of mode concept, even if implicit. For example when the system is starting up, initialisation functionality is provided which is no longer needed when the system is fully operating. Many systems also have a failure mode with reduced functionality. If there is no way of specifying these modes they have to be implemented ad hoc in the code which makes it hard to understand and maintain the system.

3.3 Transactions and interaction graphs

For each mode a number of functions must be provided, we call these transactions. A transaction consists of a collection of tasks that together provide the desired functionality. The interaction and dependencies between tasks are described by communication and synchronisation constructs. Communication is specified as a directed relation from one task to another. Synchronisation can be described by precedence relationships between tasks or by mutual exclusion of task that share a common resource. The temporal behaviour of tasks in the transaction is specified by temporal attributes of single tasks. Besides tasks, interrupts can also be specified. Including interrupts in the specification makes it possible to include them in the analysis.

3.4 Task

A task is the smallest executable unit. A task is described with a set of functional and temporal parameters:

Functional:

- **Entry function**. The entry function specifies the function to perform on each invocation. This, together with the state and input, defines the functionality of the task.
- **State**. A task has some state variables, (comparable to instance variables of an object), which keep their values across activations of the task. The variables constitute the task state.
- **Ports**. Since communication primitives are not allowed in the code, communication is specified in the interaction graph. Each task is equipped with in- and out-ports. The in-ports acts as input to the entry function and the result of the entry function is placed on the out-ports of the task.

Temporal:

- **Period time**. The period time of the task.
- **WCET**. Worst Case Execution Time of the entry function. Note that this value is assessed and used as an additional design parameter during the design and verified after implementation.
- **Release time**. Remember that every task is a member of a precedence graph and therefore has a period. The release time is the earliest time the task can be activated, relative to its period start.
- **Deadline**. The deadline is the latest time a task is allowed to terminate, relative to its period start.

The *execution semantics* of a task is at activation to read the in-ports, thereafter perform the function, and before termination write the result to its out-ports. This construction means that each task can be designed without knowing where the input data was produced and where the produced output data will be used.

4 Mapping of the design to a resource structure

The Configuration Compiler tool maps a textual based description of the design to a resource structure, as illustrated in Figure 3. The Configuration Compiler is a pre-run-time scheduler that generates dispatch tables and communication infrastructure for each mode. Besides the mapping of the model, the tool also supports specification of architecture specific attributes like performance, the time granularity of the run-time dispatcher, communication times, and number of nested pre-emptions allowed. The implementation of the Configuration Compiler is based on a heuristic tree search strategy, similar to the one presented in [Ram90]. The major difference is that this scheduler takes interrupts and architecture specific attributes into account. The current version of the tool is adapted to the real-time operating system Rubus[1].

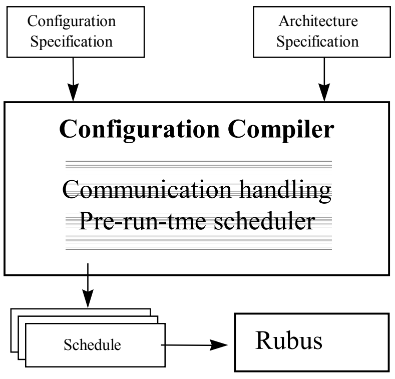


Figure 3: The Configuration Compiler

5 Development methodology

The development methodology defines the workflow when developing an application. The methodology employed in this project is iterative and quite traditional. The emphasis in the method is to derive a high level design that enables early schedulability analysis. To facilitate this it is required that synchronization, communication, and temporal attributes are defined early in the design process, which is of no problem except for execution times of the tasks. The execution times are normally derived from the code. However, in this approach we specify (estimate) an execution time budget for each task. The execution time budget is later in the implementation phase used as an implementation requirement. Estimating the execution time budgets is a delicate issue that requires highly skilled engineers with a lot of experience. However, if the estimate can not be fulfilled a negotiation strategy has to be employed. That is, execution time may be borrowed from

[1] Rubus and the Configuration Compiler are commercial products, see www.arcticus.se.

another task, which does not utilize the allocated execution time. The development methodology is general and can be adapted to different design languages (modeling languages). The method is briefly described in Figure 4 and by the following text.

I. **Requirements engineering**. Here are the requirements formulated by the customer of the system.

II. **Requirements analysis**. In this stage the functions of the application are identified from the requirements specification. An important aspect here is also to determine temporal constraints for these functions.

III. **High-level system decomposition**. In this stage the application's different operational modes are identified together with valid transitions between them, by specifying the mode transition graph.

IV. **Function decomposition and structuring**. The functions, for each mode, are decomposed into transactions. Note that one transaction could belong to several modes. Transactions are decomposed into smaller units called tasks and their low-level functions are specified together with the data flow information between them. Some high level functions has parts that have a high demand of responsiveness or are very frequent (but small) so that implementing them as tasks would be infeasible. Therefore such low-level functions are implemented as interrupts. This is formally described in an interaction graph.

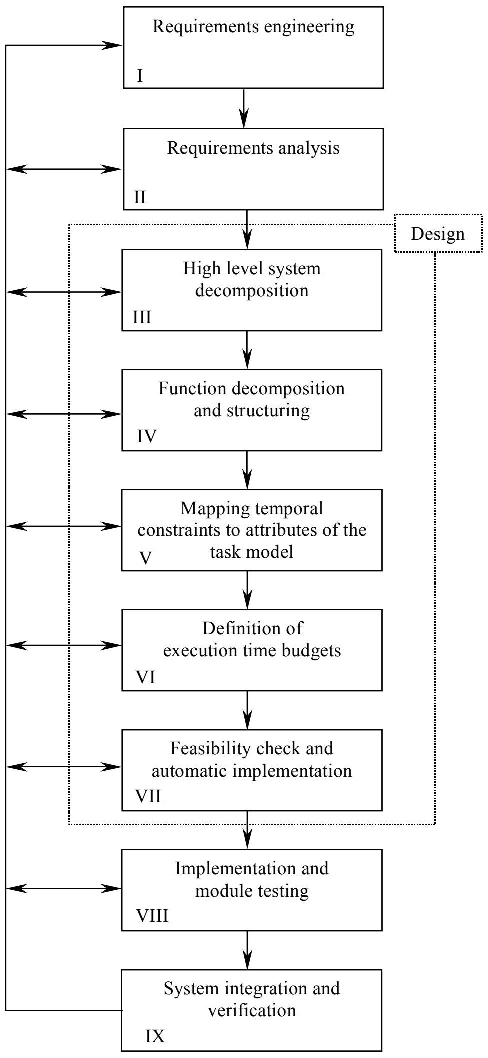

Requirements engineering
I
Requirements analysis
II
Design
High level system decomposition
III
Function decomposition and structuring
IV
Mapping temporal constraints to attributes of the task model
V
Definition of execution time budgets
VI
Feasibility check and automatic implementation
VII
Implementation and module testing
VIII
System integration and verification
IX

Figure 4: The design methodology

V. **Mapping temporal constraints to attributes of the task model**. In the previous stage the high level functions were decomposed into smaller units and structured according to the interaction between them. This step has to brake down the high level temporal requirements into temporal attributes for these smaller units. The expressiveness of the task model attributes are different, and lower level, than specified for the high level functions, so it is important that this transformation is done in a safe way, i.e., that the task model attributes does not violate any of the high level constraints. It is also important that this mapping does not overconstrain the system.

VI. **Defining Execution Time Budget**. Traditionally the assessment of WCET is done by either measurements or by statically analyzing the code produced for each task. In this approach, however, execution time budgets are defined, these budgets are later in step VIII used as implementation requirements. The reason for this is that a feasibility test for the system, and a possible re-engineering, can be done at an early stage, and thus provide early detection of design errors related to resource utilization, communication and synchronization.

VII. **Feasibility check and automatic implementation**. The formally described design can be checked for temporal correctness even if no actual (low-level) implementation has been done. This is done by a static scheduler, which tries to find a feasible schedule. Besides the schedule, the communication infrastructure is automatically generated.

VIII. **Implementation and module testing**. The implementation of tasks is simply done by traditional programming (coding). Besides the traditional functional specification, the programmer also has the execution time budget as an implementation requirement, i.e., the programmer has to implement the specified function in a way that it does not violate the budget. The module testing includes both verifying the functional behaviour as well as that the time budgets are not violated. If the time budget cannot be met a redesign has to be done.

IX. **System integration and verification**. The integration phase is usually done very quickly and without problems since the actual integration was done in the design with a strict semantics. The major work is to do the integration testing.

The above figure and listing defines the activities performed in each step, and the iteration when using this method.

6 Findings

In this section we will describe the findings acquired when introducing and using the design language and method earlier described in Section 2. The findings are categorized into those related to development methodology, technology transfer, and technical issues respectively. The development methodology covers the findings based on the use of the design language and method. The technology transfer part describes issues regarding the transferring and introduction of new technology and especially real-time technology into an organization. The technical issue part presents new or relevant technical challenges that have been discovered during this work.

6.1 Design methodology

Finding 1: The design language provides a good basis for the design description.

Motivation:

Using the design language described in Section 3 gives three major benefits when designing a system:

1. *It gives a skeleton of the application, which can be analysed without having a single line of code.*

2. *The analysis leads to early error detection of communication, synchronization, and timing errors.*
3. *Simplified system integration.*

Currently we can analyse communication, synchronization, and timing requirements. Communication is analysed in three different aspects. Firstly, the types of connected ports are checked, which ensures that the proper data types are passed to the tasks. Secondly, the analysis will reject a design where the amount and rate of data passed through the system makes it infeasible to fulfil the timing requirements. Thirdly, data consistency is checked. Again, if there is no possible way of fulfilling all timing requirements and at the same time guarantee data consistency, the design is rejected.

The analysis of the synchronization makes sure that all precedence and mutual exclusion relationships between tasks can be guaranteed in conjunction with guaranteeing the timing requirements.

Finally the analysis of the timing requirements reveals if it is possible to find a schedule for the given design and execution time budgets that fulfils these timing requirements. If it is impossible to fulfil the timing requirements the design will be rejected.

The analysis presented above leads to early detection of errors, in the design, of the properties that are analysed. Such errors are otherwise often found in the integration phase of the project and thereby cost a lot of time and effort to correct.

System integration is also simplified by the early analysis. If the implementation of the code of each task comply with the interface given by the design, i.e., retrieving data only from the in-ports, performing the desired function within the given execution time budget, and producing data only to the out-ports, then the integrated system will fulfil the design and thus satisfy the requirements. Thus, a step of the development process, that often tends to be quite troublesome and leading to costly delays in the project, are simplified.

Note that the only thing that has to be added to implement the design is the task code, everything else is automatically generated, i.e., communication, synchronization and an execution scenario (schedule).

Finding 2: The use of a precise design language

a) Enables parallel implementation and testing of the tasks.

b) Facilitates efficient integration of new personnel into the project.

Motivation:

a) The task model stipulates tasks, which have no synchronization or communication within the code. Recall from section 2, that each task uses a computational model based on input - calculation - output. That leads to that each task can be implemented and tested in parallel since each task is only dependent on its own state and the values of its in-ports to make a calculation. The module testing is, thus, very simple to make, just feeding values to the in-ports and monitoring the output. This also allows regression testing of modules.

b) One small group of people, who have good knowledge about the system and a good feeling of future demands on the systems, develops the design. The design they come up with must be stable, that is, not too many major changes are allowed to occur after the implementation phase start. If that is accomplished, it is easy to introduce new personnel into the implementation phase since each new employee or consultant only has to understand the design language and obey the given interface to be able to start to implement and test. The design language has decreased the introduction time for new employees substantially.

Finding 3: The methodology increases the time spent in the design phase but shortens the implementation time.

Motivation:

We feel that the time to complete the design phase has increased compared to similar projects, which have used traditional informal techniques (such as structured analysis and design). This is not surprising since a precise design with analysis is harder to come up with, compared to a design that just is based on written documents. However, we feel also that the precise design has lead to shorter time spent on implementation, test, and integration due to reasons described earlier in this section. We also believe that it will be much easier to maintain a system based on a precise design compared to a traditional system. This is mainly due to two reasons:

1. Normally the implementation and the design tend to diverge which makes it hard to foresee the impact of changes and added functionality. This can be avoided by the fact that the tools are useful and actually produces verified functionality. It is for example quite natural and widely accepted to use the compiler instead of adding object code here and there. Another restraining factor can be the fear of disturbing the order laid out by the tools, again compare with the compiler example.
2. Even if there is a good match between the design documents and the implementation it is not easy to foresee the impact of changes and added functionality. In our case several properties of the altered design can be analysed, as discussed earlier, already in the design phase. So changes or add-ons that does not comply with the implemented functionality will be detected.

Finding 4: Execution time budgets for tasks turned out to be good as a design tool and implementation requirement.

Motivation:

To be able to make an early capacity analysis of the resources in the system, like processors and buses, each task has to have an execution time budget. This budget states how much of the processor capacity the task is allowed to utilize. The difficulty in specifying this budget is to relate the execution time budget to the functional requirements of the task, e.g., for a controller it should be possible to fulfil the desired control performance within the specified time budget. If it is not possible this time budget is erroneous. The execution time budgets are then used as implementation requirements.

In this project we were really surprised that these budget estimations where so good. However, the engineers that specified these budgets had many years of experience in control system design and good knowledge about hardware close programming.

To verify that the implementation fulfils the requirements the execution time for the tasks was measured and sometime calculated.

6.2 Technology transfer

Finding 5: To be able to transfer real-time technology to industry; tools, education (courses, tutorials), carriers, and adapters are required.

Motivation:

Tools:

When transferring theories to the industry it is necessary that the theory is encapsulated in a tool, which shows the practical use of the theory [Sch96], unless the theory is very simple [Bat99]. A good example of a tool that encapsulates advanced technology well is a traditional compiler. In this case the tool was in the first version an application written in a high level language that was easy to adapt to up-coming requirements from the industry. To handle these up-coming requirements in an efficient way is important to succeed in the transfer, a part where the carrier described below play a significant role. The tool was later ported to a low-level language to get an efficient implementation.

Courses:

We have found out that an engineer requires at least two days of training to understand the basic real-time theory and the added methodology to be able to work with design of new systems. So in reality for an experienced engineer it will take about one week including the training course to be productive, from the model and methodology point.

Carrier:

The success of this transfer is mainly because one person, that worked in the research group where the ideas where developed, started to work as a consultant for TietoEnator ArosTech at VCE. Regardless how many good reports we write we need people that carries the results [Dal94]. A related example is the development of the control system for Volvo S80 where Ken Tindell and others carried the response time analysis for the CAN bus into a tool and implanted that tool into Volvo Car Cooperation organization [Cas98, Mel98].

Citation: "Tech transfer is a contact sport. People not paper transfer technology" [Fol96].

