

*Title:*

## **Adapting FTT-CAN for the joint dispatching of tasks and messages**

*Authors:*

M.J. Calha, J.A. Fonseca

*Address:*

Escola Superior de Tecnologia

Av do Empresário

6000 Castelo Branco

Portugal

*E-mail address:*

[mjc@est.ipcb.pt](mailto:mjc@est.ipcb.pt)

[jaf@det.ua.pt](mailto:jaf@det.ua.pt)

*Phone:*

351 – 272 339 338

*Fax:*

351 – 272 339 399

*Preferred workshop:*

Fieldbus networks

## ***ABSTRACT***

In the context of a centralized holistic scheduling of tasks and messages, the dispatching can be an issue. In this paper it is presented an architecture with a centralized dispatcher that can be easily integrated in any distributed control system without introducing a significant overhead. A central node dispatches both tasks and messages to other nodes that are only required to possess a nano-kernel. Furthermore the interdependences between producer/consumer tasks and messages are studied, resulting in a set of constraints that guarantee the system feasibility. In the paper it is also shown that this solution is easily implemented using a protocol for message dispatching such as FTT-CAN (Flexible Time-Triggered Communication on CAN).

## **1. Introduction**

Distributed embedded control systems used in automotive applications and in other type of machines and devices are often built recurring to low-processing power off-the-shelf microcontrollers. This option comes essentially from the fact that those components have proven to be robust and predictable due to their simplicity and to the past and long experience in using them. The fact that they usually are offered in the market at a low cost can be also an interesting factor in many applications.

When these simple devices are used, the processing overhead must be kept low in order to achieve the timeliness often required. However, due to the need to execute process dependent tasks such as data acquisition, actuation, pre-processing of raw data, etc, and to handle some communication activities, the microcontrollers can be already significantly loaded. It is then difficult to include additional facilities such as synchronisation or, if multiple tasks must be executed, to integrate adequate kernels, which are required if an overall real-time operation is desired.

It is then considered useful to find a solution to facilitate the operation of real-time distributed embedded systems when they are constructed with that kind of low-processing power devices. To do this, the possibility of controlling the dispatching of the tasks in the different nodes and of the messages in the communications infrastructure would be an interesting issue if this operation is achievable with such simple devices.

The study of fieldbus based distributed systems from a perspective that joins both tasks and messages has already been significantly addressed by the scientific community. A first approach to the verification of end-to-end response times for distributed real-time software systems is the holistic scheduling analysis proposed by Tindell & Clark [1]. They have shown how to analyze

distributed hard real-time systems conforming to a particular architecture. Fault tolerant real-time applications where fault tolerance is realized by groups of tasks is investigated by Bizzarri *et al* [2]. Improved techniques for the schedulability analysis of tasks with precedence relations in multiprocessor and distributed systems have been presented by Palencia & Harbour [4]. These techniques are based on the analysis of tasks with dynamic offsets exploiting the precedence relations in a more accurate way. Chevochot & Puaut [6] take in consideration not only the temporal behaviour of the application tasks but also the behaviour of the run-time support in charge of executing applications. Their work deals with a complex run-time support with fault-tolerance capabilities. Richard *et al* [7] have presented a method to handle complex asynchronous communication relations between tasks in a hard real-time distributed system in order to prove its schedulability using the holistic analysis. The method is based on the unfolding of the generalized precedence graph, resulting in a new equivalent task set (from the schedulability point of view) that can be directly used to validate the application with the classical holistic analysis. An optimal priority assignment for fixed-priority tasks and messages in an automotive computerized system has been presented by Richard *et al* [8]. The architecture is based on multiple fieldbus networks connecting uniprocessor computation units. Tasks and messages are on-line scheduled according to fixed priorities. The priority assignment is performed with a branch and bound algorithm.

None of these works are considering the possibility of controlling, in a simple way and by a centralized device, the dispatching of tasks in the different nodes of a distributed system connected by a fieldbus. In this paper it is shown that this solution can be obtained by slightly adapting the FTT-CAN protocol [3], [5] and [10]. It is also shown that this can be done without changing too much its centralised node called the master. The first data stream requirements, in order to the system to be schedulable from this joint (tasks and messages) perspective, are also derived.

The paper is organised in five further sections. In the next section, it is given an overview of the tasks and messages characteristics when distributed control systems are concerned. The correspondent parameters are also identified and defined. In section 3, the overall system architecture is presented as well as the solution to trigger the task execution in the nodes. In section 4 a brief review of the FTT-CAN protocol is presented and the main adaptations are shown. The requirements of the data streams to achieve schedulability are discussed and illustrated in section 5. Finally, some conclusions and a brief indication of the work in progress close the paper, in section 6.

## 2. Tasks and messages in distributed control systems

### 2.1. Overview

A typical distributed control system acts upon a process by acquiring a signal, processing some data and responding through an actuator. Fig. 1 represents 2 processes, A and B, being controlled by a distributed control system.

