Interference Control for Integration of Vehicular Software Components

Mikael Åkerholm, Kristian Sandström, Johan Fredriksson
Mälardalen Real-Time Research Centre, http://www.mrtc.mdh.se
Mälardalen University, Västerås, Sweden
E-mail:{mikael.akerholm,kristian.sandstrom,johan.fredriksson}@mdh.se

Abstract

Vehicular manufacturers want to reduce the number of electronic components in the vehicles foremost to reduce cost and complexity, but not to the price of decreased functionality or quality. In this work we outline a method to facilitate a reduction of hardware components in vehicles, through integration of large real-time software components to the same hardware platform. The focus is on controlling the interference caused by the integration, but practical issues as system architecture and hardware independency both in implementation and specification of components are also considered.

1. Introduction

Component Based Development (CBD) is believed to facilitate scalability and flexibility to integrate and transfer functions; enhance maintainability, and simplify reuse. In the vehicle industry CBD is practiced with hardware components called Electronic Control Units (ECUs). Typically every major vehicular function is implemented within an own ECU, and vehicle manufactures tend to be system integrators of ECUs developed by third parties. This implies that the electronic contents in the vehicles increase with the number of functions in the vehicles. Several reasons to stop this trend exist. There is not much physical space for more ECUs in the vehicles. The system is integrated around a shared communication bus, which is a potential problem since it gather all complexity and creates dependencies around the bus. As always, the most important argument for a decreased number of ECUs is that there is money to save in decreasing the number of hardware components. Firstly there is money to save on a reduction of all electrical equipment that an ECU requires, e.g., network interface card, cabling, motherboard. Secondly, the price relative to performance characteristics of micro controllers themselves yields more performance for less money, i.e., it is cheaper to purchase one powerful micro controller than achieving the same performance with two or more weaker micro controllers.

The long term objective with the work is to facilitate a reduced number of hardware components in the vehicles, through integration of vehicle functions to the same ECUs. This will decrease production cost, and overall complexity. The production cost is deceased through lower hardware cost, the gain increase with the production volume. The complexity around the bus can be decreased through the effects of a lower number of connected ECUs. However, only software in ECUs that is physically close enough is cost effective to integrate, otherwise cost of cablings to sensors and actuators can consume the gain, due to required cable length.

The contribution with this paper, is the introduction of large software components denoted SoftECUs, and interference control when several SoftECUs share a hardware platform. A SoftECU encapsulate a major vehicle function, it can contain as much software as all the software shipped within a traditional ECU. Given that several SoftECUs is integrated on the same physical ECU, the focus of the work is to control the interference that come from the integration, i.e., the interference the SoftECUs can cause to each others. The interference can take two forms, spatial, or temporal. Furthermore, the method scales to SoftECUs that are processor independent. The main reason for introducing processor independency is to simplify system evolution, e.g., migration to another processor when the current is too slow or no longer produced.

Vehicle systems can be classified as safety-critical, embedded real-time systems. Recent research and development efforts taking place in academia and industry have resulted in component technologies for such systems. Here a sample of such technologies is mentioned. The Rubus Component Model [5], from Arcticus Systems is used in the vehicle industry. It is tailored for resource constrained systems with real-time requirements. PErvasive COmponent Systems (PECOS) [9] developed by ABB and academic partners, is aimed for small embedded systems (field devices). It supports prediction of run-time properties as

memory consumption and timeliness. Prediction-enabled Component Technology (PECT) [23] is ongoing research from Carnegie Mellon University; it focuses on prediction of run-time attributes on system level from components. AutoComp [15] is ongoing research at Mälardalen Real-Time Research Centre, the focus is to provide a full component model at design time, and during compile time transform it to a resource effective mature real-time operating system. These technologies could all be suitable for vehicular systems, but they use small components for developing functions. While the approach in this work is to define large components (SoftECUs), which can be used for integration of several functions to the same hardware platform, the components under integration could be built using one of the mentioned technologies.