Adapters:

Even if we have carriers we need early adapters at the company that take the technology into the company and its organization. These people need to be authoritative to be able to sell the new technology in the organization. There is always a healthy conservatism in all organization. Therefore one must find people that are ready to invest enough time and energy to find out if the technology is applicable and gives an added value to the development of their products or not [Ben96].

Finding 6: The major problems when introducing real-time technology in an organization is to change the requirements caption process to include timing requirements.

Motivation:

Several independent sources have given the same statement (Volvo Car and Volvo Construction Equipment). Especially since all engineering disciplines within a company has to change their way of specifying requirements on the electronics. The main problem is that once a timing requirement for a high-level function has been derived, it is very hard to reconsider it later on. It seems that a timing requirement becomes more and more truthful the older the timing constraint becomes. This really comes to the surface when a new function is added and the schedulabilty test is negative depending on that the utilization of the system is too high. To add this function anyway you need to find either execution time budgets that are too generous or timing requirements that are too strict. Assuming the overestimation of execution times is neglectable, the timing requirements have to be reconsidered. To find out which timing requirements that have to be relaxed there must exist a notion of confidence of the timing requirements. As an example, the time from pressing a particular lamp switch until the light is turned on should it take 200 ms or 300 ms, if the requirements say that the confidence in specifying 200 ms is low this timing requirement could be considered to be relaxed. Thus the results from the requirements caption process must be clearly expressed and well motivated since it will be used during the complete life cycle of the system.

6.3 Technical issues

Finding 7: The task model used (described in Section 2) is in some cases too restricted when handling control jitter for simple controllers and especially for multirate controllers.

Motivation

The *limited expressiveness* in the used task model is related to the jitter problem and multirate communication problem. Specifying release times and deadlines of the tasks involved in the computation can be used to fulfil for example jitter requirements. However, this is a problem since the engineer has to distribute the release times and deadlines at the timeline to not overload a specific window of the timeline. This means that the engineer has to act as a pre scheduler to the scheduler, which is not efficient. Instead, a desired property of the task model would be to have the possibility to specify relative timing constraints. For example, a sampling task is required to run with a certain period time and have a tolerance of a specific amount (*Period time* ± *tolerance*). Relative timing constraints could also be used for specifying latency constraints, e.g., the time between sampling and actuation. Furthermore, when a controller consists of several entities that run with different period times, i.e., multirate control, one would also like to have the possibility to specify latency constraints. If the used task model supported this it would be much simpler to specify a system. Extending the task model is an easy task but to come up with useful tools to schedule a system based on such a task model is not an easy task.

Finding 8: Task model and scheduling techniques reported in literature has to be extended to take real-world requirements into consideration.

Motivation:

When the scheduling tool for this task model was developed we had to take several important aspects into account to be able to get a tool that utilized the resources of the target system efficient. The two aspects we will cover here are schedule representation and taking interrupt overhead into account when constructing the same schedule.

Schedule representation. A common representation of a static schedule is a vector, where one position in the vector represents a discrete point in time at which the execution of a task can start. The granularity of time has to be matched with the frequency of the periodic clock that drives the dispatcher, which will execute the tasks according to the schedule. If the execution time of a task is less than this granularity, or if it exceeds a multiple of the granularity with a small fraction, then the utilization of the CPU resource will decrease. This because there will be time intervals that can not be used to execute tasks. An apparent solution to this is to increase the granularity (frequency) of the periodic clock. However, with a higher frequency of the clock the dispatcher will instead use more of the CPU resources, since it will execute more often.

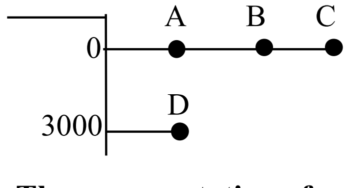


Figure 5: The representation of a schedule.

Another way of representing a schedule is as a list of rows, see Figure 5, where each row represents a point in time at which the dispatcher is to start the execution of a sequence of one or more tasks. The first task in this sequence, or *chain*, is started at the given point in time. All other tasks in the chain are started as soon as the preceding task in the sequence has completed its execution, without need for the clock to trigger the dispatcher. This representation will allow several tasks to be executed during an interval less than the period time of the dispatcher clock. Hence, the dispatcher overhead can be kept low at the same time, as the utilization of the CPU resource is high.

Interrupt overhead. Typically, pre-run-time scheduling does not account for interrupts, assuming their execution can be ignored or incorporated into task execution times. In many applications, the interrupts are, however, non-negligible and inclusion in task execution is too pessimistic and inefficient. Furthermore, as inter-arrival and execution times of interrupts are smaller than the granularity of the online dispatcher and the arrival times are unknown, interrupt-handling routines cannot be modeled as pre scheduled tasks. The application of server algorithms, e.g., sporadic server [Spr89] total bandwidth server [Spu94], and slack stealing [Leh92] are not feasible due to the short response times that are required.

The key issue for static scheduling accounting for interrupts is the consideration of the overhead. If interrupts occur at run-time, interrupt-handling routines are executed. The delay this poses on task execution must be accounted for when the system is scheduled. Evidently, an inherent, minimum amount - the worst case penalty - to handle a worst case scenario has to be reserved, according to minimum inter-arrival times and execution times. Any amount exceeding this, however, is overhead imposed by the used method. It is this overhead that has to be kept small for efficient utilization of the processor. During this project we had to develop a method that handled interrupts in an efficient manner. This method combines a tree search algorithm with response time analysis, see the paper by Sandström et al [San98].

Finding 9: To make a pre-run-time scheduler tool really useful, feedback has to be provided to the user when the system is not schedulable.

Motivation

When applying scheduling in industrial projects, engineers are faced with a problem that only to a very limited degree has been attacked by the real-time research community, namely how to provide constructive feedback to the user in cases when a feasible schedule can not be found.

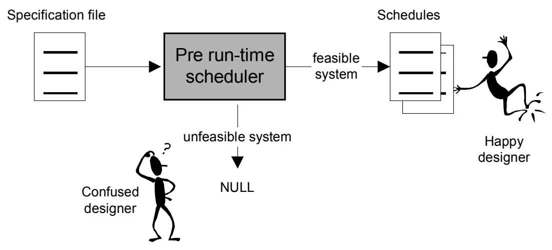


Figure 6: Pre run-time scheduling: present situation

This *limited feedback problem* leads to confused designers, as illustrated in Figure 6, which more or less at random have to optimize and modify the specification. However, to help the designer to come up with a specification for which the pre-run-time scheduler can find a feasible schedule, there is a need for heuristics that analyze the specification for semantic problems and give constructive feedback to the user. That is, the user should be provided hints to how the problem can be resolved, i.e., how the specification can be modified to allow the generation of a feasible schedule.

We have developed a method to provide feedback to the user by calculating a load function for the system. By identifying bottlenecks in the system specification we can guide the designer in modifying the input to the pre-run-time scheduler. The underlying hypothesis is that there is a correlation between the points in time when the load function has a high value, and the locality of the bottlenecks in the specification that leads to an infeasible schedule, [All96].

Finding 10: To minimize the verification effort when only small updates have been done to the application an incremental scheduling is needed.

Motivation

When an application has been tested and used in a vehicle for some time without any problems, the application is accepted and released. If then later some new functionality is added one wants to keep as much as possible of the execution order in the application to avoid major re-verification efforts.

This is not possible today, that is, when adding new functionality to the application a completely new schedule has to be generated. The major drawback of this approach is that the application verification and validation has to be completely redone to guarantee the functionality.

A desired feature of a scheduler would be to have the possibility to incrementally add new tasks to the application without affecting the already verified and unchanged part. A scheduler that takes both the updated design specification and the old verified schedule as input could solve this problem. The scheduler could try to find space in the old schedule for the new tasks or if not minimize the number of changes.

We believe that it is more important to keep the order than keeping the exact start times of the tasks, as long as the timing requirements are fulfilled.

We believe this because there often are margins in the execution windows for the tasks while a change of order could have severe impact for example on multirate transactions, which often are sensitive on data age.

7 Conclusion and Future research

We believe the presented project has been successful in transferring real-time technology from a university to an industrial partner. As a result, the industrial partner has adopted a more systematic and formalized design process, which has shortened the overall development cycle compared to similar previous projects. It also seems that the quality of the products has met the requirements.

However this transfer goes both ways, industry has also provided new relevant challenges for academia. An example of this is the limited expressiveness of the task model for real world constraints including specification of jitter constraints and specifying relative timing constraints (suited for multi rate control systems). It would be quite easy to extend the task model with such attributes, but the mapping of these to an implementation and the feasibility check, including schedule construction, is not a trivial task. Another example is the limited feedback problem of the static scheduler when it is unable to find a feasible schedule. There is a lot to gain if information can be given to the designer where to find the bottlenecks in the design and specification. The need of a incremental scheduler is also pointed out, which would be very useful when maintaining the application.

Remember: tech transfer is a contact sport, people not paper transfers technology!

Acknowledgements: We would like to thank Jack Stankovic, Hans Hansson, Sasikumar Punnekat, and Ivica Crnkovic for valuable discussions and for reviewing earlier versions of this paper. We would also like to thank Krithi Ramamritham for encouraging us to write this paper.

Mälardalen Real-Time research Centre (MRTC; www.mrtc.mdh.se) is a research centre in Västerås, Sweden, supported by Swedish industry, the Swedish Foundation for Knowledge and Competence Development (KK-stiftelsen) and Mälardalen University.

8 References

[Ben96] J. L. Bennett. Building Relationships for Technology Transfer. Communications of the ACM, Volume 39 Number 9. Sep. 1996.

[Spr89] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. The Journal of Real-Time Systems 1, 27-60 (1989)

[Leh92] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft aperiodic tasks in fixed priority preemptive systems. In Proc. IEEE Real-Time Systems Symposium. Dec. 1992.

[All96]B. Allwin, K. Sandström, and C. Eriksson. Constructive Feedback Turn Failure into Sucess for Pre-Run_time Schduled Systems. 11th Euromicro Workshop on real-time systems.

[Spu94] M. Spuri and G. C. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. In Proc. IEEE Real-Time Systems Symposium. Dec. 1994.

[San98] K Sandström, C. Eriksson, and G. Fohler. Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System. In Proceedings of the fifth International Conferance on Real-Time Computing Systems and Applications, pp. 158-165, October 1998. ISBN 0-8186-9209-X.

[Fol96] Jim Foley. Technology Transfer from University to Industry. Communications of the ACM, Volume 39 Number 9. Sep. 1996.

[Cas98] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg Volcano a revolution in on-board communications. Volvo Technology Report. 98-12-10.

[Mel98] K. Melin. Volvo S80: Electrical system of the future Volvo Technology Report. 98-12-11.

[Bos00] J Bosch. Design and Use of Software Architectures, Adopting and Evolving a Product-Line Approach Addison Wesley. ISBN 0-201-67494-7, June 2000 (forthcoming)

[Kal88] D. Kalinsky and J. Ready. Distinctions between requirements specification and design of real-time systems. Conference proceedings on TRI-Ada '88 , 1988, Pages 426 – 432.

[Dal94] M Dalziel. Effective university-industry technology transfer. Canadian Conference on Electrical and Computer Engineering, 1994 , Conference Proceedings, Page(s): 743-746 vol.2

[Bat99] I Bate and A. Burns. An Approach to Task Attribute Assignment for Uniprocessor Systems. Proceedings of the 11th Euromicro Conference on Resal-Time Systems, York, England, UK, June, 1999

[Sch96] J. Scholtz. Technology Transfer through Prototypes. Communications of the ACM, Volume 39 Number 9. Sep. 1996.

[Ram90] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In 10th Int. Conf. on Distributed Computing Systems, pages 108-115, 1990.

B

Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System

by

Kristian Sandström, Christer Norström, Gerhard Fohler

Mälardalen Real-Time Research Centre, Department of Computer Engineering, Mälardalen University, Västerås, Sweden

In Proceedings of the fifth International Conferance on Real-Time Computing Systems and Applications, October 1998.
IEEE Computer Society.

Handling Interrupts with Static Scheduling in an Automotive Vehicle Control System

Kristian Sandström, Christer Norström, Gerhard Fohler

Mälardalen Real-Time Research Centre, Department of Computer Engineering, Mälardalen University, Västerås, Sweden

Abstract

The requirements of industrial applications only rarely permit the exclusive use of single paradigms in the development of real-time systems. Product cost, reuse of existing solutions, and efficiency require diverse, or even opposing methods to coexist or to be integrated. In this paper, we deal with one problem encountered during the development of a real-time system for motion control in automotive vehicles, the integration of static scheduling and interrupts. The user mandates pre run-time scheduling for a number of reasons, e.g., predictability, testability and low run-time overhead. However, the interrupt overhead can not be ignored in a safety critical system, and therefore has to be accounted for when creating a static schedule. We propose a method that combines static scheduling and run-time interrupts by applying standard static scheduling techniques and exact analysis. The appropriateness of this method is underlined by successful industrial deployment.

1 Introduction

Requirements of industrial applications only rarely permit the exclusive use of single paradigms in the development of real-time components. Product cost, reuse of existing solutions, and efficiency require diverse, or even opposing methods to coexist or to be integrated. In this paper, we deal with one problem encountered during the development of a real-time system for motion control in automotive vehicles, the integration of static scheduling and interrupts.

This method of managing interrupt and static scheduling has been applied in industry, for development of the control system for automotive vehicles. The real-time system managing the motion of the automotive vehicle consists of two single micro controller units and redundant busses. The application tasks are assumed to be precedence constrained, allow additional synchronisation via mutual exclusion, and to communicate with tasks of the same and different period times. Communication between interrupt routines and application tasks is indirect via memory only, i.e., there is no direct, synchronised communication. Minimum inter-arrival times for interrupts are known but exact points in time for arrivals are unknown. The particular problem to be solved is to construct static schedules, i.e., start times and completion times for tasks in such a way, that interrupts are accounted for in a safe and efficient way. We will discuss the concrete industrial requirements leading to this problem and our general solution in this paper.

The user mandates pre run-time scheduling for a number of reasons, e.g., predictability, testability and low run-time overhead. However, various hardware architectures rely on the use of interrupts, e.g., to read sensor data and manage bus messages. This company had used a time-triggered architecture for many years. As the constructed systems where quite small and simple, the pre run-time schedules where hand crafted. In this new version of the control system the functionality and complexity has increased dramatically. The consequence of this was that the engineers needed the help of tools for schedule construction. The first version of the tool created the schedules but did not take the effects of interrupts into account. The interrupt overhead was non-negligible and as a result there were a lot of deadline violations during run-time. The first attempt to solve this problem by the engineers was to increase the worst-case execution time (WCET) of the tasks by multiplying it by a constant. The result was that fewer, but not all, tasks missed their deadlines. It was not possible to increase this constant so that all tasks met their deadline and still have a schedulable system. The second attempt was to increase the WCET for each task by the worst-case interrupt interference. No task would miss its deadline with this approach, but it was impossible to find a feasible schedule for the system due to the pessimistic assumption. Therefore an algorithm that more exactly calculates the worst-case interrupt interference for a given task set was required.