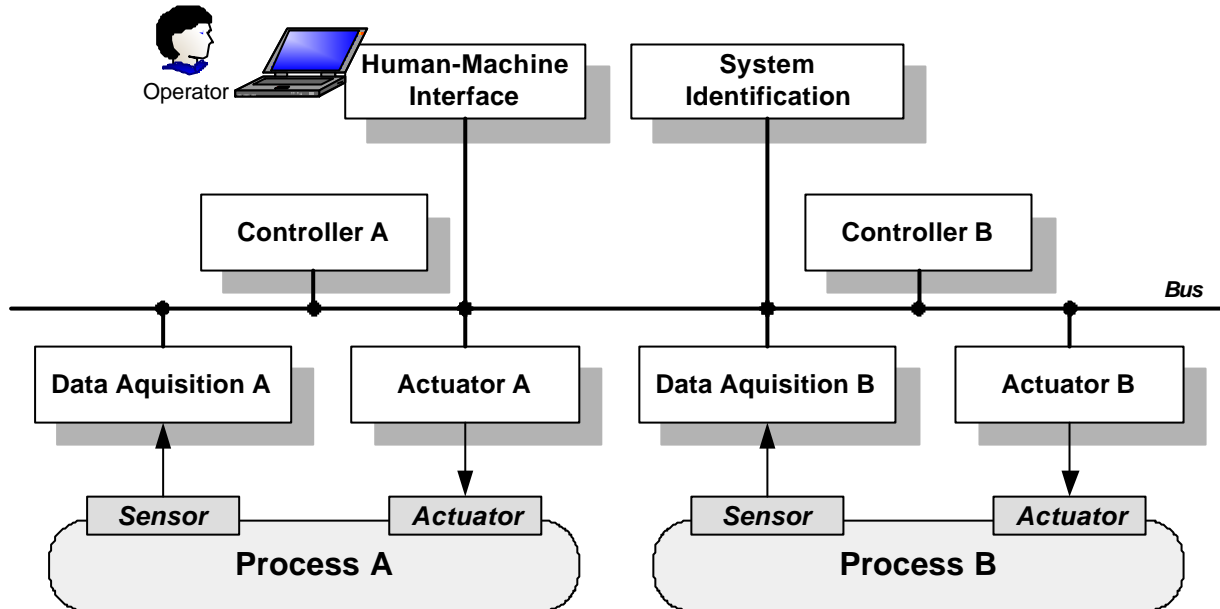


Fig. 1 – Typical distributed control system

A node (depicted by a rectangle) represents a processing unit with private memory. All the nodes are connected with a common bus. At each node there is, at least, a task that produces and/or consumes one or more messages. In this simple example there are several messages that are transmitted across the bus, like the message from *Data Acquisition A* to *Controller A*. There is some probability that the producer tasks try to send a message at approximately the same time, in a way that a collision would occur. To avoid such a collision, some kind of synchronization is required. Now if the system is expanded to a higher number of nodes, each with several tasks, all communicating through the same bus, the probability of collisions is much higher. The problem is how to guarantee a global synchronization without compromising system responsiveness to external events.

In order to describe a system like this, tasks, messages and their relations have to be characterized.

### 2.2. Tasks

In a distributed system, the interaction between tasks can be classified according to 2 basic types:

- Stand-alone,
- Interactive.

In the first type are included the tasks that perform some kind of closed-loop control and don't communicate with other tasks. While the tasks that exchange data with other tasks are included in the second type.

Each interactive task communicates with other tasks in the system, using the message-passing paradigm, and can be decomposed to a simpler form where, at most, they produce and/or consume a single message:

- Producer, the task produces one message;
- Consumer, the task consumes one message;
- Producer/Consumer, the task consumes one message and produces another message.

It is assumed hereafter that the messages are sent at the end of the tasks and received at their beginning.

The set of all the synchronous tasks in the system can be typically expressed as:

$$S_T = \{ ST_i : (C_i, T_i, D_i, Pr_i), i=1..N_{st} \}$$

Where  $N_{st}$  is the number of synchronous tasks, and each task  $ST_i$  is characterized by the following parameters:

- $C_i$  – The worst-case execution time (on each release);
- $T_i$  – The period;
- $D_i$  – The deadline measured relatively to the release instant;
- $Pr_i$  – The priority.

In order to achieve a global synchronization, other parameters have to be defined, namely:

- $N_i$  – Node where the task runs;
- $Ph_i$  – The relative phasing, which determines the first release instant, after system boot;
- $MP_i$  – Message produced (only for interactive tasks);
- $MC_i$  – Message consumed (only for interactive tasks).