Outline: Section 2, gives a general overview of integration of SoftECUs, section 3 the overall architecture of a component technology supporting SoftECUs. Section 4 defines the SoftECUs, while section 5 deal with the specification of temporal attributes. Section 6, address the integration. While section 7 derives a model of the run-time behavior, eventually sections 8 contain discussions, conclusions, open issues and future work.

2. General Overview

A general overview of the intended usage of SoftECUs is given in figure 1. Each SoftECU is developed, verified and merchandised independently, by subsystem suppliers. Suppliers use their own tools and create a suitable internal architecture during development of a SoftECU. The main difference for sub system suppliers compared to the situation in the business segment today, is that components are delivered as software only (SoftECUs), instead of both software and hardware (ECUs).

However, in order to secure that the SoftECUs will be assigned a share of the processor that are sufficient, when sharing the processor with other SoftECUs, we propose an additional specification of the SoftECUs processor share requirements. The specification is denoted the *temporal reservation* of the SoftECU. The temporal reservation is a cornerstone in this work, and its origin is from one of the three possibilities listed below:

- the temporal reservation can be provided by the developer;
- the temporal reservation can be in the form of requirements expressed by the integrator;
- it can be negotiated between involved parties;

The task for the system integrator is besides building a system with correct functionality, to verify that all inference caused by the integration is controlled, and it is here the technical contribution of the method is applicable. The interference caused by the integration is either spatial or temporal. For spatial interference each SoftECU is executing within a memory protected process, with possibility for multi–threading or -tasking within the process. The temporal interference on the other hand is eliminated by a time sharing algorithm residing in the RTE that shall be used in cooperation with a couple of specified integration steps.

Furthermore, the method optionally let the temporal specification be separated from the hardware platform, given that the speedup ratio between the development hardware and known reference hardware platform can be determined. Hardware independency is proven through practical use for software without real-time requirements; but becomes an issue when dealing with real-time applications. The problem comes from execution time variations on different hardware, and temporal analysis rely on execution time specifications.

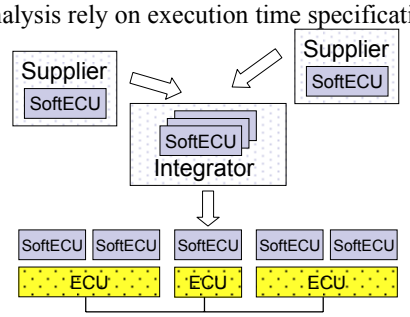

Fig. 1, usage of softECUs

3. Component Technology Architecture

This section gives a rough description of the architecture of a component technology, suitable for the vehicular domain and SoftECUs. Basic terms used throughout the paper are also introduced and explained. The terminology and basic system architecture is with the purpose to be appropriate for the domain influenced from the AUTOSAR[1] standardization project, which is a standardization effort taking place in the vehicle domain by some major actors.

A schematic overview showing relationships between different concepts is shown in figure 2. On the top level in the figure, there are two interconnected ECUs. The interconnection is typically a CAN[2] or

[1] AUTOSAR Homepage: http://www.autosar.org
[2] CAN Homepage: http://www.can-cia.de/can/

LIN[3] bus, and there are also a number of upcoming standards as TT-CAN[4], and Flexray[5]. Each ECU has as a layered architecture; the contents in each of the layers are briefly described below:

- in the hardware layer the microcontroller, hardware parts for communication, I/O units, and different types of memory, are the main blocks;
- the hardware abstraction layer contain hardware dependent code, and provides a hardware independent interface for the above layers;
- basic software typically include, device drivers, transfer layers for communication technologies, and diagnostics software;
- Eventually, the RTE consists of interface for the SoftECUs, provides communication channels, and implements a processor sharing algorithm.

The focus in this paper is on controlling the interference the different SoftECUs will cause each others when they share the same physical ECU. Looking at the layered architecture in figure 2 again, the interface that is addressed by this research is between the SoftECUs, and the RTE. However, the interface between the RTE and the SoftECUs contain more than interference control. In brief the RTE must contain a full flavored programming interface for the SoftECUs, it contains interface to communication mechanisms, and I/O units and it maintain a consistent view of the system time.