Typically, static scheduling does not account for interrupts, assuming their execution can be ignored or incorporated into task execution times. In a number of applications, the interrupts are, however, non-negligible and inclusion in task execution is too pessimistic and inefficient. Furthermore, as inter-arrival and execution times of interrupts are smaller than the granularity of the online dispatcher and the arrival times are unknown, interrupt-handling routines cannot be modelled as pre scheduled tasks. The application of server algorithms, e.g., sporadic server [4] total bandwidth server [6], and slack stealing [5] are not feasible due to the short response times that are required. The same holds for slot shifting [1], which, in addition, is only reactive, i.e., it is applied to existing schedules, instead of providing guarantees along with schedule construction. Ramamritham has proposed an algorithm that distributes resources during schedule construction [2], but does not give guarantees either.

The key issue for static scheduling accounting for interrupts is the consideration of the overhead. If interrupts occur at run-time, interrupt-handling routines are executed. The delay this poses on task execution is accounted for feasibly in the schedule beforehand. Evidently, an inherent, minimum amount - the worst case penalty - to handle a worst case scenario has to be reserved, according to minimum inter-arrival times and execution times. Any amount exceeding this, however, is overhead imposed by the used method. It is this overhead that has to be kept small for efficient utilisation of the processor. The naive approach of adding a worst case penalty to every task, each may be ”hit” by a worst case interrupt arrival, is overly pessimistic. In this industrial application, the naive approach results in a utilisation of **140.1** %. Instead, our algorithm eliminates consideration of unnecessary penalties by utilising information about task execution behaviour at run-time for a given schedule. We provide analysis to be used during schedule construction, as well as, being applied to already constructed schedules.

Our algorithm enables the co-existence of the seemingly adverse paradigms of static scheduling and interrupts. It determines interrupt overhead during schedule construction in an efficient way and allows the analysis of existing schedules for feasible interrupt handling. The described algorithm is successfully deployed as part of a commercial toolkit for design of embedded real-time systems.

The rest of this paper is organised as follows: section 2 describes the industrial application and its requirements and section 3 describes the task model and the representation of a schedule. The proposed algorithm and its applicability are described in section 4. Section 5 summarises the paper.

2 Application characteristics

The application is a vehicle control system with high demands on safety, reliability, and timeliness. The hardware in the system consists of a number of nodes that are connected via redundant buses. The application contains tasks, running at different period times, which collaborate to perform a certain control function. The system contains about 80 tasks with well-defined functionality allocated to two nodes. Each node is very I/O intensive. The complete system has about 150 I/O channels connected to it.

The application contains tasks with three different period times: 10 milliseconds, 50 milliseconds, and 100 milliseconds. A few tasks require longer period times, in these cases a task is executed with a period time of 100 milliseconds and the actual period time is controlled by an internal counter. The reason for this construction has been to keep the size of the schedule at an appropriate level. The execution times of the tasks in the application range from about 10 μs to 1 millisecond. To be able to fulfil the jitter requirements a clock tick resolution of one millisecond is used. The requirements on jitter for I/O are between 2 and 20 milliseconds depending on the rate of data. Due to the fact that a majority of all tasks have an execution time that is less than one clock tick it is of vital importance that the scheduler can schedule several tasks within one clock tick. As a consequence the dispatcher has to support switching between these tasks without the occurrence of a clock tick (see Section 3).

The application is, due to the construction of the hardware, quite interrupt intensive. As this application has a number of interrupts and the effect these interrupts have on the timing of the tasks has to be taken into account when constructing the schedule. Hence, the minimum inter-arrival time and duration of the interrupts are specified in the design. The minimum inter-arrival times for the interrupts are 250 µs, 500 µs, and 1 millisecond.

The worst-case utilisation of the processors for the critical part is around 80%. Divided into 35% for interrupts and 45% for application tasks. The spare capacity left is used by soft real-time tasks. During run-time the spare capacity will be more than the remaining 20% if the load is less than the worst case.

3 Task model and run-time representation

The task model allows a number of requirements to be expressed. For each task the following temporal requirements have to be specified:

- Period time
- Release time (relative the start of the period)
- Deadline (relative the start of the period).
- Worst case execution time

Relations between tasks can be specified by:

- Precedence relationships
- Mutual exclusion relationships (shared resources)
- Communication
  - Synchronous communication (communication between task that have same period time)
  - Asynchronous communication, i.e., communication between tasks running with different period times. The semantics of asynchronous communication is that messages are transferred with the lowest frequency of the sender and the receiver

In addition, all interrupts have to be specified by minimum inter arrival time and worst case execution time of the interrupt routine.

When creating a schedule, not only these requirements have to be taken into account, but also the run-time representation of the schedule. A common representation of a static schedule is a vector, where one position in the vector represents a discrete point in time at which the execution of a task can start. The granularity of time has to be matched with the frequency of the periodic clock that drives the dispatcher, which will execute the tasks according to the schedule. If the execution time of a task is less than this granularity, or if it exceeds a multiple of the granularity with a small fraction, then the utilisation of the CPU resource will decrease. This because there will be time intervals that cannot be used to execute tasks. An apparent solution to this is to increase the granularity (frequency) of the periodic clock. However, with a higher frequency of the clock the dispatcher will instead use more of the CPU resources, since it will execute more often.

Another way of representing a schedule is as a matrix, where each row represents a point in time at which the dispatcher is to start the execution of a sequence of several tasks. The first task in this sequence, or *chain*, is started at the given point in time. All other tasks in the chain are started as soon as the preceding task in the sequence has completed its execution, without need for the clock to trigger the dispatcher. This representation will allow several tasks to be executed during an interval less than the period time of the dispatcher clock. Hence, the dispatcher overhead can be kept low at the same time as the utilisation of the CPU resource is high.

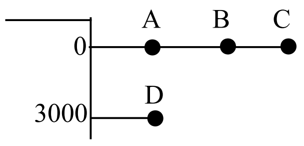


Figure 1. The representation of a schedule.

The chains in the matrix are ordered with ascending start times. In Figure 1 we see two chains: one starting at time zero including task A, B and, C, and the other starting at time 3000, including only task D. Task B and C is defined as *chain successors* to A. If pre-emption of tasks is allowed, the schedule can be constructed so that a chain can be pre-empted by another chain that has a later start time. As soon as the last task in that chain completes, the pre-empted chain will resume its execution. It is the task of the static scheduler to construct chains that unconditionally will fulfil all the requirements put on the task set.

The latter representation is used by the run-time system that is used together with the method described in this paper.

The following rules apply to the construction of chains:

- The start time of a chain has to be a multiple of the OS clock tick.
- Consecutive chains have ascending start times.
- The earliest start time of a task is the start time of the chain. The reason for this is that the minimal execution times are unknown, and therefore is assumed to be zero.
- The latest completion time of a task *T* is the sum of the worst-case execution time (*WCET*) of all chain predecessors of the task, plus the *WCET* of the task itself. If the task *T* or any of its chain predecessors is pre-empted by another chain, the *WCET* of all the tasks of that chain have to be taken into account when calculating the scheduled completion time for task *T*.
- A task *T* might be pre-empted if the scheduled completion time of task *T* is greater than the start time of a succeeding chain. A succeeding chain is

a chain that has a start time greater than the start time of the chain that task *T* is part of.

- If pre-emption is not desired the schedule is constructed in such way that the last task of a chain always has a scheduled completion time that is less or equal to the succeeding chain's start time and thus pre-emption can not occur.

4 Algorithm

The algorithm has to introduce as little additional overhead as possible. One way of doing this, instead of penalising each task, is to make use of the fact that tasks are executed in chains. For example, if *n* tasks are executed after each other in the same chain they can, from an algorithmic perspective, be viewed as one single task. We can then calculate the completion time of the n'th task with exact analysis [3] for one task having an execution time of the sum of the *n* tasks. The critical instant will be at the start time of the chain and the interrupts is regarded as higher priority tasks. This is repeated for task n-1 disregarding task n and so on. The advantage of this approach is that we can have a single critical instant for all the *n* tasks. In this way, the total penalty for all tasks will be lower than the naive approach in most cases (never higher).

If pre-emption is considered things become more complicated. For a pre-empted chain we have to consider the delay introduced by the pre-emption. We also have to take into account the interference, from interrupts, put upon both the pre-empting chain and the pre-empted chain. With a critical instant, as before, at the start time of the chain, we can calculate which task that will be pre-empted, and where during its execution. We then insert the pre-empting chain at this point and continue calculating the interference on the rest of the chain including the pre-empting chain.

More precisely, the effect that the interference of an interrupt has on a specific task *T* can be split into two different cases that are of interest:

1. *The interrupt hits the task T or a task that is a chain predecessor.*

   Assume that task *T* is hit by an interrupt. The effect this will have on task *T's* completion time, will be the same as if task *T's* execution time would be prolonged with the execution time of the interrupt routine. As a consequence, the completion time of all tasks that are chain successors, to the task that were hit, will be delayed with the same amount. Conclusion; if task *T* or a task that is a chain predecessor is hit by an interrupt the *direct* worst-case delay is the *WCET* of the interrupt routine. See Figure 2.

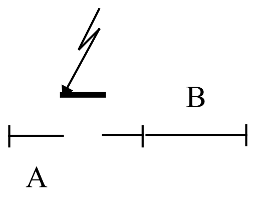


Figure 2. The interference of interrupts.

2. *The interrupt hits a task that is executing in a chain that has pre-empted task T or a chain predecessor to task T.*

   The chain that is hit by the interrupt will be affected according to case 1. This will delay the resume point of the pre-empted chain. Therefore task *T* and all its chain successors will be delayed by the *WCET* of the interrupt routine. Conclusion: the *direct* delay of task *T* will be the *WCET* of the interrupt routine.

   The term *direct* delay is used because a delay of a task could lead to a pre-emption that would not normally occur if the task were not delayed. See Figure 3.

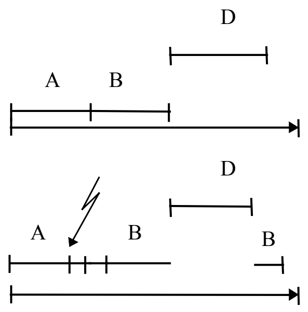


Figure 3. An interrupt may cause pre-emption.

It should be noted that in case 2, the pre-empting chain could in itself be pre-empted. Though, all such cases could be explored with a combination of case one and two. In all other cases, the task, which we considered, will not be affected.

4.1 Calculation of worst case completion time

Assuming that chains are enumerated and that tasks in a chain are enumerated in ascending order, with the first task in the chain numbered one. For a given task *i* in a chain *ch* the worst-case completion time, relative the start time of the chain, is then given by:

$$R_i = \sum_{n=1}^{i} C_n^{ch} + \sum_{\forall p \ni ic(R_i)} \sum_{m=1}^{nofTask(p)} C_m^{p} + \sum_{\forall_{\text{interrupt}}} \left\lceil \frac{R_i}{T_{\text{interrupt}}} \right\rceil C_{\text{interrupt}}$$

Where:

- C_b^a denotes the *WCET* of task *b* in chain *a*

- ic(R_i) denotes all chains *p* that conforms to:
  $(p \neq ch) \wedge startTime(ch) < startTime(p) \leq (R_i + startTime(ch))$
  This is all chains that pre-empt task *i* or any of its chain predecessors.

- *nofTask(p)* is the number of tasks in chain *p*.

To show whether this analysis is correct, a general chain configuration could be expressed as a set of tasks conforming to exact analysis. If, for this task set, the exact analysis could be written as the equation above, then this equation can rely on the proof of exact analysis [3]. This proof is available in the appendix.

4.2 Applying the algorithm

Below is a pseudo code description of an implementation of the algorithm. The objective is to calculate the completion time for a task *i*, residing in a chain *ch* with start time *t*. Given is a set of interrupts with minimal inter arrival time and *WCET*.

Let *TaskInterference* be the sum of the *WCET* of all tasks that interfere with task *i*. Tasks in chains with start time less than *t* are of no interest, because they do not interfere with task *i*.
Let *hChains* be all chains that have a start time greater than *t*.

```
1. TaskInterference = $WCET_i$ + $\sum$ WCET of all tasks
   that precede $task_i$ in chain ch.
2. $R_i^0$ = TaskInterference.
3. For every chain, $h_{chain}$, in hChains do
      If $R_i^n$ + t > startTime($h_{chain}$) then
         TaskInterference = TaskInterference +
            $\sum$ WCET of all tasks in $h_{chain}$
      $R_i^n$ = $R_i^n$ + $\sum$ WCET of all tasks in $h_{chain}$.
      Remove $h_{chain}$ from hChains
    EndIf
4. $R_i^{n+1}$ = TaskInterference + $\sum_{\forall_{interrupt}} \left\lceil \frac{R_i^n}{T_{interrupt}} \right\rceil C_{interrupt}$
   If $R_i^{n+1}$ + $t_{chain}$ > deadline($task_i$) then abort.
   Else If $R_i^{n+1}$ = $R_i^n$ and $R_i^{n+1}$ + $t_{chain}$ < start time of
   all chains in hChain then
         $R_i^{n+1}$+ t is the latest completion time of
   task i.
   Else go to step 3.
```

When the algorithm is incorporated into the scheduling phase, this algorithm is used while constructing the schedule. This means that some of the chains in step 3 might not exist from start when calculating the completion time of task$_i$. Instead they will ”appear” as the scheduling proceeds. If a new chain, that pre-empts task$_i$, is created the algorithm has to be applied to that chain in a hierarchical fashion. The scheduler tries to put as many tasks as possible in the same chain, as this will decrease the run-time system overhead.

As an example, assume the following task set:

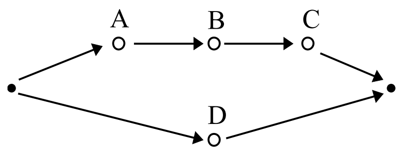


Figure 4. Precedence graph of the example.

The arrows in between tasks, in Figure 4, denote precedence relationships, the filled circles denotes start and end of the graph.

Task	WCET	Release time	Deadline	Period Time
A	2000	0	5000	5000
B	200	0	5000	5000
C	1000	0	5000	5000
D	800	3000	4000	5000

Table 1. Specification of tasks.

Interrupt	WCET	Minimum inter arrival time
Interrupt1	100	1000
Interrupt2	100	3000

Table 2. Specification of interrupts.

Assume that the scheduler uses pre-emption and earliest deadline as search heuristic. The clock tick is 1000. A schedule that does not consider the interrupt interference will then look like Figure 5.