Considering the new parameters, the set of all the synchronous tasks in the system can now be expressed as:

$$S_T = \{ ST_i : (C_i, T_i, D_i, Pr_i, N_i, Ph_i, MP_i, MC_i), i=1..N_{st} \}$$

### **2.3. Messages**

Every interactive task uses messages to exchange data with other tasks. Particularly, the periodic message set can be typically expressed as:

$$S_M = \{ SM_m : (C_m, T_m, D_m, Pr_m), m=1..N_{sm} \}$$

Where  $N_{sm}$  is the number of synchronous messages, and each message  $SM_m$  is characterized by the following parameters:

- $C_m$  – The transmission time;

- $T_m$  – The period;
- $D_m$  – The deadline measured relatively to the release instant ;
- $Pr_m$  – The priority.

In order to achieve a global synchronization, other parameters have to be defined, namely:

- $Ph_m$  – The relative phasing, which determines the first release instant, after system boot;
- $PT_m$  – Producer task;
- $CTL_{m,i}$  – Consumer task list.

Considering the new parameters, the set of all the synchronous messages in the system can now be expressed as:

$$S_M = \{ SM_m : (C_m, T_m, D_m, Pr_m, Ph_m, PT_m, CTL_{m,i}), m=1..N_{sm}, i=1..N_{sct} \}$$

Where  $N_{sct}$  is the number of synchronous consumer tasks.

## 2.4. Data streams

A data stream is a group of interacting tasks that starts with a producer task and finishes with a consumer task. Producer/consumer tasks will appear inside the data streams.

To exemplify what a data stream is, let's consider a simple situation with 6 tasks, running in 4 nodes, and 3 messages, originating the following data streams:

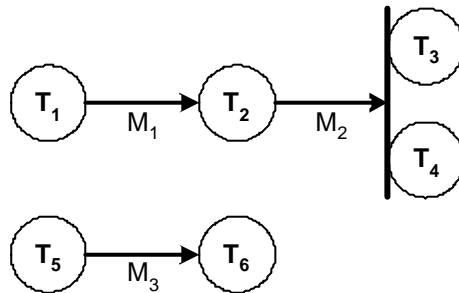


Fig. 2 – Data streams

In the first stream, message  $M_1$  is produced by task  $T_1$  and consumed by task  $T_2$ . Message  $M_2$  is produced by task  $T_2$  and consumed by both task  $T_3$  and task  $T_4$ . In the second stream, message  $M_3$  is produced by task  $T_5$  and consumed by task  $T_6$ .

The analysis of the data streams is of major importance to the definition of the relative phasing,  $Ph$ , of each task and message.

## 2.5. Dispatching tasks and messages: the problem

In order to achieve the desired global synchronization, there is the need of a reliable mechanism to dispatch tasks and messages. Such a mechanism has to guarantee the system feasibility without introducing a significant overhead due: to processing, in each node, and to communication, in the bus.

In the following chapters, a solution for the joint dispatching of tasks and messages is presented.

### 3. Architectural solution for a joint dispatching of tasks and messages

#### 3.1. System architecture

A solution for the problem previously described can be a centralized dispatching of tasks and messages. In order to not overload the existing nodes (Stations), another special node (Master) is added to the system to accomplish this goal.

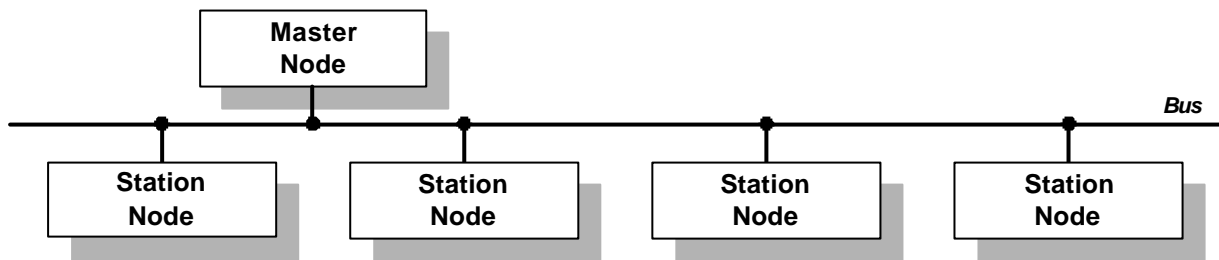


Fig. 3 – Master/station nodes architecture

The Master triggers the execution of tasks in the Stations and the exchange of messages in the bus, in a time-triggered manner.

Each Station node acts upon a trigger event and, in accordance, dispatches any task or message. The Stations have a variable number of tasks to be executed and can produce both synchronous and asynchronous/sporadic messages.

The bus acts as a triggering vehicle for tasks and messages.