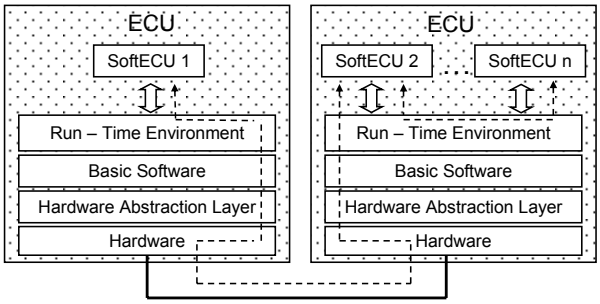

Fig. 2, a schematic picture showing the relationship between different concepts

4. The SoftECU

A SoftECU is described by definition 1, in the remaining part of this section the definition is motivated, explained and enlarged.

Definition 1 *A SoftECU is a software unit that contains a major part of a vehicular function. It can be in pre-compiled intermediate format (black-box), or in source code format (white-box). It comply with the rules of the underlying run-time environment, and can execute on a node alone, or sharing the node with other SoftECUs.*

A SoftECU encapsulates a major part of a vehicular function; it is a unit of exchange between suppliers of vehicular functions and vehicular manufacturers. When building distributed vehicular functions that is physically distributed over the vehicle, it is necessary to deliver the function as several SoftECUs.

The SoftECUs can be black-box, meaning that the source code of the SoftECUs do not have to be directly visible for a system integrator. The integrators knowledge of a SoftECU can be limited to the associated specification of the SoftECU. However the components can also be delivered as source code, in some cases vehicle manufacturers need full access to the source code, e.g., for verification of safety critical functions.

The SoftECUs are not allowed to have hidden dependencies; the only dependencies that are allowed are exchanging data on the shared bus (virtual bus within node boundaries) utilizing interfaces provided by the RTE. It is compatible with the form of interaction that is used in the business segment today.

The SoftECUs must comply with the rules defined by the RTE, which can be compared to a component framework, middleware, or operating system. However, the descriptions of the RTE in this work only address problems that come from the integration itself, i.e., processor sharing problems.

A SoftECU can execute on a processor alone or sharing it with other SoftECUs, under controlled interference. The interference SoftECUs can cause each others is either temporal or spatial. Controlling temporal interference is a matter of maintaining real-time constraints of all SoftECUs, thus they have their real-time constraints specified. Control of spatial interference is achieved with memory protection.

Further requirements on the specification of SoftECUs must be added for achieving other qualities than interference control. As for the ECUs used today, a functional description, specification of interconnection to specific hardware components, amount and rate of data transferred on the bus.

5. Specifying the Processor Share Requirements

Each SoftECU has to specify the share of the processor that it requires, the share is specified with an arbitrary number of reservations, each reservation as a tuple in the reservation vector **R**. To specify how much

[3] LIN Homepage: http://www.lin-subbus.de
[4] TT-CAN Homepage: http://www.can-cia-de/can/ttcan
[5] Flexray Homepage: http://www.flexray-group.com

processing time a SoftECU requires, and when it requires that time, a specification that consists of two parameters is suggested:

- **R**={<S_D,T,D,J>^{1}…<S_D,T,D,J>^{n}} is the reservation vector, where each of the tuples represents a reservation of a pre-emptive and re-entrant *service time* S_D, that is reserved at a rate T, which is the *period time* of the reservation. Furthermore, each reservation can be reserved with the optional temporal constraints, *deadline* D and *jitter* J. Jitter is a constraint of the periodicity of the activation point expressed as a maximum allowed deviation from the nominal period time, while deadline is the latest point in time relative to each activation when the reserved service time must have been granted. The different parameters of a reservation are visualized in figure 3. We note that the following conditions must be true for the reservation to be valid (S_D <= D <= T) and (J <= T).
- The other part of the specification is the parameter P_D. The parameter is optional but required if processor independency are desired. It represents the speedup of the development platform related to a reference platform, i.e., the speedup of the platform where **R** is valid. Determining the speedup in the general case between two hardware platforms is non-trivial, this work does not provide any solution rather a discussion in a succeeding section, but given that it is possible we show how the temporal reservation in **R** can be handled to become processor independent. Notice that the method does not require P_D, which is the reason why it is separated from the reservation vector.