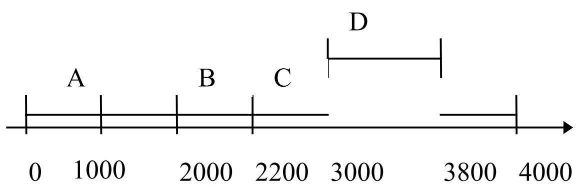


Figure 5. The schedule for the example.

If we calculate the worst case completion time for each task, with the analysis presented, the completion times of the tasks will be:

$$R_A^1 = 2000 + 0 + \left\lceil \frac{2000}{1000} \right\rceil 100 + \left\lceil \frac{2000}{3000} \right\rceil 100 = 2300$$

$$R_A^2 = 2000 + 0 + \left\lceil \frac{2300}{1000} \right\rceil 100 + \left\lceil \frac{2300}{3000} \right\rceil 100 = 2400$$

$$R_A^3 = 2000 + 0 + \left\lceil \frac{2400}{1000} \right\rceil 100 + \left\lceil \frac{2400}{3000} \right\rceil 100 = 2400$$

Task *A* is the first task in the chain and the first term will therefore be equal to the wcet of *A*. There is no task pre-empting *A*, hence, the second term is zero. The total worst-case interference from the interrupts is 400 and the worst-case completion time of *A* is 2400.

$$R_B^1 = (2000 + 200) + 0 + \left\lceil \frac{2200}{1000} \right\rceil 100 + \left\lceil \frac{2200}{3000} \right\rceil 100 = 2600$$

$$R_B^2 = (2000 + 200) + 0 + \left\lceil \frac{2600}{1000} \right\rceil 100 + \left\lceil \frac{2600}{3000} \right\rceil 100 = 2600$$

The first term for task *B* is the wcet of task *A* and *B*. Note that task *B* is not “affected” by any interference from the interrupts. This does not mean that an interrupt could not pre-empt *B*, but if it does, the completion time of *A* will be less than the worst case and therefor task *B* will have an earlier start time.

$$R_C^1 = (2000 + 200 + 1000) + 800 + \left\lceil \frac{4000}{1000} \right\rceil 100 + \left\lceil \frac{4000}{3000} \right\rceil 100 = 4600$$

$$R_C^2 = (2000 + 200 + 1000) + 800 + \left\lceil \frac{4600}{1000} \right\rceil 100 + \left\lceil \frac{4600}{3000} \right\rceil 100 = 4700$$

$$R_C^3 = (2000 + 200 + 1000) + 800 + \left\lceil \frac{4700}{1000} \right\rceil 100 + \left\lceil \frac{4700}{3000} \right\rceil 100 = 4700$$

Since task *C* clearly will be pre-empted by *D*, the second term will be the sum of wcet of all tasks in the pre-empting chain, i.e., the wcet of task *D*.

$$R_D^1 = 800 + 0 + \left\lceil \frac{800}{1000} \right\rceil 100 + \left\lceil \frac{800}{3000} \right\rceil 100 = 1000$$

$$R_D^2 = 800 + 0 + \left\lceil \frac{1000}{1000} \right\rceil 100 + \left\lceil \frac{1000}{3000} \right\rceil 100 = 1000$$

Task *D* is not affected by any interrupt interfering with the chain *ABC*, since its start time is absolute and not dependent on A, B, and C. The worst case completion time of *D* is 1000 + 3000 = 4000.

The utilisation in this example is 94%. The naive approach of including interrupt overhead into task execution times will result in an utilisation of 102%.

5 Conclusions

In this paper we presented methods to combine static scheduling and online interrupt handling in the real-time system controlling the motion of vehicles. We have described the real-world application and derived specific requirements. Meeting these and consideration of cost and efficiency necessitate the use of interrupts.

We propose analysis that allows the processing demand of online interrupt requests to be taken into account during schedule construction, i.e., into the timing of task chains. The naive approach of including interrupt overhead into task execution times is prohibitively inefficient. Rather, our analysis limits the amount of penalty to be included for runtime interrupt handling, by identifying task chains affected by worst case interrupt arrival. In this industrial application, our methods result in a schedule size[2] of **74.5** % of *LCM,* to be compared with **140.1** % using the naive approach. A lower bound would be to summarise the *WCET* of all tasks and perform exact analysis on that sum. In this case we would get a schedule size of **72.9** % of *LCM*. This is the number that our method would result in if all tasks would execute in one single chain. Though, this is not possible with the actual specification.

The resulting static schedules allow the coexistence of the seemingly conflicting paradigms of offline schedule construction and online interrupt handling in an efficient way. The appropriateness of our approach is underlined by its successful use in automotive vehicles.

6 References

[1] 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, Dec. 1995.

[2] K. Ramamritham, G. Fohler, and J.-M. Adan. Issues in the static allocation and scheduling of complex periodic tasks, In Proc. 10th IEEE Workshop on Real-Time Operating Systems and Software, NY, USA, May 1993.

[3] M. Joseph and P.K. Pandya. Finding response times in a real-time system. Comp. J., 29(5). 1986.

[4] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. The Journal of Real-Time Systems 1, 27-60 (1989)

[5] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft aperiodic tasks in fixed priority preemptive systems. In Proc. IEEE Real-Time Systems Symposium. Dec. 1992.

[2] Schedule size is defined as the percentage of time of the LCM schedule that is allocated to the tasks. Interrupts that occur at instants during the LCM where no tasks are scheduled are not included in the schedule size. Therefor the schedule size can be lower than the utilisation is not strange.

[6] M. Spuri and G. C. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. In Proc. IEEE Real-Time Systems Symposium. Dec. 1994.

Appendix

In this appendix we show how a general chain structure can be expressed as a set of tasks conforming to the exact analysis theory. For this task set we will show that the exact analysis can be re-written as the presented formula for analysing interrupt interference.

Definition: A chain predecessor to a task *i* is any task that is scheduled to execute before task *i* and is allocated to the same chain.

To calculate the worst case response time for a task *i*, assume the following:

1. A critical instant at the start of the chain that task, *i*, reside within. Any task in a chain is *ready* to execute at the chain start time. If all chain predecessors and all pre-empting tasks execute for zero amount of time, then the task will start its execution at the start time of the chain.
2. The interference that we have to consider is
   - Tasks that precede task *i* in the same chain.
   - Tasks that pre-empt task *i* or any of the chain predecessors to task *i*.
   - Interrupts.
3. All tasks have a period time equal to *LCM*. Task that have a shorter period time, in the specification, than *LCM* will be replaced by *LCM*/period time number of instances of that task. Each instance in considered on its own in the schedule. These instances will be referred to as separate tasks.
4. All task have a deadline less or equal to *LCM* (because all task have deadline less or equal to the period time)
5. Interrupts have arbitrary period times, with known minimal inter arrival time.

To calculate the worst case response time for a task *i*, using exact analysis, we have to know which tasks that have a higher priority than task *i*, hp(*i*). For task *i*, let:

A. All chain predecessors to task *i* be in the set of higher priority tasks. Exact analysis assumes a critical instant with all tasks ready at a specific point in time [3]. Hence, the chain predecessors could be modelled as higher priority task in descending order of priority.

B. All tasks that might pre-empt task *i* or any of its chain predecessors be in the set of higher priority tasks. The critical instant is the worst case, i.e., if a task *i* will fulfil its deadline for a critical instant it will meet its deadline for all other cases to [3]. Thus, if a higher priority task will be ready at a later time, e.g., at the time of the start of a pre-empting chain, task *i* will still meet its deadline. Therefore all of these tasks can be modelled as higher priority tasks.

C. All interrupts be in the set of higher priority tasks. Interrupt can be modelled as tasks with higher priority than tasks in A and B and with a period time equal to the minimal inter arrival time of the interrupt.

Exact analysis gives us:

$$(1)\ R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Equation (2) divides the sum of all higher priority tasks in to two sums. Where task*(i)* is the tasks relating to A and B above, the second is the interrupts.

$$(2)\ \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \leftrightarrow \sum_{\forall j \in task(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \sum_{\forall k \in \text{interrupt}} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

Consider the first sum in equation (2), $\sum_{\forall j \in task(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$. This contains all task that apply to A and B. According to assumption 3, all these tasks have a period time equal to *LCM*. All tasks have a deadline less or equal to *LCM*, i.e., assumption 4. From this follows that the response time has to be less or equal to *LCM*. If the response time is greater than *LCM* we can stop the calculation[3], because the deadline is missed. This gives:

$$(3)\quad \mathrm{T_j} = LCM \wedge \mathrm{R_i} \le LCM \Rightarrow \frac{R_i}{T_j} \le 1 \Rightarrow$$

$$\Rightarrow \left\lceil \frac{R_i}{T_j} \right\rceil = 1 \Rightarrow \left\lceil \frac{R_i}{T_j} \right\rceil C_j = C_j \Rightarrow \sum_{\forall j \in task(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j = \sum_{\forall j \in task(i)} C_j$$

This simply says that the tasks will only execute once during *LCM*, which is what was stated in assumption 3. Inserting the result of equation (2) and equation (3) in equation (1) yield:

$$(4)\ R_i = C_i + \sum_{\forall j \in task(i)} C_j + \sum_{\forall k \in \text{interrupt}} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

The sum of all tasks can be divided in to two sums. The sums of all chain predecessors to task *i*, according to A, and the sum of all tasks that pre-empt task *i* or any of its chain predecessors, according to B.

$$\sum_{\forall j \in task(i)} C_j \leftrightarrow \sum_{\forall l \in pred(i)} C_l + \sum_{\forall m \in prem(i)} C_m$$

Where pred(*i*) is the chain predecessors according to A and prem(*i*) is the tasks in B. If this is substituted in equation (4), this leads us to:

[3] In many cases the calculation could be stopped before the response time reaches *LCM*. This because the deadline can be smaller than *LCM* and a start time greater than zero.

$$(5)\, R_i = C_i + \sum_{\forall l \in pred(i)} C_l + \sum_{\forall m \in prem(i)} C_m + \sum_{\forall k \in \text{interrupt}} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

We will show that the worst-case completion time analysis formula, presented in section 5.1, is equivalent to equation 5.

$$(6)\ R_i = \sum_{n=1}^{i} C_n^{ch} + \sum_{\forall p \ni ic(R_i)} \sum_{m=1}^{nofTask(p)} C_m^p + \sum_{\forall_{\text{interrupt}}} \left\lceil \frac{R_i}{T_{\text{interrupt}}} \right\rceil C_{\text{interrupt}}$$

The two first terms in equation (5) describes all chain predecessors to task *i* and task *i*. This is equal to the first term in equation (6).

$$(7)\, C_i + \sum_{\forall l \in pred(i)} C_l \leftrightarrow \sum_{n=1}^{i} C_n^{ch}$$

The third term in equation (5) is all tasks that pre-empt task *i* or any chain predecessor of task *i*. The second term in equation (6) is all tasks in all succeeding chains that has start time less than the response time of task *i*, i.e., all tasks that pre-empt *task i* or any of its chain predecessors. Thus, the two terms are equal.

$$(8)\ \sum_{\forall m \in prem(i)} C_m \leftrightarrow \sum_{\forall p \in ic(R_i)} \sum_{m=1}^{nofTask(p)} C_m^p$$

The last term in equation (5) and the last term in equation (6) are the same.

$$(9)\ \sum_{\forall k \in \text{interrupt}} \left\lceil \frac{R_i}{T_k} \right\rceil C_k \leftrightarrow \sum_{\forall_{\text{interrupt}}} \left\lceil \frac{R_i}{T_{\text{interrupt}}} \right\rceil C_{\text{interrupt}}$$

If the terms in equation (5) that are equal to terms in equation (7) to equation (9) is substituted with these, then we have the following equation.

$$R_i = \sum_{n=1}^{i} C_n^{ch} + \sum_{\forall p \ni ic(R_i)} \sum_{m=1}^{nofTask(p)} C_m^p + \sum_{\forall_{\text{interrupt}}} \left\lceil \frac{R_i}{T_{\text{interrupt}}} \right\rceil C_{\text{interrupt}}$$

That is, equation (6). This gives that equation (5) and equation (6) are equal.

Relaying on the proof [3] of exact analysis the analysis in equation (6) is correct.

C

Managing Temporal Constraints in Control Systems

by

Kristian Sandström and Christer Norström

Mälardalen Real-Time Research Centre, Department of Computer Engineering, Mälardalen University, Västerås, Sweden

Mälardalen Real-Time Research Centre, MRTC Technical Report 02/45

Managing Temporal Constraints in Control Systems

Kristian Sandström and Christer Norström

Mälardalen Real-Time Research Centre, Department of Computer Engineering, Mälardalen University, Västerås, Sweden

Abstract

Design and implementation of motion control applications include the mapping of control design to real-time system implementation. Important parameters from control design include deviation from nominal period time of an activity, end-to-end timing constraints, temporal correlation between different sampling tasks, and constraints on temporal variations in output. These parameters should also be considered in the real-time systems design, since translating them to simple deadlines may lead to sub-optimal solutions. Many real-time systems in industry today are based on pre-emptive priority based run-time systems, and hence, it is highly desirable to fulfill the temporal requirements by correctly assigning attributes such as priorities and offsets to the tasks executing in such systems. However, this is a non-trivial mapping, which should be supported by appropriate methods and tools. In this paper we propose a method, which by assigning priorities and offsets to tasks provides guarantees that complex timing constraints are met. The method handles periodic and sporadic tasks, shared resources, and varying execution times of tasks. We present the method, which uses a genetic algorithm, together with simulation results, showing that the proposed method is capable to efficiently handle complex constraints on task sets of realistic sizes covering most embedded control systems.

1 Introduction

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

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

Bate and Burns [2], propose a related method for assigning offsets and priorities to a fixed priority pre-emptive task set. They define a specification model that allows for expressing similar constraints as defined in Section 2 of this paper. However, their method does not consider the use of shared resources between sporadic and periodic tasks. Furthermore, attribute assignment for dealing with constraints on period time variation is managed using a heuristic algorithm with local optimization that, according to the authors, can lead to attribute assignments causing unschedulable systems when a feasible solution exists. For this reason it is difficult to extend the method to incorporate the additional constraints we would like to consider. Several researchers have attacked the same problem by generating off-line schedules [3][4][5]. A major disadvantage of such a solution is that it cannot be handled by a standard priority-based RTOS. Furthermore, they do not support pre-emption, sporadic activities, and varying task execution times. In [6][7] a method is presented for translating off-line schedules to task attributes for fixed priority systems (FPS). This method could be combined with methods in [3][4][5] and would enable the use of priority-based RTOS for those methods. However, the combined approached would still inherit the limitations of not supporting pre-emption, sporadic activities, and varying task execution times when searching for a solution to the complex constraints. The method allows for using on-line acceptance test for sporadic and aperiodic tasks to use spare capacity from the tasks translated from the off-line schedule to FPS, while the methods presented in this paper add no run-time overhead for managing sporadic tasks. Moreover, when translating an off-line schedule the method in [6][7] in some cases have to represent an off-line scheduled task by more than one FPS task. This can result in an FPS system with artificial tasks and thereby a greater number of tasks compared to our method. However, if more instances are used it is possible to find solutions that cannot be found otherwise. It is possible using the method presented in this paper to include more than one instance for some or all tasks.