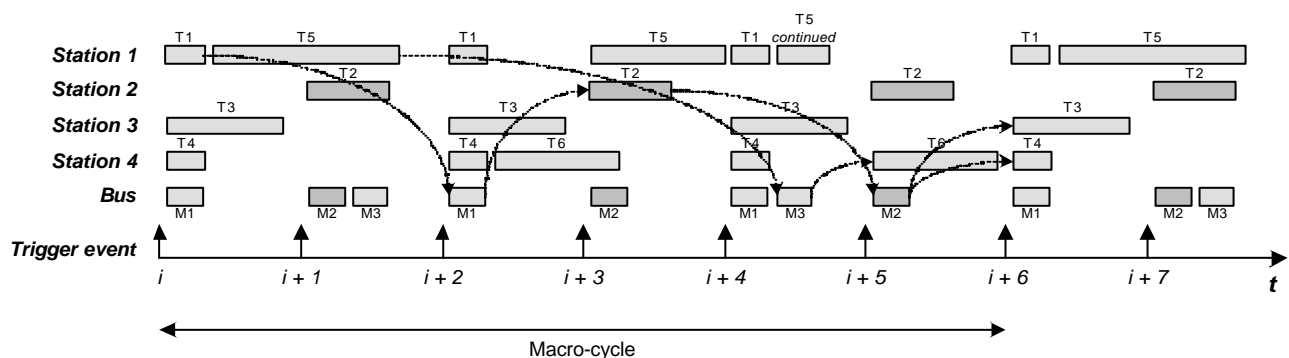


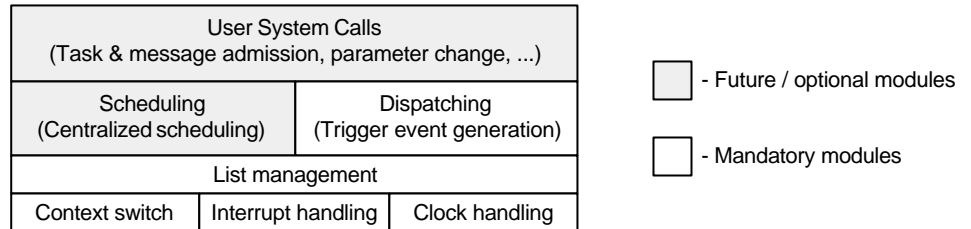
Fig. 4 – Triggering of task execution and message sending

The example in Fig. 4 shows a typical producer/consumer environment with preemption at node level of task 5 by task 1 (with higher priority). Every Station dispatcher and message production is synchronous with the trigger event. The data flow of this scenario is the one depicted in Fig. 2.

### 3.2. Node architecture

In order to present the node architecture, the type and layers of the Master and Station kernels have to be described.

The Master has a micro-kernel with 4 layers as depicted in Fig. 5.

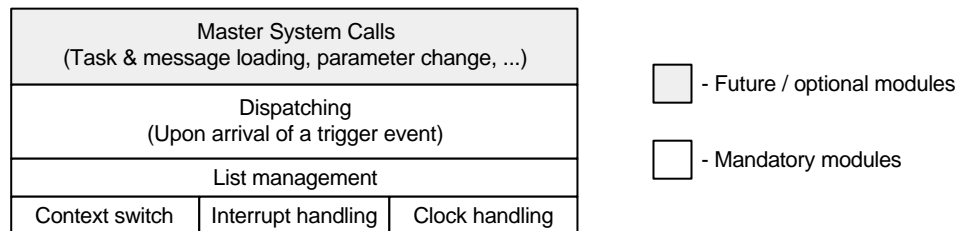


**Fig. 5 – Layers of the micro-kernel in the master node**

Based on a schedule, the Master coordinates the exchange of messages on the bus and the task dispatching at each Station through a special trigger event. The schedule can be created in the Master, or in a different node.

The lower layers (Fig. 5) are responsible for the basic kernel support of the above operations.

Each Station has a nano-kernel with 4 layers as depicted in Fig. 6.



**Fig. 6 – Layers of the nano-kernel in the station node**

Upon a trigger event, the kernel dispatches any task or message that is instructed to. The interval between trigger events becomes the time slice for this kernel.

From the trigger event, various scenarios can occur with task dispatching:

- No task is selected for execution;
- A task is selected for execution, and the Station was idle;
- A task is selected for execution, but there is another task still running;
- More than one task is selected for execution.

Due to the inexistence of a scheduler in the Stations, the kernel just has to act in accordance with the trigger event (see Fig. 7) and execute, respectively, the following procedures:

- Keep doing the same, either idle or running a task;
- Start execution of the selected task;
- Preempt the running task, start execution of the selected task and upon its termination resume the previous task;

- Start the successive execution of each selected task.