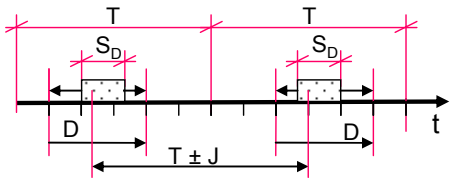

Fig. 3, visualisation of the different attributes in a reservation, specifying a service time S that is reserved at a rate T, with a jitter requirement J, and deadline D. Developers specify SoftECUs processor share requirements with an arbitrary number of reservations in the requirements vector R.

5.1. The Reservation Vector

The service time required from activities that can execute with the same rate in the SoftECU are grouped and assigned a reservation in the reservation vector. However the activities that are grouped must be pre-emptive and re-entrant within the boundaries created by the reservation, since the distribution of the reservation shall be under total control by the RTE. The method allows several such allocations, and can describe complex real-time behavior. Below we discuss how to specify the processor share reservation for periodic, sporadic and single loop activities. However, the basis for the specification is to transform all types of reservations to periodic reservations. A basic condition is that the SoftECUs and the RTE are synchronized and have exactly the same timekeeping, which is not a big deal if services provided by the RTE are utilized.

Some activities may already be of periodic nature, e.g., commonly used real-time tasks. These are the most straight forward type of activities to allocate a reservation for. The service time requirements for all tasks with the same period time are summed and expressed as one allocation. When the characteristics of the application allow, tasks with period times that are multiples of each others can be assigned a reservation with the lowest rate. Let the Fixed Priority Tasks (FPS) tasks in table 1 correspond to all tasks implemented in a SoftECU, two examples of suitable reservations for that SoftECU are R_1 = {<4,5,-,->} and R_2 = {<3,5,-,->, <3,15,-,->}. The resulting run-time behavior for the reservation R_1 is illustrated in figure 4, while the run-time behavior for the reservation R_2 is illustrated in figure 5.

	T	P	C
A	5	H	1
B	5	M	2
C	15	L	3

Table 1, a set of FPS tasks within a SoftECU, the tasks have period time (T), priority (P) High (H), Medium (M), or Low (L), and execution time (C).

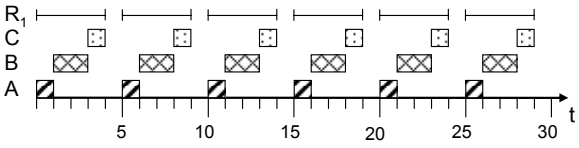

Fig. 4, execution trace of tasks A, B, and C, served with the reservation R_1

In figure 4, the SoftECU has service time reserved for its tasks with a single periodic reservation, resulting in spare capacity for other SoftECUs of one time unit every fifth. In the figure at time 0, the reservation of 4 time units made in R1 is served. Internally in the SoftECU resulting in that the high priority task A executes 1 time unit, followed by the medium priority task B that executes 2 time units. Eventually within the first instance of the allocated service time task C starts to

execute for 1 time unit, but is cut off since the reserved 4 time units has elapsed. The same pattern is repeated for all instances of the reserved service time. Notice that task C, will get its 3 time units of execution with a periodicity of 15 time units as required, but sliced in three different parts.

In figure 5, the SoftECU has service time reserved with two different period times. In the figure at time 0, the reservation of 3 time units every 5th is served. Internally in the SoftECU resulting in that the high priority task A executes 1 time unit, followed by the medium priority task B that executes 2 time units. The reservation of 3 time units every 15th is also served from time 0, and that capacity is used for execution of task C for two time units. At time 5, task A and B becomes ready for execution again and the service time allocated as 3 every 5th is served, A and B use that time, before the low priority task C can execute its remaining 1 time unit. Finally at time 10, service time for three time units are reserved, and used by A and B. The described pattern is repeated every 15th time units.