In [8] the authors present a design methodology for real-time systems with end-to-end timing constraints, temporal correlation between different sampling tasks and constraints on temporal variations in output. The methodology derives period times, deadlines, and offsets for the tasks.

However, the task model does not agree with a standard priority-based RTOS and constraints on period time variation cannot be expressed. Furthermore, the method assumes that task execution times are static. The work presented in [9] uses genetic algorithms for minimizing jitter in communication using field busses. The problem solved is quite different from the one presented in this paper in that only jitter is minimized and messages do not have interrelated temporal constraints

The motivation for the work presented in this paper mainly originates from our participation in a real industrial project where we used a specification model with support for periodic tasks, deadlines, precedence relationships, mutual exclusions, and offsets [10]. By using this model we can express all timing constraints required by the application. However, the designer has to manually translate the timing constraints into attributes of the used model. This is possible for simple systems, but in systems with many such requirements it becomes very difficult to assign these attributes manually. Even if the designer succeeds in finding a feasible mapping, we get a maintenance problem [10].

In this work we use an enhanced specification model that supports temporal dependencies between tasks. We will show that we can solve the problem of mapping a system described by this specification model to a run-time system model in an efficient way by using a genetic algorithm (GA). There are several reasons for using the GA approach. GA is a general optimisation method that has been used successfully for solving a wide variety of complex problems including scheduling, e.g., in [11][12][13][14]. It can also easily be extended to optimise on other attributes such as minimising the response time of handling an event. One of the most important properties of the GA is its ability to deliver a result that fulfils a subset of the timing constraints in cases where it is impossible to fulfil all constraints. This information is important since the designer then can get an indication of which constraints that can not be fulfilled and thereby simplify the re-modelling of the application. Also, even if not all timing constraints are fulfilled, the application requirements may in some cases still be fulfilled since the robustness of the control design can tolerate deviations from the specification. However, this has to be verified by control analysis. Simulation results show that our algorithm performs well compared to the algorithm presented in [2] and that it finds solutions to a high degree when the considered systems are schedulable.

Thus, the contributions of this paper are:

- A specification model for describing systems with complex timing constraints.
- A synthesis algorithm that assigns priorities and offsets to tasks to fulfil the timing constraints given our specification model.
- Simulation results showing the efficiency of the suggested method.

The rest of this paper is organised as follows. Section 2 describes the used system model. The method for attribute assignment is covered in Section 3. In Section 4 simulation results are presented, followed by the conclusion of the paper in Section 5. An extensive example of the method can be found in Appendix A.

2 System model

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

2.1 Run-time system model

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

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

2.2 Specification model

The specification model defines the information that has to be specified for each periodic and sporadic task, as well as the constraints that can be expressed on a task set. A periodic task is defined by its worst-case execution time (WCET), best-case execution time (BCET), and nominal period time. The nominal period time is the desired rate at which the task should be executed. Below, the terms start time, completion time, and release are used. The start time of a task is the actual time when the first instruction is executed on the processor, as opposed to release, which is the time when the task becomes ready to execute. The completion time of a task is the time when the last instruction has been executed.

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

- Deadline – deadline relative to the release of the task.
- Precedence – constraint specifying the execution order between two tasks.
- Separation – constraint of a minimum distance between the completion of one task and start of another task.
- Jitter – the maximum allowed deviation from the nominal period time. Jitter constraints are used to control the deviation from nominal period for e.g., sampling and actuation activities.

    - Start Jitter – maximum allowed deviation of a task's start time from its nominal period time. The constraint is specified by an upper (S_h) and lower bound (S_l) on the time between two consecutive executions of a task.
    - Completion Jitter – maximum allowed deviation of a task's completion time from its nominal period time. The constraint is specified by an upper (C_h) and lower bound (C_l) on the time between two consecutive executions of a task.
- Latency – constraint specifying a maximum allowed distance between the start of one task and the completion of another task.
- Correlation – constraint on the maximum time between executions of two or more tasks executing in parallel. This constraint is used to correlate concurrent sampling or actuation activities in time.
- Shared resources – specification of the tasks that use semaphores and times for the tasks critical sections.

Periodic tasks may have a varying execution time and phasing relative to each other and hence the start time and completion time for a task can vary. This must be considered when finding an attribute assignment meeting the constraints for a task set. Therefore the analysis of a task set is performed for all instances over the least common multiple (*lcm)* of tasks period times. This is necessary since calculation of the earliest- and latest start time and completion time without considering all instances would be too pessimistic. As an example, consider the tasks τ_1 and τ_2 depicted in Figure 1. The tasks have a period time of 10 and only one constraint, a precedence constraint between τ_1 and τ_2. Assume a completion time of τ_1 equal to 3, and a start time of τ_2 equal to 4 in one period and a completion time of τ_1 equal to 2, and a start time of τ_2 equal to 2 in the next period. The latest completion of t_1 relative to the period is 3 and the earliest start relative to the period for τ_2 is 2, i.e., the precedence is violated considering only the task timing while if the separate instances are considered one can see that precedence is achieved between τ_1 and τ_2.

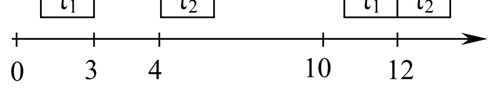


Figure 1. The execution of the two tasks τ_1 and τ_2.

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

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

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

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

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

Deadline <*deadline*, τ_i> holds iff

$$lct(\tau_i^n) - (periodTime(\tau_i) * n + offset(\tau_i)) \leq deadline$$

Precedence <τ_i, τ_j> holds iff

$$lct(\tau_i^n) \leq est(\tau_j^n)$$

Separation <*separation*, τ_i, τ_j> holds iff

$$est(\tau_j^n) - lct(\tau_i^n) \geq separation$$

Start Jitter <S_h, S_l, τ_i> holds iff

$$lst(\tau_i^{n+1}) - est(\tau_i^n) \leq S_h \wedge est(\tau_i^{n+1}) - lst(\tau_i^n) \geq S_l$$

Completion Jitter <C_h, C_l, τ_i> holds iff

$$lct(\tau_i^{n+1}) - ect(\tau_i^n) \leq C_h \wedge ect(\tau_i^{n+1}) - lct(\tau_i^n) \geq C_l$$

Latency <*latency*, τ_i, τ_j> holds iff

$$\begin{aligned}
&T_i = T_j \rightarrow \forall n(lct(\tau_i^n) \leq est(\tau_j^n) \wedge lct(\tau_j^n) - est(\tau_i^n) \leq latency) \wedge \\
&T_i > T_j \rightarrow \forall n \exists m : lct(\tau_i^n) \leq est(\tau_j^m) \wedge lct(\tau_j^m) - est(\tau_i^n) \leq latency \wedge \\
&T_i < T_j \rightarrow \forall n \exists m : lct(\tau_i^m) \leq est(\tau_j^n) \wedge lct(\tau_j^n) - est(\tau_i^m) \leq latency
\end{aligned}$$

Correlation <*Correlation*, τ_i, $\tau_{i+1}, \ldots,$ τ_{i+m}> holds iff

$$\forall j,k \in i..(i+m) : \left| lst(\tau_j^n) - est(\tau_k^n) \right| \leq Correlation$$

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

3 Attribute Assignment

This section describes the algorithm for assigning priorities and offsets to the periodic tasks and priorities to sporadic tasks in order to meet the constraints specified for a task set. It is assumed that constraints are specified according to the model defined in the previous section. The heart of the attribute assignment is a genetic algorithm that assigns offsets and priorities, evaluates the assignments, and incrementally finds new assignments, thereby gradually achieving the required system behaviour. The general idea of a GA is to let individuals in a population gradually improve by the mechanisms of natural selection. In this case the individuals consists of attribute assignments for a tasks set and the environment to master is the constraints put on that task set. An overview of the structure and operation of the genetic algorithm used is given below.

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

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

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

The analysis performed to calculate the earliest and latest start times and completion times for the instances of the task set can be divided into two cases: 1) The earliest start time and completion time are calculated disregarding the sporadic tasks, using the best-case execution times, and assuming that no tasks are blocked when using shared resources. 2) The latest start time and completion time is calculated considering interference from sporadic tasks, using the worst-case execution times and assuming maximal blocking.

In the objective function given below, task instances are assumed to be enumerated starting with zero for the first instance. The function numInst() returns the number of instances for a task during the *lcm* of the complete task set. The objective function is executed for each of the attribute assignments contained in the GA population. For each assignment an objective value is returned, and that value is then used to rank the different attribute assignments for a given task set. Pessimism in the objective function can be reduced if the mechanism of the used run-time system is considered. For example, in pre-emptive priority based systems, tasks with the same offsets are not influenced by sporadic activities independently of each other. The goal of this work has not been to provide an optimal objective function, the goal has been focused on the overall success of the method, which is indicated by the simulations in section 4.

Objective function

```
    $objective = 0$
    for each task $\tau_i$ “Deadline”
        for each instance $n$ of task $\tau_i$
            if $lct(\tau_i^n) > deadline(\tau_i) + n \cdot periodTime(\tau_i) + offset(\tau_i)$
                $deviation = lct(\tau_i^n) - deadline(\tau_i) - n \cdot periodTime(\tau_i) - offset(\tau_i)$
                $objective = objective + deviation / deadline(\tau_i) /$ numInst$(\tau_i)$
    for each precedence constraint $<\tau_i , \tau_j >$
        for each instance $n$ of task $\tau_i$ and $\tau_j$
            if $lct(\tau_i^n) > est(\tau_j^n)$
                $deviation = 1$
                $objective = objective + deviation /$ numInst$(\tau_i)$
for each separation constraint $<separation, \tau_i , \tau_j >$
        for each instance $n$ of task $\tau_i$ and $\tau_j$
            if $est(\tau_j^n) – lct(\tau_i^n) < separation$
                $deviation = separation – (est(\tau_j^n) – lct(\tau_i^n))$
            objective = objective + deviation / separation / numInst($\tau_i$)
for each start jitter constraint <$S_h$, $S_l$ $\tau_i$>
        for each instance n of task $\tau_i$
            if $lst(\tau_i^{n+1}) - est(\tau_i^n) > S_h$  “$(lcm + lst(\tau_i^0)) - est(\tau_i^n)$ when
                                                            $n = numInst_i - 1$”
                deviation = $(lst(\tau_i^{n+1}) - est(\tau_i^n)) - S_h$
                objective = objective + deviation / $S_h$ / 2 / numInst($\tau_i$)
            if $est(\tau_i^{n+1}) - lst(\tau_i^n) < S_l$  “$(lcm + est(\tau_i^0)) - lst(\tau_i^n)$ when
                                                            $n = numInst_i - 1$”
                deviation = $S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n))$
                objective = objective + deviation / $S_l$ / 2 / numInst($\tau_i$)
Completion Jitter calculated as start jitter above.
for each latency constraint <latency, $\tau_i$ , $\tau_j$ >
        if periodTime($\tau_i$) = periodTime($\tau_j$)
            for each instance n of task $\tau_i$ and $\tau_j$
                if $lct(\tau_i^n) \leq est(\tau_j^n)$
                    If $lct(\tau_j^n) - est(\tau_i^n) > latency$
                        deviation = $(lct(\tau_j^n) - est(\tau_i^n)) - latency$
                        objective = objective + deviation / latency / numInst($\tau_i$)
                else
                    objective = objective + 1 / numInst($\tau_i$)
        if periodTime($\tau_i$) > periodTime($\tau_j$)
            for each instance n of task $\tau_I$
                find the instance $\tau_j^m$ with earliest lct, where $lct(\tau_i^n) \leq est(\tau_j^m)$
                if an instance $\tau_j^m$ is found
                    if $lct(\tau_j^m) - est(\tau_i^n) > latency$
                        deviation = $(lct(\tau_j^m) - est(\tau_i^n)) - latency$
                        objective = objective + deviation / latency / numInst($\tau_i$)
                else
                    objective = objective + 1 / numInst($\tau_i$)
        if periodTime($\tau_i$) < periodTime($\tau_j$)
            Analogous to $periodTime(\tau_i) > periodTime(\tau_j)$
for each correlation constraint $<correlation, \tau_i, \ldots, \tau_m>$
        for each instance $n$ of task $\tau_i$ to $\tau_m$
            find the maximum difference between $lst(\tau_k^n) - est(\tau_l^n)$ for
            any k and l
                    in [i..m] where $k \neq l$.
            if $lst(\tau_k^n) - est(\tau_l^n) > correlation$
                $deviation = lst(\tau_k^n) - est(\tau_l^n) - correlation$
                $objective = objective + deviation / correlation / numInst(\tau_i)$
End Objective function
```

4 Results

A series of simulations have been carried out to evaluate the performance of the proposed method. The first set of simulations shows the success ratio, for the GA, at assigning priorities and offsets to task sets so that the constraints for the task sets are met. For this set of simulations all the constraints presented in this paper are used by the generated task sets. The second set of simulations compares the method in this paper to the algorithm presented in [2] by Bate and Burns. To be able to compare the two approaches the constraints not supported by the algorithm in [2] have been removed.

4.1 Simulation set up

Periodic and sporadic tasks are randomly generated until the specified utilization is reached. The periodic utilization is randomly generated in [0, U], and the sporadic utilization equaling U – periodic utilization, where U is the desired system utilization. The period time of the periodic tasks are randomly selected from a number of predefined period times and the WCET is randomly generated as a percentage of the period time, the percentage is specified as a range, e.g., 2%-4% of the period time. The BCET for a periodic task is defined as a percentage of the WCET of the task. Table 2 displays the numbers used for the periodic tasks in the simulations presented in this paper. The minimum inter-arrival time for sporadic tasks are randomly generated from a predefined set of ranges and the WCET is generated in the same way as for the periodic tasks. The parameters for the sporadic tasks in the simulations are given in table 1.

Min. Inter-arrival time	**Distribution**
[0,1000]	20
[1000, 5000]	70
[5000,20000]	10
WCET in percentage of min. inter-arrival	
[0,1]	30
[1,2]	40
[2,5]	30

Table 1. Data for generating sporadic tasks.