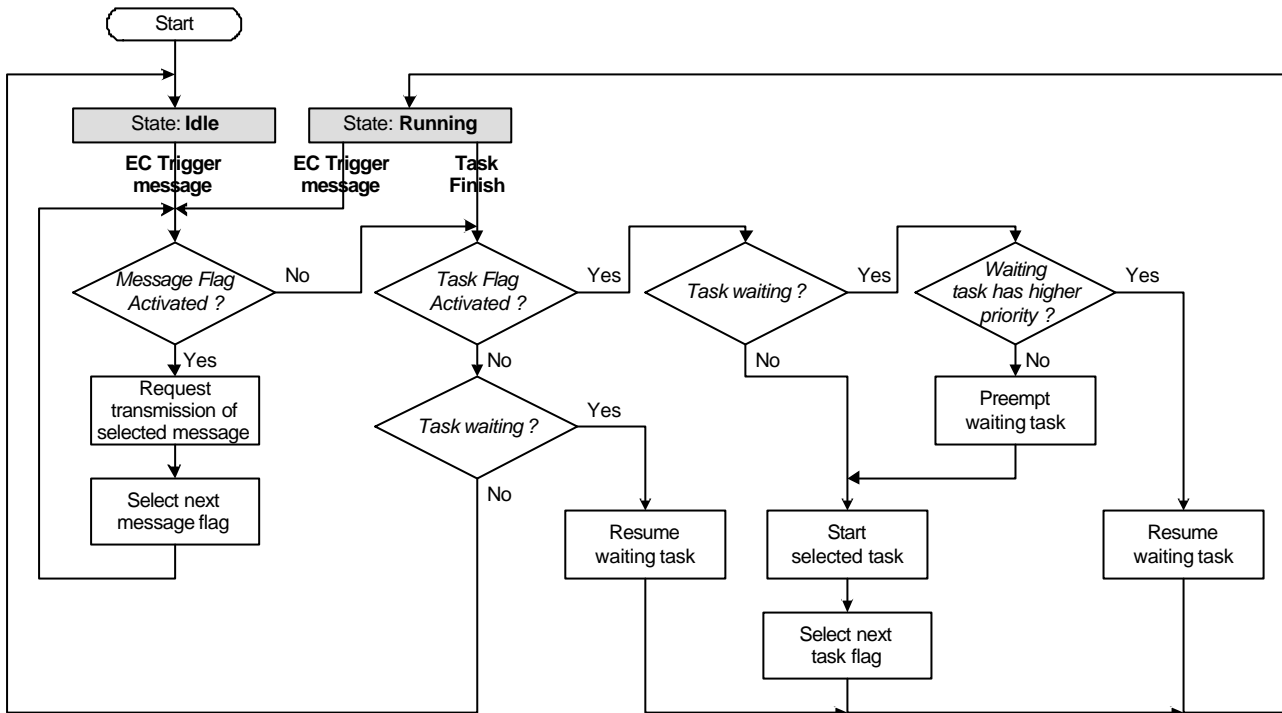


Fig. 7 – Station kernel flowchart

Task preemption can occur as planned in the schedule. This possibility allows the system to achieve a high responsiveness permitting higher priority tasks to preempt others.

The existence of more than one task selected for execution is a way to use a possible dead time between the end of the execution of the first task and the next trigger event. The order of execution of a set of selected tasks is pre-determined and can take into consideration characteristics like: priority, execution time, period, deadline, etc...

From the trigger event, any message selected for dispatching instructs the kernel to transmit that message.

The granularity of the trigger event should be tuned with system parameters like: task and message periods, average size of tasks, average size of messages, number of tasks and messages, etc...

Tasks and messages can be loaded through the common bus, maintaining the connectivity needs to a minimum.

## 4. Using FTT-CAN for task dispatching

### 4.1. Brief review of FTT-CAN

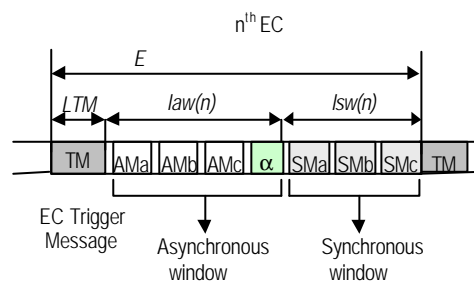
The basis for the FTT-CAN protocol (Flexible Time-Triggered communication on CAN) has been first presented in [3]. Basically, the protocol makes use of the dual-phase elementary cycle concept in order to combine time and event-triggered communication with temporal isolation. The time-

triggered traffic is scheduled on-line and centrally in a particular node called master. This feature facilitates the on-line admission control of dynamic requests for periodic communication because the respective requirements are held centrally in just one local table. With on-line admission control, the protocol supports the time-triggered traffic in a flexible way, under guaranteed timeliness (dynamic planning-based scheduling paradigm).