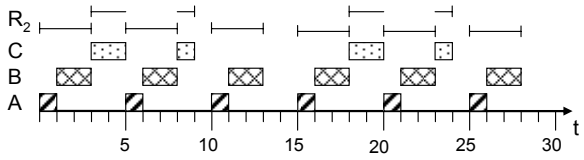

Fig. 5, execution trace of tasks A, B, and C, served with the reservation R_2

Sporadic activities are another base class of activity that often can be identified; they can arrive at the system at an arbitrary point in time with a known maximum arrival rate. Reserving capacity for a sporadic process is done by allocating the capacity for the case with the maximum arrival rate. This is done by setting the period time for the reservation to the maximum arrival rate, when the analysis in conjunction with the integration is performed, i.e., during analysis sporadic activities are treated as periodic activities. While during run-time the RTE has instead of serving the reservation periodically, serve it when a certain event occur but not more often than the reserved capacity.

Another expected type of internal implementation is those implemented as a single cyclic program, called single (or main) loop program. Capacity for these must also be done by a periodic reservation. The period time shall be set to correspond to the cycle time. The length of the reservation shall be the execution time for one cycle in the loop.

Jitter and deadline constraints can be specified for the reservations, they are typically deduced from control applications for performance reasons, and much of the applications in vehicles are related to control activities, e.g., various engine, wheel-spin, and brake-lock control. Typically computer based control applications suffer of unpredictable or to long input to output latencies (sampling-actuation delays) and varying periodicity in the samples (sampling jitter). Input output latency is restricted through the deadline, and constraints on the periodicity through setting maximum allowed jitter. A tool that can be used for simulating these parameters impact on control performance, and find the suitable jitter and deadline constraints for the reservations is JitterBug [8].

5.2. The Speedup

As an option, the temporal reservation for the SoftECUs can be processor independent. It relies on that the speedup ratio between different processors can be determined. That is a non-trivial problem; it is not even clear how processor performance shall be expressed, even less how to determine it [13]. In [18] it is argued that the only consistent measure, when reporting performance of a processor in a single number, is the total execution time. Current state of practice is to determine it through some form of benchmark program, e.g., Whetstone the first major synthetic benchmark [6], or Rhealstone a Real-Time benchmark [7]. In this case it might be possible to determine the speedup through execution or analysis of the SoftECU in question, since it is not a comparison of the processors that are desired rather the speedup for the particular code in the SoftECU. However the actual method to find the speedup is not in focus of this work, but given that it is possible, it is used to achieve processor independent reservation specifications of the SoftECUs.

6. Integration

Integration in a SoftECU based system involves all the engineering work done by the system integrator, it involves all from specifying the functional requirements for the system, its SoftECUs and to verification. In this work, the focus is limited to joining SoftECUs that are verified in isolation to the same platform. Firstly run-time mechanisms addressing the integration problem residing in the RTE are described, followed by the main activities in the integration process; eventually a model describing the effects of the integration on 0the run-time behavior inside SoftECUs is presented.

6.1. The Run-Time Environment

The focus is on the parts of the RTE that address the problems that come from the integration of several

SoftECUs to the same hardware platform, each SoftECU is assumed to be verified by suppliers possibly on hardware platforms different from the integration platform.

Each SoftECU is a single process in the integration platform; the process is scheduled for execution as specified through the processor share requirements. The method to control spatial inference is memory protection, each SoftECU is allocated to an own memory protected process. As the SoftECUs executes in separated processes with separate address spaces, they cannot directly interfere with each others data. However all shared resources, such as any common platform code or data must also be protected, otherwise spatial interference could occur indirectly through that shared resource. Memory protection is practically possible and is common in many systems; it might require hardware support for performance reasons. Such hardware support is implemented in many modern processors [13].