Period time	**Distribution**
10000	20
25000	20
50000	40
100000	20
WCET in percentage of period	
[0,2]	45
[2,4]	50
[4,8]	5
BCET in percentage of WCET	
[0,70]	10
[70,80]	30
[80,90]	30
[90,97]	30

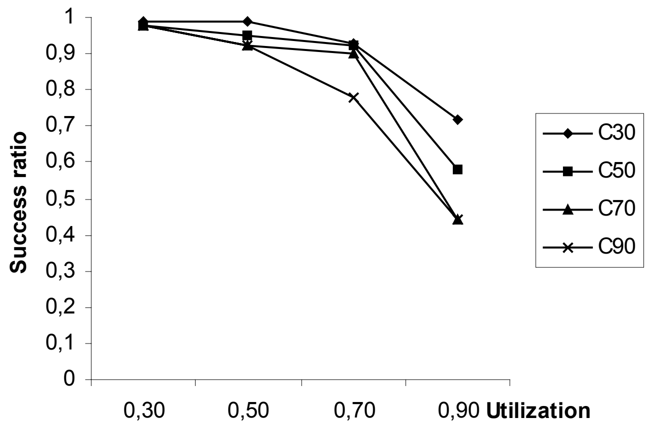
Table 2. Data for generating periodic tasks.

When the tasks with period times, WCETs, and BCETs have been generated, offsets and priorities are randomly generated. The temporal behavior of the task set is then analyzed and the constraints are generated based on the temporal information so that the task set fulfils the constraints. The number of constraints generated are based on a predefined percentage of the number of periodic tasks, e.g., if 70% is specified for the amount of constraints and the number of periodic tasks are 20, then there will be 14 tasks involved in the constraints. Deadlines are not included in this number since a deadline is always randomly generated for each task.

The simulations are performed for four different utilization levels, 30%, 50%, 70%, and 90%. For each utilization level simulations are carried out for different amounts of constraints, 30%, 50%, 70%, and 90%, where the constraints are generated as presented above. There are 100 task sets generated for each utilization and constraints level. The simulations were run on a 550 MHz Pentium III processor with 128 MB RAM. The software is implemented using C/C++. For the basic data structures and operations of the genetic algorithm we used the GAlib genetic algorithm package, written by Matthew Wall at the Massachusetts Institute of Technology. The GA will terminate on success, or when reaching 2000 generations, or when there is no improvement in the objective value for 100 generations.

4.2 Success ratio for the GA

For this simulation all the constraints presented in this paper are used by the generated task sets as well as shared resources between periodic and sporadic tasks (and other combinations). There is an even distribution between the numbers of tasks assigned to the different constraints in order to get a good coverage of all the constraints. Start and completion jitter is not treated as two separate constraints when the amount of constraints is calculated. If a jitter constraint is generated, both start and completion jitter is generated for the task and it is viewed as one task with one constraint. The graph shows the percentage of task sets with attributes assigned that fully meet the constraints. The average number of sporadic and periodic tasks, as well as the total number of tasks is given in Table 3. Finally, Table 4 shows the average computation time for processing the task sets, including computation times for both correct and incorrect assignments.



Graph 1. The success ratio of the GA algorithm for different load and number of constraints.

Constraints / Utilization	30%	50%	70%	90%
30%	11.7/5.8/5.9	19.3/9.3/10.0	26.9/12.6/14.2	34.6/16.3/18.3
50%	11.8/6.1/5.8	19.3/9.4/9.8	27.0/12.7/14.34	34.4/16.5/17.9
70%	11.6/5.5/6.2	19.7/9.8/9.9	26.2/14.1/12.3	34.4/16.8/17.6
90%	11.6/5.6/6.0	19.0/10.2/8.8	27.2/13.2/14.0	33.8/16.3/17.5

Table 3. The average number of tasks for the simulation, displayed as total/periodic/sporadic

Utilization \ Constraints	30%	50%	70%	90%
30%	1	1	1	1
50%	10	12	18	17
70%	98	128	124	132
90%	699	792	882	774

Table 4. The average computation time in seconds.

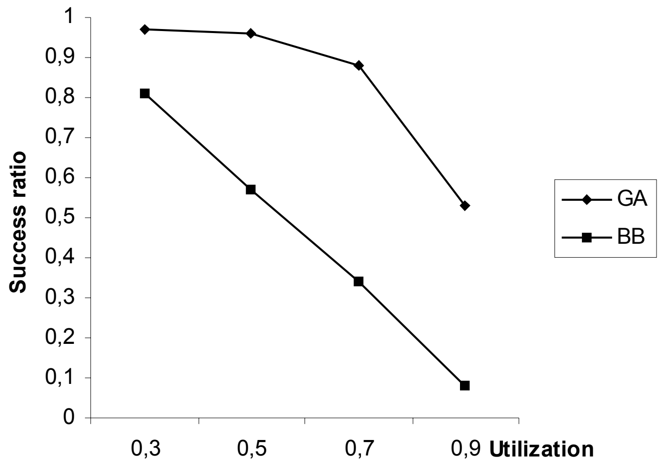
The simulations indicate that the method solves the attribute assignment to a high degree and that the success ratio decreases as the utilization and the number of constraints increases. The number of tasks given by the second graph shows that the results are valid for fairly large embedded systems, while the computation time given in the last graph indicates that for significantly larger task sets, in terms of number of tasks, with high utilization the computation times may be too long to be practical. The computation time of the algorithms is mainly related to the number of tasks and not so strongly to the number of constraints because the analysis is the computationally most demanding part. Although, a large number of constraints may increase the computation time by requiring more generations of the GA, and thereby more analysis, to find a solution. The correlation between the number of constraints and the computation time is also dependent on the termination criteria of the GA. Since the GA terminates on a given number of iterations and when improvement of the algorithm is to slow, large task set with many constraints will to a higher degree be stopped by the termination criteria compared to a large task set with few constraints. Thus, the computation time is kept down at the cost of a lower success ratio. Relaxing the termination criteria would increase computation times but also most likely the success ratio.

4.3 Comparison with Bate and Burns' method

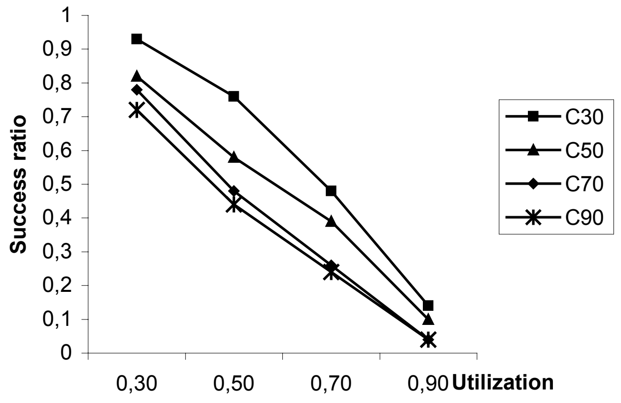
In this section we compare the performance of our method with that of the method presented by Bate and Burns' [2]. Since their method handles less general constraints, we will here use a restricted simulation set-up, in that we have to exclude start jitter, correlation, and shared resources from the task model.

There is an even distribution between the number of tasks assigned to jitter constraints and the number of tasks assigned to separation and latency constraints. The ratio between tasks with latency compared to separation constraints will be 4/1. The ratio is set to reflect an assumed higher number of latency constraints than separation constraints in real systems; this assumption is based on many years industrial experience gained by the authors.

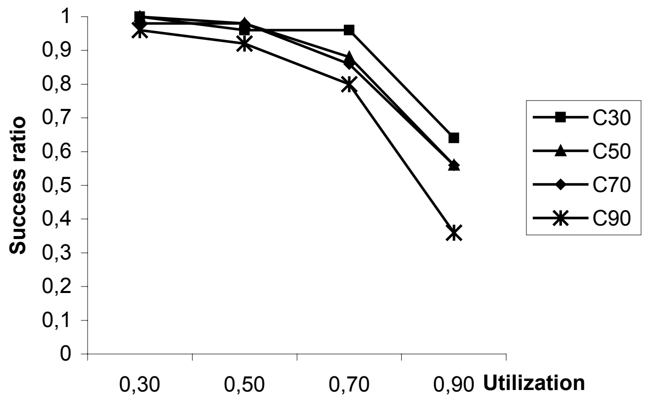
The first graph for this set of simulations (graph 2) shows, for both algorithms, the percentage of task sets with attributes assigned that fully meet the constraints. The success ratio depicted is the average ratio for all constraint levels. The following two graphs (graphs 3 and 4) give the success ratio for each constraint level. In the graphs below, the GA algorithm is denoted GA and the algorithm in [2] is denoted BB.



Graph 2. The average success ratio.



Graph 3. Success ratio for the BB algorithm.



Graph 4. Success ratios for the GA algorithm.

The average numbers of tasks are close to the numbers for the first simulation, given in Table 3. The computation times for the GA algorithm are in the same magnitude as those given in Table 4, while the algorithm in [2] has computation times of at most a few milliseconds.

Graph 2, displaying the success ratio for the two algorithms, shows that the algorithm presented in this paper performs well compared to the algorithm in [2], especially for task sets with high utilization. The difference in success ratio depends on that the GA performs global optimization of the attributes, distributing the tasks execution, using offsets, as needed for meeting the constraints, while the algorithm in [2] does not globally optimize the attribute assignment, but assign attributes based on the properties of each constraint individually. As the utilization increases the effect of using global as opposed to local optimization becomes more apparent.

5 Conclusion

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

In this paper we propose a method for fulfilling complex temporal requirements by assigning priorities and offsets to tasks, running on a standard commercial RTOS. The method uses a genetic algorithm to search for an acceptable solution, i.e., a solution satisfying all constraints. However, even in cases when an acceptable solution cannot be found, the genetic algorithm will provide a near optimal solution indicating which constraints that are difficult (or impossible) to satisfy). This is important from an engineering perspective, since the result can be used as input for remodelling of the application.

Results from simulation shows that the algorithm presented in this paper has a high success ratio in assigning attributes that make schedulable task sets meeting their constraints. Moreover, in comparison to the algorithm in [2] the GA algorithm performs well, with noticeable higher success ratio. The computation times for the GA algorithm is considerable much longer than for the algorithm in [2], which handles a substantially simpler task model. However, the simulation results show that the proposed method efficiently assigns attributes for task sets of a size that covers most embedded control systems, in reasonable time for an off-line tool.

The method proposed in this paper supports specification of jitter constraints for both task start and completion times, making it possible to more precisely control the variations in period time of a task. More over, separation constraints, and latency constraints are supported, as are correlation constraints between tasks executing in parallel. The varying execution times of tasks are supported as well as shared resources between sporadic and period tasks.

Finally, future work includes adding optimisation on other criteria, e.g., minimisation of the number of used offset and priority levels, and general minimisation of e.g., jitter, in order to have as low jitter as possible in the system. Due to the architecture of the GA it is easy to add new optimisations as the ones listed above, and since it only requires additional functionality in the objective function it is cheap in terms of computation time.

6 References

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

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

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

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

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

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

[7] Radu Dobrin, Yusuf Özdemir, Gerhard Fohler. Task Attribute Assignment of Fixed Priority Scheduled Tasks to Reenact Off-Line Schedules. In *Proceedings of RTCSA 2000* Korea , December 2000.

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

[9] F. Coutinho, J.A. Fonseca, J. Barreiros, E. Costa. Jitter Minimisation with Genetic Algorithms, Proceedings 3rd IEEE International Workshop on Factory Communication Systems, Portugal, September 2000.

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

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

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

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

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

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

Appendix A: Example

In this example we assume an application for which constraints needed for implementation have been derived from a control design and expressed according to the specification model described in Section 2. The example system consists of 4 periodic tasks and 1 sporadic task. In Table 5 the periodic tasks are listed.

Task	Wcet	Bcet	Period time
A	2	2	20
B	3	3	20
C	2	2	20
D	3	3	20

Table 5: The example task set.

In addition there is one sporadic task, *SP*, in the system. The sporadic task *SP* has a *Wcet* of 2 and a *minimum inter-arrival time* of 9. Furthermore, task *SP* has a *deadline* constraint of 6. The constraints that apply to the periodic task set are given below.

Start Jitter, <21, 19, A>

Start Jitter, <21, 19, C>

Latency, <9, A, B >

Separation <4, C, D >

Given the specification above, the GA tries to find an attribute assignment in the run-time model such that the execution of the task set fulfils the constraints. The termination criterion of the GA is that the objective function evaluates to zero, i.e., one genome meets all the constraints.

The offset and priority for a periodic task τ_i is represented by the tuple op_i = <offset, priority> and the priority for a sporadic task is represented by p_j = <priority>. The complete representation for the task set of the example is op_A, op_B, op_C, op_D, p_{SP}. A high value represent a high priority.

The size of the population of the genetic algorithm is 3 and the number of offspring generated in each generation is 2. The initial population is given in table 6.

Gen	*op*	op_B	*op*	*op*	p_{SP}
1	<0	<13,1	<0	<1	<3>
2	<2	<6,2>	<9	<1	<5>
3	<1	<9,5>	<0	<1	<3>

Table 6. The initial population.

According to the operation of the GA the next step is to apply the objective function to the genomes of the population. Included in this is to make analysis of each genome.

In Figure 2 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 1.

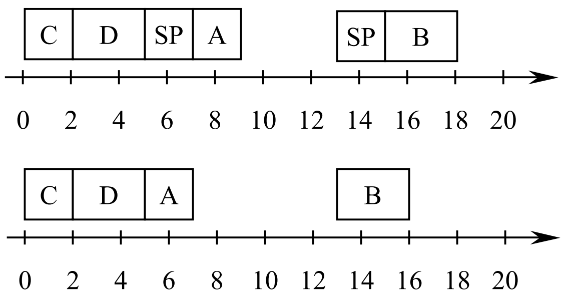


Figure 2. The worst and best case execution scenario.

Table 7 shows the start times and completion times for the tasks.

Tas	*lst*	*lct*	*est*	*ect*
A	7	9	5	7
B	15	18	13	16
C	0	2	0	2
D	2	5	2	5
SP	-	7	-	-

Table 7. Start and completion of the tasks.

The result of the objective function, presented in Section 3, for genome 1 can then be calculated as:

Start Jitter, <21, 19, A>

$deviation = (lst(\tau_i^{n+1}) - est(\tau_i^n)) - S_h = (20+7) - 5 - 21 = 1$

$objective = objective + deviation / S_h / 2 / numInst(\tau_i) =$

$= 0 + 1/21/2/1 = 0{,}024$

$deviation = S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n)) = 19 - (25-7) = 1$

$objective = objective + deviation / S_l / 2 / numInst(\tau_i) =$

$= 0{,}024 + 1/19/2/1 = 0{,}05$

Start Jitter, <21, 19, C>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ ***"Constraint met"***

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ ***"Constraint met"***

Latency, <9, A, B >

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency = 18 - 5 - 9 = 4$

$objective = 0{,}05 + 4 / 9 = 0{,}49$

Separation <4, C, D >

deviation = separation – (est(τ_j^n) – lct(τ_i^n)) = 4 – (2-2) = 4

objective = 0,49 + 4 / 4 = 1,49

Deadline <6, SP>

deviation = lct(τ_i^n) - deadline = 7 – 6 = 1

objective = 1,49 + 1 / 7 = 1,63

Objective: 1,63

In Figure 3 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 2.