The FTT-CAN takes advantage of the native MAC of CAN to reduce communication overhead and support a high efficiency and flexibility in the time-triggered traffic. The protocol relies on a relaxed master-slave medium access control in which the same master message triggers the transmission of messages in several slaves simultaneously (master/multi-slave). The eventual collisions between slaves' messages are handled by the native distributed arbitration of CAN. The protocol also takes advantage of the CAN arbitration to handle event-triggered traffic in the same way as the original protocol does. Particularly, there is no need for the master to poll the slaves for pending event-triggered requests. Slaves with pending requests may try to transmit immediately, as in normal CAN, but just within the respective phase of each elementary cycle. This scheme allows a very efficient combination of time and event-triggered traffic, particularly resulting in low communication overhead and shorter response times.

The nomenclature used in the protocol follows. In FTT-CAN the bus time is slotted in consecutive Elementary Cycles (ECs) with fixed duration. All nodes are synchronised at the start of each EC by the reception of a particular message known as *EC trigger message*, which is sent by a particular node called *master*.

Within each EC the protocol defines two consecutive windows, asynchronous and synchronous, that correspond to two separate phases (Fig. 8).



**Fig. 8 – The Elementary Cycle in FTT-CAN**

The former one is used to convey event-triggered traffic, herein called *asynchronous* because the respective transmission requests can be issued at any instant. The latter one is used to convey time-triggered traffic, herein called *synchronous* because its transmission occurs synchronously with the ECs. The schedule for each EC is conveyed by the respective EC trigger message. Since this window is placed at the end of the EC, its starting instant is variable and it is also encoded in the

respective EC trigger message. The asynchronous window has a duration equal to the remaining time between the EC trigger message and the synchronous window.

An in-depth analysis of the FTT-CAN protocol is available in [10].

## ***4.2. How it can be adapted to trigger tasks***

The FTT-CAN protocol relies in the system operator to correctly characterize the messages. Parameters like the message period are dependent on the worst case execution time of both the producer task, and the consumer task.

Each task has its own characteristics, namely: execution time, running node and type of interaction with other tasks. So in order to achieve a feasible schedule for the messages, tasks have also to be considered in a holistic scheduling.

In a system with several tasks running in several nodes, a node may have more than one task assigned. The problem is how to schedule tasks in order to accomplish the intended global schedulability without increasing the system complexity, i.e. maintaining a low overhead.

To accomplish the goal the EC trigger message must be redesigned to accommodate in its data field, apart from the synchronous messages that must be transmitted and additional coding, also which tasks must be started in the current EC. This way, the EC trigger message can be used to also trigger remote task execution.

## ***4.3. What restrictions have we***

The trigger message data field has now to accommodate two flag areas, one for tasks and another for messages. Each bit from the task data field is a flag that indicates whether a task must be dispatched in the current EC, or not. In a similar way, each bit from the message data field is a flag that indicates whether a message must be transmitted in the current EC, or not.

Recalling the example from Fig. 4, the trigger message could have a data field with 2 bytes, one byte for tasks and another one for messages, as shown in Fig. 9.

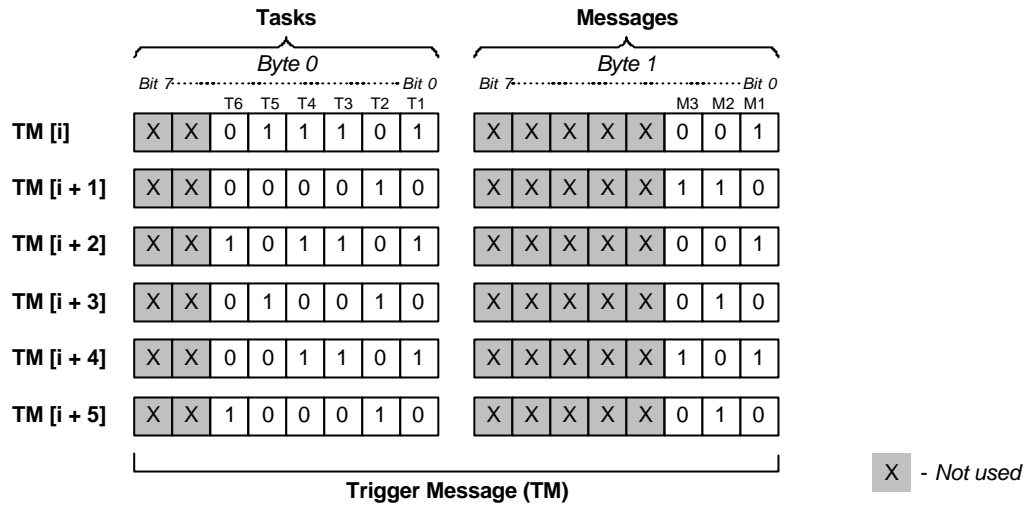


Fig. 9 – EC Trigger Message data contents

The definition of the data field can be done during the initialization phase but if the sum, of tasks, synchronous messages and codes, exceeds 63 (limit of CAN2.0A) then an extended trigger message can be used. For small systems one trigger message should be enough, but for larger systems various extended trigger messages could be used.