The main run-time mechanism in the integration platform from a temporal view is a processor sharing algorithm, which guarantees that all processes will get the reserved share. It should be possible to use a resource sharing algorithm based on General Processor Sharing (GPS) [11][12], which originally was intended for flow control in gateway nodes. For instance the Stride scheduling algorithm [22], the Earliest Eligible Virtual Deadline First algorithm (EEVDF) [21], and the Earliest completion-time GPS algorithm (EGPS) [19]. However, we propose Fixed Priority Scheduling (FPS) with the simple extension that processes always gets the service time they request and no more. The motivation for FPS before other more advanced scheduling algorithms or processor sharing algorithms, is that the behavior has been widely analyzed [1][2][14][17], it is mature and proven by wide use in commercial products. The application of FPS with static service time as in this case, makes the jitter analysis techniques even simpler.

The RTE should also offer communication and I/O mechanisms, as briefly described in section 3. Suitable extensions in an integration platform for the vehicular industry might be built in monitoring support, for efficient fault localization in workshops. Furthermore, additional mechanisms catering for safety like redundancy and safety kernels.

6.2. Integration Activities

Before integration, or migration to another platform, some engineering activities shall be done. The purpose is to find out if it is possible to integrate the desired SoftECUs on the desired platform, i.e., will the SoftECUs get the share of the processor that is specified through their allocations.

The trick is not only that SoftECUs get a time share equal to the required service time, the time must also be given within certain boundaries. Consider moving SoftECUs to slower platforms, with a single deadline that is equal to the length of the allocated service time, it is impossible to move such a SoftECU to slower platform. However this is detected with ordinary real-time analysis. The basic condition for a platform migration is that SoftECUs get the same computing power within all allocated intervals. A processor with speedup ratio 3, executes the same code three times as fast. The different steps required for integration is listed below:

1. For SoftECUs developed and verified on another platform, find the speedup for the hardware platform used for integration. The result P_I is the speedup for the integration platform relative to the same reference platform, as the speedup for the development platform is relative to. For SoftECUs developed for the integration platform directly, set $P_I=P_D\neq 0$
2. Calculate service time requirements for all reservations relative to the chosen integration platform; replace S_D representing the service time required on the development platform with S_I using equation 1, where S_I represents the service time requirement on the integration platform.

$$S_I = S_D * P_D / P_I \quad \textbf{(1)}$$

3. Calculate offsets [20][10] that are used for controlling the jitter. The offset for an allocation represent the earliest time, relative to the start of each period, when the reserved service time S_I can be serviced. The offset (O) is calculated for all services time reservations with jitter constraints, methods that can be used are, e.g., [4] or [16].
4. Next step is to perform a priority assignment for FPS scheduling of the reservations. This can be done by, e.g., [3], or [16].
5. Eventually, temporal analysis has to be performed; it has been extensively covered in the research community. With exact analysis we can calculate response times for all allocations and verify that all deadlines are met, as if they were real-time tasks. The number of calculations necessary has been reduced for priorities assigned using a deadline monotonic heuristics in [3]. More general FPS analysis techniques with complex constraints is presented in [14].

6.3. Run-Time Behaviour

The parameters supplied by the developer in the reservation vector restrict the window when the reserved time can be serviced. In this section a model of the run-time behavior with respect to parameters in the reservation vector and hardware is presented. The model gives an expression for the time an event generated and taking place inside a SoftECU can occur, given a possible interval of occurrence on the development platform. The model can be used to calculate the impact of integration for different events, e.g., start and completion times for sampling-actuation, or tasks.

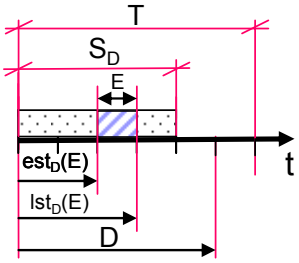

Fig. 6, the relative interval [$est_D(E)$, $lst_D(E)$] for the occurance of an internally generated event E

Figure 6 shows an interval with the earliest start time $est_D(E)$, and latest start time $lst_D(E)$. The interval represents the time of an event E generated internally in the SoftECU, relative to the activation of an instance of reserved service time, when the SoftECU is executed in isolation on the development platform. That is the case when the reserved time can be served immediately utilizing the full capacity of the processor until completion. The occurrence of the event must be expressed as an interval, due to possible variations of execution times for preceding activities.