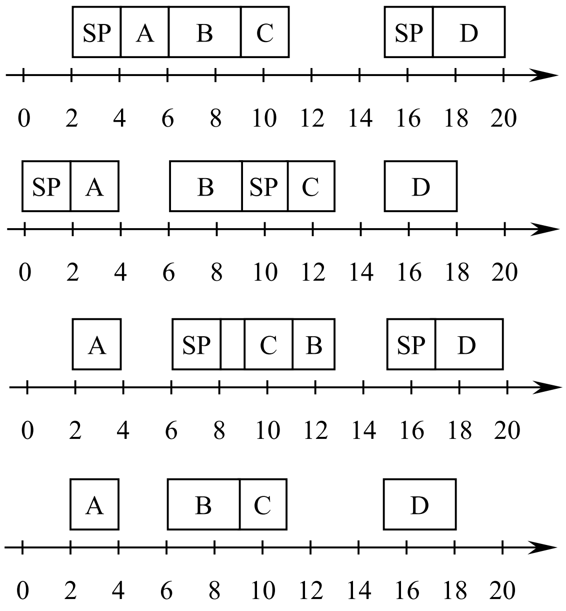


Figure 3. The worst and best case execution scenario.

Table 8 shows the start times and completion times for the tasks.

Tas	*lst*	*lct*	*est*	*ect*
A	4	6	2	4
B	8	13	6	9
C	11	13	9	11
D	17	20	15	18
SP	-	2	-	-

Table 8. Start and completion of the tasks.

The result of the objective function for genome 2 can then be calculated as:

Start Jitter, <21, 19, A>

deviation = (lst(τ_i^{n+1}) – est(τ_i^n)) - S_h = 24 – 2 – 21 = 1

objective = 0 + 1/21/2 = 0,024

deviation = S_l – (est(τ_i^{n+1}) – lst(τ_i^n)) = 19 – (22-4) = 1

objective = 0,024 + 1/19/2 = 0,05

Start Jitter, <21, 19, C>

deviation = $(lst(\tau_i^{n+1}) - est(\tau_i^n))$ - S_h = 31 – 9 – 21 = 1

objective = 0,05 + 1/21/2 = 0,074

deviation = $S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n))$ = 19 – (22-4) = 1

objective = 0,074 + 1/19/2 = 0,1

Latency, <9, A, B >

deviation = $(lct(\tau_j^n)$ - $est(\tau_i^n))$ – *latency* = 13 – 2 – 9 = 2

objective = 0,1 + 2 / 9 = 0,32

Separation <4, C, D >

deviation = *separation* – $(est(\tau_j^n) - lct(\tau_i^n))$ = 4 – (15-11) = 0

objective = 0,32 + 0 / 4 = 0,32

Deadline <6, SP>

$lct(\tau_i^n)$ <= *deadline* "***Constraint met***"

Objective: 0,32

In Figure 4 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 3.

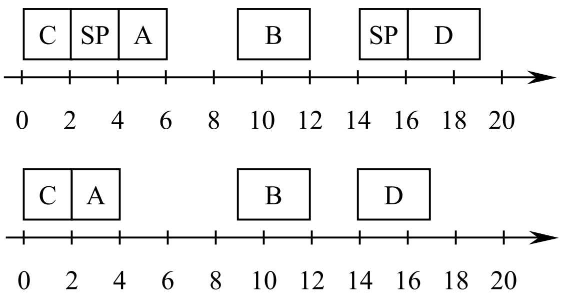


Figure 4. The worst and best case execution scenario.

Table 9 shows the start times and completion times for the tasks.

Tas	*lst*	*lct*	*est*	*ect*
A	4	6	2	4
B	9	12	9	12
C	0	2	0	2
D	16	19	14	17
SP	-	5	-	-

Table 9. Start and completion of the tasks.

The result of the objective function for genome 3 can then be calculated as:

Start Jitter, <21, 19, A>

deviation = $(lst(\tau_i^{n+1}) - est(\tau_i^n))$ - S_h = 24 – 2 – 21 = 1

objective = 0 + 1/21/2 = 0,024

$deviation = S_l - (est(\tau_i^{n+1}) - lst(\tau_i^n)) = 19 - (22\text{-}4) = 1$

$objective = 0{,}024 + 1/19/2 = 0{,}05$

Start Jitter, <21, 19, C>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ ***"Constraint met"***

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ ***"Constraint met"***

Latency, <9, A, B >

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency = 12 - 2 - 9 = 1$

$objective = 0{,}05 + 1 / 9 = 0{,}16$

Separation <4, C, D >

$separation < (est(\tau_j^n) - lct(\tau_i^n))$ ***"Constraint met"***

Deadline <6, SP>

$lct(\tau_i^n) < deadline$ ***"Constraint met"***

objective: 0,16

Since no genome meets the termination criteria the GA proceeds with the next step, i.e., to generate offspring from the population. In this example we use one point crossover. In the crossover operation a new genome is formed by selecting a position in the genome, taking the information to the left of that position from one genome, and combine it with the information to the right of that position from another genome. In the example the crossover results in two new genomes. By combining op_A and op_B from genome 2 with op_C, op_D, and p_{SP} from genome 1 the first genome is produced. The second genome is produced by combining op_A and op_B from genome 2 with op_C, op_D, and p_{SP} from genome 3. Table 10 display the new genomes.

Genome	op_A	op_B	op_C	op_D	p_{SP}
4	<2,4>	<6,2>	<0,5>	<1,4>	<3>
5	<2,4>	<6,2>	<0,4>	<14,1>	<3>

Table 10. The new genomes.

The next step is to apply a random mutation to the new genomes with some probability. In the example op_A of genome 4 is changed from <2,4> to <4,4> by mutation.

In Figure 5 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 4.

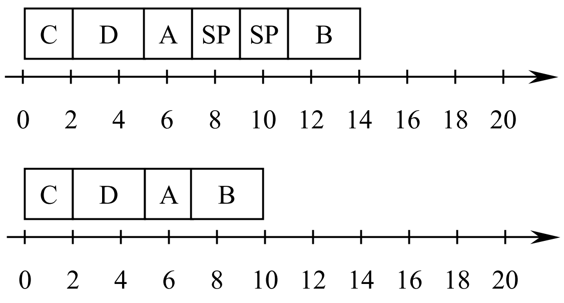


Figure 5. The worst and best case execution scenario.

Table 11 shows the start times and completion times for the tasks.

Tas	*lst*	*lct*	*est*	*ect*
A	5	7	5	7
B	11	14	7	10
C	0	2	0	2
D	2	5	2	5
SP	-	9	-	-

Table 11. Start and completion of the tasks.

The result of the objective function for genome 4 can then be calculated as:

Start Jitter, <21, 19, A>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ ***"Constraint met"***

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ ***"Constraint met"***

Start Jitter, <21, 19, C>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ ***"Constraint met"***

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ ***"Constraint met"***

Latency, <9, A, B >

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency = 14 - 5 - 9 = 0$

$objective = 0 + 0 / 9 = 0$

Separation <4, C, D >

$deviation = separation - (est(\tau_j^n) - lct(\tau_i^n)) = 4 - (2\text{-}2) = 4$

$objective = 0 + 4 / 4 = 1$

Deadline <6, SP>

$deviation = lct(\tau_i^n) - deadline = 9 - 6 = 3$

$objective = 1 + 3 / 6 = 1{,}5$

objective: 1,5

In Figure 6 the worst and best case scenarios' for the task set are displayed, assuming offset and priorities according to genome 5.

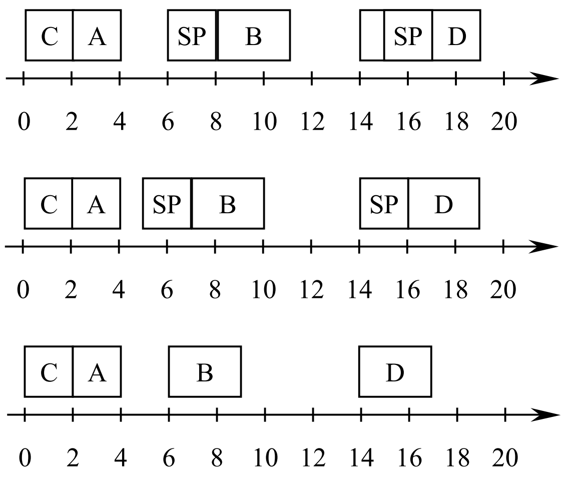


Figure 6. The worst and best case execution scenario.

Table 12 shows the start times and completion times for the tasks.

Tas	*lst*	*lct*	*est*	*ect*
A	2	4	2	4
B	8	11	6	9
C	0	2	0	2
D	16	19	14	17
SP	-	6	-	-

Table 12. Start and completion of the tasks.

The result of the objective function for genome 5 can then be calculated as:

Start Jitter, <21, 19, A>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ ***"Constraint met"***

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ ***"Constraint met"***

Start Jitter, <21, 19, C>

$(lst(\tau_i^{n+1}) - est(\tau_i^n)) < S_h$ ***"Constraint met"***

$S_l < (est(\tau_i^{n+1}) - lst(\tau_i^n))$ ***"Constraint met"***

Latency, <9, A, B >

$deviation = (lct(\tau_j^n) - est(\tau_i^n)) - latency = 11 - 2 - 9 = 0$

$objective = 0 + 0 / 9 = 0$

Separation <4, C, D >

$separation < (est(\tau_j^n) - lct(\tau_i^n))$ ***"Constraint met"***

Deadline <6, SP>

$lct(\tau_i^n) <= deadline$ ***"Constraint met"***

objective: 0

The best genomes are selected and the resulting population with their respective objective value is given in Table 13.

Genome	op_A	op_B	op_C	op_D	p_{SP}	*value*
5	<0,2>	<13,1>	<0,5>	<1,4>	<3>	0
2	<2,4>	<6,2>	<9,3>	<15,1>	<5>	0,16
3	<1,4>	<9,5>	<0,4>	<14,1>	<3>	0,49

Table 13. The resulting population.

As the best genome meets the termination criterion the GA is terminated. We have found an offset and priority assignment that fulfil all the temporal constraints.

Frame Packing in Real-Time Communication

by

Kristian Sandström, Christer Norström, Magnus Ahlmark

Mälardalen Real-Time Research Centre, Department of Computer Engineering, Mälardalen University, Västerås, Sweden

In proceedings of RTCSA 2000, Korea, December 2000.
IEEE Computer Society

Frame Packing in Real-Time Communication

Kristian Sandström, Christer Norström, Magnus Ahlmark

Mälardalen Real-Time Research Centre, Department of Computer Engineering, Mälardalen University, Västerås, Sweden

Abstract

A common computational model in distributed embedded systems is that the nodes exchange signals via a network. Most often a signal represents the state of some physical device and has a signal size ranging from a single bit up to a few bytes. Furthermore, each signal typically has a deadline requirement. The communication networks used are often based on a broadcast bus where fixed or variable sized frames are transmitted. The amount of data that can be transmitted in each frame is almost always bigger than the size of a signal. Thus, from a resource perspective it would be desirable if each frame could transport several signals.

In this paper we investigate how to assign signals to periodic frames with the objective function to minimise the network bandwidth requirement while not violating specified deadlines. This problem is NP-hard, but can for most typical applications be solved efficiently by using simple heuristics. The effectiveness of our algorithm is demonstrated by applying it to signal sets derived from automotive applications for a CAN based system and for the newly developed, low cost and low speed, Local Interconnect Network (LIN). The results can be of great use in cost sensitive embedded systems such as car control systems, where the used hardware, communication networks and nodes (typically micro-controllers), have to be highly utilised to keep the production cost at a minimum level.

1 Introduction

Today most modern cars are computer controlled in order to decrease the production cost (especially to reduce the amount of installed cables) and to facilitate the implementation of new functionality such as anti skid, which is very hard, or even impossible, to implement in purely mechanical systems [4].

When replacing classical solutions, such as connecting a switch directly to a device, e.g., a motor or a lamp, with a computer network based solution; the status of the sensor has to be sampled, transmitted over the network, received by the consuming node, and finally actuated to the device within an appropriate time interval. Each sensor entity sent over the network is called a signal. A common computational model in distributed embedded systems is that the nodes exchange signals via a network [5].

The timing requirements for each signal sent over the network have to be derived from the controlled process. Thus, each signal has a size and a timing requirement specification. The timing terminology used in this paper: The End To End Deadline (ETED) is the maximum delay from a stimuli until a response is given to the environment for a specific function. An ETED timing requirement for a function has to be broken down to individual timing requirements for the components that constitute the function. This is the application engineer’s task. We will use *deadline* in this context to denote the timing requirement for a signal sent on the network. More specifically the deadline specifies the maximum delay between when a signal is available at the sending node's communication subsystem until it is available for the application(s) on the receiving nodes.

Since each node most likely will send several signals, the signals should be packed in frames so that the communication bandwidth usage is minimised. Consider the case when only one signal is included in each frame (the size of a signal is considered to be less than the size of a frame) and the frame is transmitted periodically with a minimum frequency that fulfils the deadline. Then the communication cost would be high because each signal would get the burden of all overheads in a frame, such as control information and checksum. Consider the opposite situation, the signals are packed in as few frames as possible and there exists two different sizes of frames (i.e. frames that carry different amounts of data and have different transmission times). Furthermore, assume that the signals fit into one large frame. If the signals have about the same deadline, it would be beneficial to send them in one large frame. On the other hand if we have for example two groups of signals that have quite a large difference in deadlines then it would be beneficial to divide the data into two frames. Because the smallest deadline in each frame determines the period time of the frame, and thus the bandwidth utilisation would become less than packing all signals into a large frame.

To assign signals to frames is difficult since (1) the signals are asynchronous (i.e., the different signals are available for the communication subsystem at non synchronised times) and (2) many protocols for embedded systems allow different frame sizes with different transmission times, e.g., in a CAN-based system, the data is transmitted in frames containing between 0 and 8 bytes of data.

Thus, the problem investigated in this paper is: Given a finite set of signals for each node, where each signal is characterised by a deadline and a size. Further a finite number of different sized types of frames with different transmission times are given. Find a mapping of signals to periodic frames, which will minimise the bandwidth utilisation of the communication network such that all of the signals are uniquely assigned to frames and that the frames are globally schedulable.

When comparing different packing alternatives we have chosen to define an utilisation measure for a frame as the transmission time divided by the deadline of the frame, and consequently the utilisation measure for the network as the sum of utilisation measures for all frames. Note that we use deadline instead of period time in the definition of the utilisation measure, because we want to separate the frame packing from scheduling. In the scheduling phase, periods are determined based on deadlines, frame transmission time, and the scheduling method used. A straightforward solution is to transmit each frame with a period time that is equal to half of the deadline then the deadline requirement for each frame will be fulfilled, but possibilities exist [13].