The discretization of time imposed by the EC trigger message puts a constraint over the task and message parameters:

- Tasks –  $T_i$  and  $Ph_i$  have to be rounded to an EC multiple (note that  $C_i$  can be less than the EC duration);
- Messages –  $T_m$  and  $Ph_m$  have to be rounded to an EC multiple (this restrictions were already presented and thoroughly discussed in FTT-CAN based systems).

## 5. Requirements and pre-checking

### 5.1. Parameter restrictions

In order to define the parameter restrictions it's assumed that these are message-based, which means that the messages impose restrictions to the tasks.

Some tasks and messages parameters have to be defined at creation time, namely:

- Stand-alone tasks – all except  $Ph_i$  ;
- Interactive tasks –  $C_i$  and  $Pr_i$  ;
- Messages – all except  $Ph_m$  .

The node of execution of a task, parameter  $N_i$ , might not be specified by the user, meaning that the Master is free to allocate it to the best Station. The definition of the best Station is beyond the scope of this article, but several aspects can be considered like: load balancing, communications

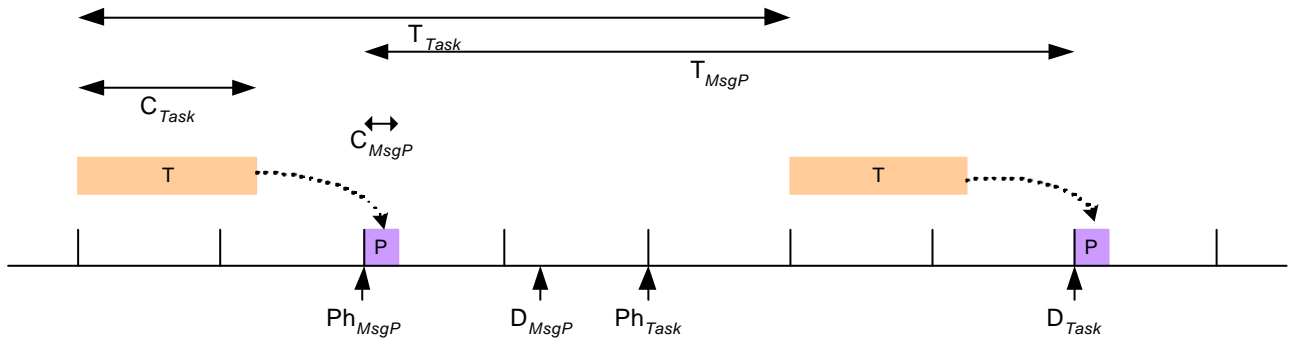
performance, availability, usage of unique hardware or software capabilities, etc... All the other parameters (of tasks and messages) are determined during analysis phase.

In order to generate the restrictions for the tasks, it's necessary to define the equations that permit the calculus of the tasks parameters based on the messages parameters. These equations are based on the assumption that, for all messages, the deadline is never greater than the period. For the determination of the tasks deadline, a function that rounds-up a value to the nearest EC multiple,  $CeilIEC()$ , is used.

According to the type of the task, several restrictions apply in dependence of the messages produced and/or consumed, namely:

- Stand-alone tasks
  - No restrictions;
- Producer tasks

The task deadline ( $D_{Task}$ ), which is measured relative to the task release instant, must occur in the EC before the release instant of the message being produced ( $Ph_{MsgP}$  in the case of the first release). This guarantees that the task has finished and the produced data has already been delivered to the kernel when the EC trigger message, that dispatches the message, arrives.



**Fig. 10 – Task produces a message**

Considering a producer task T that produces a message P (Fig. 10), the restrictions are:

- $T_{Task} = T_{MsgP}$
- $D_{Task} = CeilIEC( C_{Task} + (T_{MsgP} - CeilIEC(D_{MsgP})) )$
- $Ph_{Task} = Ph_{MsgP} - D_{Task}$

It can be understood from these equations that when the  $D_{MsgP}$  comes closer to the value of  $T_{MsgP}$ , the  $D_{Task}$  approaches the value of  $C_{Task}$ . So we can conclude that when the

transmission window of the message ( $MsgP$ ) widens, the execution window of the task ( $Task$ ) narrows, and vice-versa.

The relative phasing of the task is, logically, inferior to the relative phasing of the message to be produced.

➤ Consumer tasks

The task release instant ( $Ph_{Task}$  in the case of the first release), must occur in the EC after the deadline of the message being consumed ( $D_{MsgC}$ ). This guarantees that the message has been transmitted and the data to be consumed has already been delivered to the kernel when the EC trigger message, that dispatches the task, arrives.