The earliest start time of the event $est_I(E)$, relative to the activation of an instance of reserved service time, on another hardware platform together with other SoftECUs is illustrated to the left in figure 7. It occur when the SoftECU is served with the reserved time immediately, and execute undisturbed until completion. However, the speedup between the integration platform and the development platform must be considered. The expression is given in equation 2, the relative speedup of the integration platform compared to the development platform (PD/PI) times the start time of the earliest start time of the event on the development platform.

The latest start-time $lst_I(E)$ relative to the activation of an instance of reserved service time is given by equation 3, and illustrated to the right in figure 7. All reserved service time are served before the deadline of the reservation, which is guaranteed through temporal analysis during the integration. Thus, the latest start-time $lst_I(E)$ for an internally generated event is as close to the deadline as possible without violating it. That situation appears when the reservation is exposed to the maximum temporal interference from SoftECUs.

$$est_I(E) = est_D(E) * PD/PI \quad \textbf{(2)}$$

$$lst_I(E) = (D - O) - PD/PI * (S_D + lst_D(E)) \quad \textbf{(3)}$$

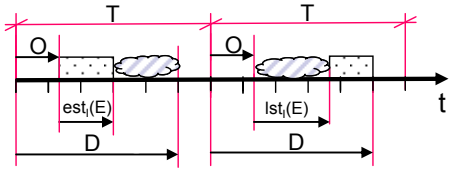

Fig. 7, the earliest and latest occurrence of an internally generated event $est_I(E)$ and $lst_I(E)$, under maximum interference from other SoftECU, and possibly on another processor than the temporal reservation is aimed for

The possible interval for the occurrence of an event E, independent of hardware platform and other SoftECUs is thus given by the interval [$est_I(E)$,$lst_I(E)$]. The fundamental information can be used for deriving expressions for many important temporal run-time characteristics. Consider the start (Ts) and completion time (Tc) of a periodic task in a SoftECU, then [$est_I(Ts)$,$lst_I(Ts)$] and [$est_I(Tc)$, $lst_I(Tc)$] are possible intervals for occurrence of the events that the task starts and finish its execution respectively. It is trivial to determine expressions for e.g., maximum response time max(R) (4), minimum and maximum time between two consecutive activations min(T) (5) and max(T) (6). In figure 8, max(R) max(T), and min(T) is visualized.

$$max(R) = lst_I(Tc) - est_I(Ts) \quad \textbf{(4)}$$

$$max(T) = T + est_I(Ts) - lst_I(Ts) \quad \textbf{(5)}$$

$$min(T) = T + lst_I(Ts) - est_I(Ts) \quad \textbf{(6)}$$

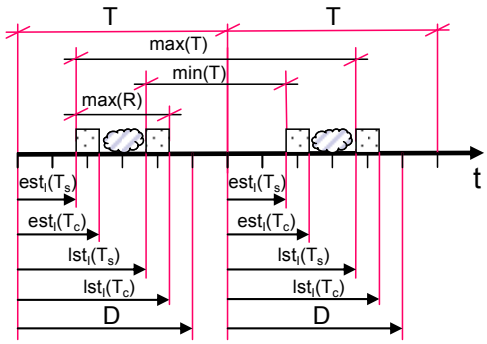

Fig. 8, visualisation of maximum response time max(R), maximum and minimum time and minimum time between two consecutive activations, max(T), and min(T)

7. Conclusions

In this paper we have presented a method for control of interference caused by integration of large real-time software components, denoted SoftECUs. We show how the method can be used with specification of real-time constraints that can be processor independent, given that it is possible to determine the speedup between processors. The usage context and main parts of a component technology using the method is also briefly described.

As future work, the first step is to verify this method in practice, then iterative add and verify different parts towards a full software component model supporting the integration of software components containing different vehicular functions to the same hardware platform.

8. References

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

[2] N. C.Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying New Scheduling theory to Static Priority Pre-Emptive Scheduling. In Software Engineering Journal, pages 284–292, 1993.