Thus we have a set of frames where each frame has a period time and a transmission time, which we have to perform schedulability analysis on. Several mature techniques for schedulability analysis of periodic frames for different protocols exist, including the technique developed for the CAN-bus by Tindell et al. [1,2,3] and techniques for off-line generation of timetables 9.

The problem we address is similar to the task allocation and scheduling problem that has been studied by many researchers, e.g. [9,11]. The main difference is that most often in task allocation, a system with a finite set of nodes is given while in our case we have non-finite set of frames. Furthermore, the task allocation and scheduling problem is harder since tasks often have relations between each other, including mutual exclusion and precedence. Our work also relates to the work done in multimedia applications where multiple streams are to be guaranteed as in [12], where they model the problem as a multidimensional bin-packing problem. Their problem is slightly more complex since they handle different kinds of resources like, disk, CPU and network resources. However, we have not found any work that has attacked the frame-packing problem.

The contributions of this paper are that we:

- Formulate the packing problem.
- Show that the packing problem is NP-hard
- Present a simple heuristic for frame packing that we show is very effective.
- Demonstrate the effectiveness of the algorithm on realistic sized problems derived from the automotive industry.

The rest of this paper is organised as follows. Section 2 presents our system model and the formalisation of the problem. Section 3 presents the proposed algorithms, whereas in Section 4 the corresponding analysis results are presented. Finally in Section 5 we will draw some conclusion.

2 Problem statement

2.1 System model

We assume a distributed system consisting of a set of nodes interconnected via a communication network. The communication protocol is assumed to be a packet transmission protocol with a limited set of frame sizes. A frame contains one or more signals and the size of a signal is assumed to be less or equal to the size of the largest frame. Each node transmits and receives signals, where a signal has one producer and one or more consumers. Each signal has a specified size and deadline. We assume that the period time of generation of new signal values is greater than the deadline of the signal. The nodes may or may not have a global synchronised time base.

2.2 Problem formulation

For each node the following problem has to be solved. Given a finite set $S = \{s_1, s_2, ..., s_n\}$ of signals with size $sz(s_i) \in N^+$ and a deadline $d(s_i) \in N^+$. We define a frame *f* as a collection of signals from *S*. Each frame has an associated transmission time $c(f_j) \in N^+$ and a size $sz(f_j) \in N^+$, defined by the used communication protocol.

The problem is now to find a mapping of *S* into a set of frames $F = \{f_1, f_2, ..., f_l\}$, such that each $s_i \in S$ is included in a unique f_j, with $\sum_{\forall s_i \in f_j} sz(s_i) \le sz(f_j)$, and which minimises the bandwidth utilisation measure $U = \sum_{\forall f_k \in F} \frac{c(f_k)}{\min_{\forall s_i \in f_k}(d(s_i))}$.

Each frame has to be transmitted with a rate that fulfils the deadline requirement on the signal with the shortest deadline in the frame f_i. The objective is to map the signals into frames such that the bandwidth requirement *U* is minimised, while making sure that frames are schedulable.

This problem is *NP*-hard in the strong sense since it easily can be shown that it is a special case of the well known "bin packing" problem, which is a *NP*-hard combinatorial optimisation problem [7]. The "bin packing" problem is obtained when all signals have the same deadline and when there is only one size of frames. Then our optimisation problem becomes to pack the signals in as few frames as possible, which is exactly the "bin packing" problem. So if our problem is proven to belong to class *P* then should also the "bin packing" problem belongs to that class, which is a contradiction, unless $P = NP$.

3 An engineering approach: mapping signals to frames

The frame-packing problem is a NP-hard problem and hence we need to solve the problem by using heuristic techniques. To get a measure of the effectiveness of our algorithms, a theoretical lower bound for the utilisation is derived for the signals. This theoretical lower bound is never higher than the real lower bound.

The lower bound is calculated by assuming that each signal is transmitted in a frame with the lowest cost per bit and the deadline of the frame is the same as the deadline of the signal. A frame has a transmission time and a data size.

We define the lowest theoretical overhead per bit by $MINOH = \min_{\forall f}(c(f)/sz(f))$. The minimal theoretical signal utilisation, *SU*, for a signal *s* is calculated by $SU(s) = \frac{sz(s)}{d(s)} \times MINOH$. The theoretical lower bound of the utilisation for all signals is calculated by $LB(S) = \sum_{\forall s \in S} SU(s)$. Intuitively, this corresponds to packing each signal in a minimum overhead frame, together with other frames with the same deadline that completely fills up the frame.

Our heuristic approach is to first sort the signals in increasing deadline order and then pack the signal into frames by a heuristic algorithm. We will consider two type cases of packing, the first packing algorithm (fixed frame size) considers only one size of the frames and exploits the first fit algorithm and the second algorithm (linear frame selection) uses heuristics for deciding which frame size to be used. A more detailed description of the algorithms can be found in [6].

3.1 Fixed frame size

The algorithm for fixed size frames assigns signals to a frame until a signal does not fit into the frame, then a new frame is created and the signal is assigned to that frame.

3.2 Linear frame selection

The algorithm starts off with a frame of the smallest frame size and assigns signals to that frame. When a signal *s* does not fit into the frame a selection is made; the cost (in bandwidth usage) for using a larger frame that fits all signals including *s* is compared with the cost of keeping the original frame and assigning *s* to a new frame with the smallest possible size. The alternative with the lowest cost is preferred. Moreover, when several frames have been created the algorithm first traverses the frames in order, trying to fit the signal into some unused space. If that is not successful the procedure described earlier is started.

A nice property of both algorithms presented is that they are polynomial time algorithms. Which in practice mean that they are very fast to run even for large signal sets.

4 Simulation

To evaluate the quality of our algorithms we will perform analysis for type-cases of signal sizes and deadline distributions, both for a Controller

Area Network (CAN) [10] based system and the slow and low cost Local Interconnection Network (LIN) [8]. CAN is a broadcast bus designed to operate at speeds up to 1 Mbps. Data is transmitted in frames containing between 0 and 8 bytes of data. A LIN installation usually runs at the speed of 5-20 Kbps/s and is intended to be used for control of internal lights, window drivers, selection switches, etc. in automotive systems. Data is transmitted in frames containing 2, 4 or 8 bytes of data.

We have chosen to study these two buses because they operate on different speeds and have different sets up of possible frame sizes. Further, a CAN based system is more likely to be used for sending larger signals in terms of number of bits since it is mostly used for sending control data, while the LIN based system is mostly used for replacing simple on/off logic. The sizes and deadline distributions for each bus have been derived from discussions with our industrial partners [14].

To generate signal sets we have developed a test case generator that takes the following as input:

- The theoretical lower bound bandwidth, which is used for regulating the amount of signals to be generated.
- The distribution of signal sizes (e.g., 70% 1 bit signals, 20% 2 bit signals and 10% 4 bit signals)
- The distribution of deadlines (e.g., 20% of the signals has a deadline of 10, 25% of the signals have a deadline of 25 etc.)

4.1 CAN simulation

Signals were created with a distribution of the signal size according to Table 1 and each signal was given one out of nine different deadlines. Table 1 gives also the probability for assigning a specific deadline to a signal.

Size distribution		**Deadline distribution**	
Size	**Probability**	**Deadline**	**Probabi**
1	0.20	20	0.07
2	0.20	40	0.20
3	0.10	50	0.25
5	0.10	75	0.05
8	0.20	100	0.10
1	0.05	150	0.10
1	0.15	200	0.10
		250	0.10
		400	0.03

Table 1. Distribution of signal sizes (a) and deadlines (b) for the CAN simulation.

The graph presented in Figure 1 shows the bandwidth utilisation of the frames as a function of generated signal sets with different loads. The graph was obtained by running 10000 generated signal sets for each load level. The graphs include the result from the lfs algorithm and the fixed frame size algorithm. The fixed frame size algorithm was executed for 8 different frame sizes, however smaller CAN frames have been omitted as they result in much higher bandwidth utilisation. The network was assumed to operate at 500 Kbps.

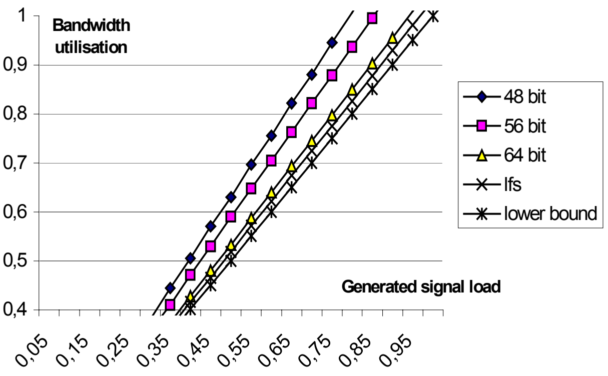


Figure 1. The performance at different load levels.

As can be seen from the graph we are close to optimal in fact we are just some percents above the optimal and thus it seems that we have a rather good heuristic.

4.2 LIN-simulation

Signals were created with a distribution of the signal size according to Table 2 and each signal was given one out of seven different deadlines. Table 2 gives also the probability for assigning a specific deadline to a signal. The cost for the three different frame sizes was assumed to be 15, 20 and 25 respectively.

Size distribution		**Deadline distribution**	
Size	**Probabilit**	**Deadlin**	**Probabilit**
1	0.50	50	0.05
2	0.20	75	0.10
3	0.20	100	0.20
10	0.05	150	0.20
16	0.05	200	0.20
		400	0.20
		1000	0.05

Table 2. Distribution of signal sizes (a) and deadlines (b) for the LIN simulation.

The graphs presented in Figure 2 shows the bandwidth utilisation of the frames as a function of generated signal sets with different loads. The graphs were obtained by running 10000 generated signal sets for each load level.

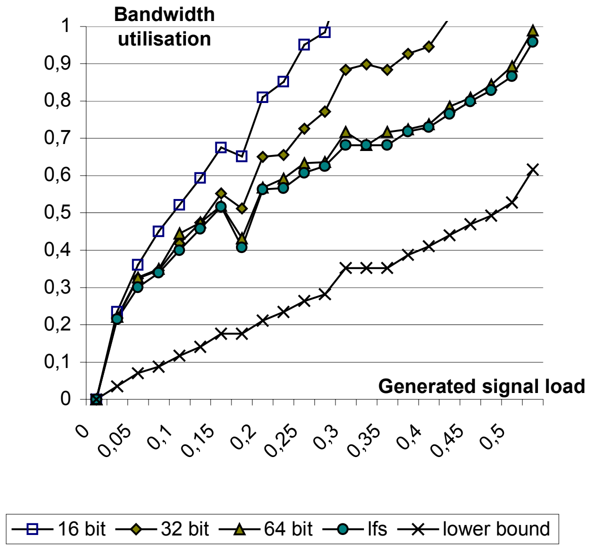


Fig. 2. The performance of the algorithms at different load levels generated

For small signal sets the 16 and 32 bit frames in the LIN simulation gives better performance than the 64 bit frame because the effect of not filling up the "last" frame is less significant. Further, compared to the CAN simulation, the LIN simulation also has a larger gap between the lower bound and the lfs algorithm since the price of not filling up the last frame is much higher (because the CAN-bus runs on a much higher speed), the CAN simulation includes significantly more signals and frames, and that only 3 frame sizes can be used in LIN.

4.3 Discussion

The CAN simulation includes many more signals, and hence more frames, than the LIN tests and thus the price of not filling up the "last" frame is less significant.

For small signal sets the 16 and 32 bit frames in the LIN simulation gives better performance than the 64 bit frame, because the effect of un-used space in the last frame is in average much higher.

It is quite easy to construct "pathological" cases where for example the 64 bit fixed frame behave much worse than the lfs algorithm.

Since all algorithms are so cheap to run one can always select the best result provided by any of the algorithms.

5 Conclusion

In this paper we have presented the frame-packing problem, made a formalisation of the problem, showed that the problem is NP-hard, presented a heuristic solution, and demonstrated the heuristics effectiveness on signal sets that have been derived from real automotive applications.

The results from this paper can be used for many different communication networks where several small signals have to share the space available in one frame.

Further research includes looking into the issue of adjusting the period times of the frames in an efficient way.

An interesting theoretical problem is to find out if it is possible to find an approximation algorithm, which can give a worst-case upper bound on the waste of bandwidth for the algorithms presented in this paper.

Acknowledgements: We would like to thank Hans Hansson, Sasikumar Punnekat, Jukka Mäki-Turja, Ralf Elvsén, and Henrik Thane for valuable discussions and for reviewing earlier versions of this paper.

Mälardalen Real-Time research Centre (MRTC; www.mrtc.mdh.se) is a research centre in Västerås, Sweden, supported by Swedish industry, the Swedish Foundation for Knowledge and Competence Development (KK-stiftelsen) and Mälardalen University.

6 References

[1] Tindell K., J. Clark, Holistic Schedulability Analysis for Distributed Hard Real-time Systems. Technical Report YCS197, Real-Time Systems Research Group, Univ. of York, 1993.

[2] K. W. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control networks. Technical Report YCS229, Dept. of Computer Science, University ofYork, June 1994.

[3] K. W. Tindell, A. Burns and A. J. Wellings. *Calculating Controller Area Network (CAN) message response times*, Control Engineering Practice 3(8):1163-1169, 1995.

[4] K. Melin. Volvo S80: Electrical system of the future Volvo Technology Report. 98-12-11.

[5] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg Volcano a revolution in on-board communications. Volvo Technology Report. 98-12-10.

[6] C. Norström, K. Sandström, Magnus Ahlmark. Frame Packing in Real-Time Communication, MRTC Technical report, July 2000, www.mrtc.mdh.se.

[7] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman San Francisco, CA, ISBN 0-7167-1045-5, 1979.

[8] LIN Protocol Specification, http://www.lin-subbus.org/

[9] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In 10th Int. Conf. on Distributed Computing Systems, pages 108-115, 1990.

[10] Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication, ISO DIS 11898, February 1992.

[11] J Jonsson and J Vasell. Evaluation and comparison of task allocation and scheduling methods for distributed real-time systems. In proceeding of Second IEEE International Conference on Engineering of Complex Computer Systems, 21-25 Oct. 1996.

[12] H. Jiandong and D. Ding-Zhu. Resource management for continuous multimedia database applications. In proceedings of RTSS'94. 1994.

[13] A. K. Mok. FUNDAMENTAL DESIGN PROBLEMS OF DISTRIBUTED SYSTEMS FOR THE HARD–REAL-TIME ENVIRONMENT. Ph.D. thesis MIT 1983.

[14] C. Norström, K. Sandström, M. Gustafsson, J. Mäki-Turja, and N.-E. Bånkestad. Findings from introducing state-of-the-art real-time techniques in vehicle industry. In industrial session of the 12th Euromicro Conference on Real-Time Systems, Stockholm, Sweden, 2000.