[3] N. C. Audsley, A. Burns, M. Richardson, and A. J. Wellings. Deadline monotonic scheduling theory. In Proceedings of 18th IFAC Workshop on Real Time Programming, pages 55--60, Bruges, Belgium, June 1992.

[4] A. Bate, I. Burns. An approach to task attribute assignment for uniprocessor systems. In: Proceedings of the 26th Annual International Computer Software and Applications Conference, IEEE (2002)

[5] I. Crnkovic, and M. Larsson, Building Reliable Component-Based Software Systems, Artech House publisher 2002 ISBN: ISBN 1-58053-327-2

[6] H. J. Curnow, and B. A. Wichmann. A syntehetic benchmark, The Computer Journal, 19(1):80, 1976.

[7] R. Kar and K. Porter, Rhealstone . a Real-Time Benchmarking Proposal, Dr. Dobbs' Journal, February 1989.

[8] B. Lincoln, and A. Cervin, Jitterbug: A Tool for Analysis of Real-Time Control Performance, In *Proceedings of the 41st IEEE Conference on Decision and Control,* Las Vegas, NV, December 2002.

[9] O. Nierstrasz, G. Arévalo, S. Ducasse, R. Wuyts, A. Black, P. Müller, C. Zeidler, T. Genssler, R. van den Born, A Component Model for Field Devices, Proceedings of the First International IFIP/ACM Working Conference on Component Deployment, Germany, June 2002.

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

[11] A. K. Parekh, and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case; IEEE/ACM Transactions on Networking, Volume: 1 , Issue: 3 , June 1993, Pages:344 - 357

[12] A. K. Parekh, and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the multiple node case; IEEE/ACM Transactions on Networking, Volume: 2 , Issue: 2 , April 1994, Pages:137 – 150

[13] D. A. Patterson and J. L. Hennesay, Computer Organization & Design the Hardware / Software Interface, second edition, Morgan Kaufmann Publishers, Inc, 1998, ISBN 1-55860-428-6

[14] O. Redell, M. Törngren. Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In: Proc. Eighth IEEE Real-Time and Embedded Tech-nology and Applications Symposium, IEEE (2002)

[15] K. Sandström, J. Fredriksson, M. Åkerholm, Introducing a Component Technology for Safety Critical Embedded Real-Time Systems. In International Symposium on Component-based Software Engineering (CBSE7) Edinburgh, Scotland, May 2004, Springer Verlag

[16] K. Sandström, C. Norström. Managing complex temporal requirements in realtime control systems. In: In 9th IEEE Conference on Engineering of Computer-Based Systems Sweden, IEEE (2002)

[17] L. Sha, R. Rajkumar, and J. Lehoczky. Task Period Selection and Schedulability in Real-Time Systems.IEEE Transactions on Computer, 39(9), 1990.

[18] J. E. Smith, Characterizing computer performance with a single number, Communications of the ACM, Volume 31, Issue 10 (October 1988), Pages: 1202 – 1206, 1988

[19] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems; 17th IEEE Real-Time Systems Symposium, 1996., 4-6 Dec. 1996; Pages:288 – 299

[20] K. Tindell. Adding Time Offsets to Schedulability Ananlysis. Technical Report, Department of Computer Science, University of Yourk, January 1994.

[21] T.-W. Kuo; W.-R. Yang; K.-J. Lin. EGPS: a class of real-time scheduling algorithms based on processor sharing; Proceedings. 10th Euromicro Workshop on Real-Time Systems, 1998. , 17-19 June 1998, Pages:27 - 34

[22] C. A. Waldspurger, W. E. Weihl. Stride Scheduling: Deterministic Proportional-Share Resource Management Technical Memorandum MIT/LCS/TM-528; MIT Laboratory for Computer Science; Cambridge, MA 02139; June 22, 1995

[23] K. C. Wallnau. Volume III: A Technology for Predictable Assembly from Certifiable Components, Technical report, Software Engineering Institute, Carnegie Mellon University, April 2003, Pittsburgh, USA