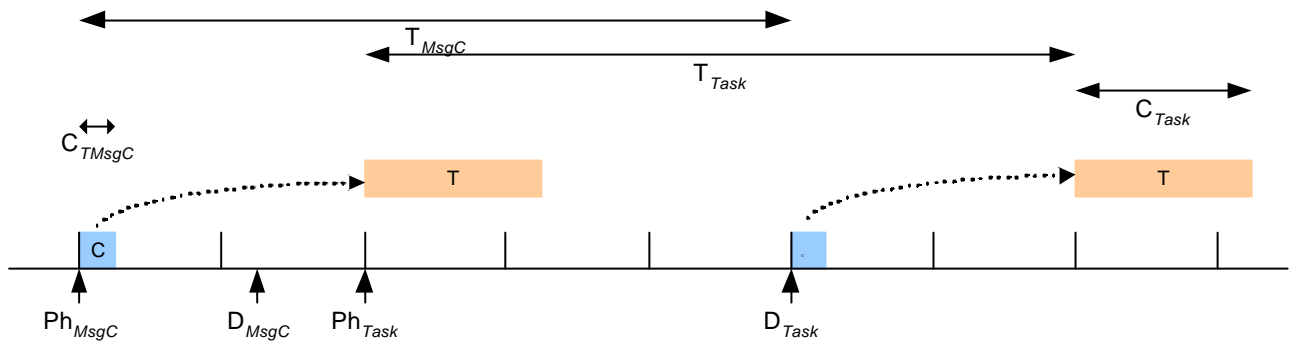


Fig. 11 – Task consumes a message

Considering a producer task  $T$  that consumes a message  $C$  (Fig. 11), the restrictions are:

- $T_{Task} = T_{MsgC}$
- $D_{Task} = \text{CeilIEC}( C_{Task} + (T_{MsgC} - \text{CeilIEC}(D_{MsgC})) )$
- $Ph_{Task} = Ph_{MsgC} + D_{MsgC}$

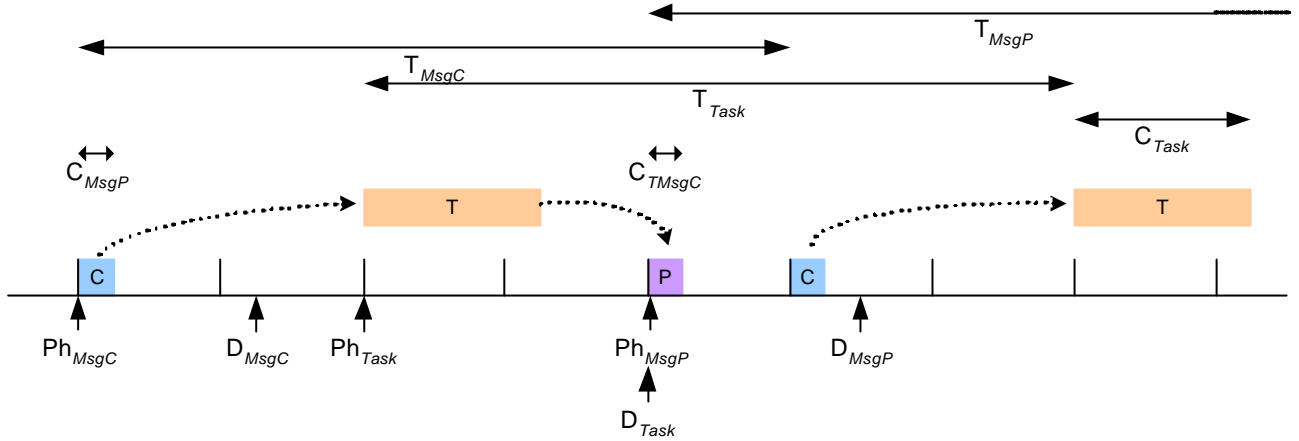
It can be understood from these equations that when the  $D_{MsgC}$  comes closer to the value of  $T_{MsgC}$ , the  $D_{Task}$  approaches the value of  $C_{Task}$ . So we can conclude that when the transmission window of the message ( $MsgC$ ) widens, the execution window of the task ( $Task$ ) narrows, and vice-versa.

The relative phasing of the message to be consumed is, logically, inferior to the relative phasing of the task.

➤ Producer/consumer tasks

The task release instant ( $Ph_{Task}$  in the case of the first release), must occur in the EC after the deadline of the message being consumed ( $D_{MsgC}$ ). This guarantees that the message has been transmitted and the data to be consumed has already been delivered to the kernel when the EC trigger message, that dispatches the task, arrives.

The task deadline ( $D_{Task}$ ), which is measured relative to the task release instant, must occur in the EC before the release instant of the message being produced ( $Ph_{MsgP}$  in the case of the first release). This guarantees that the task has finished and the produced data has already been delivered to the kernel when the EC trigger message, that dispatches the message, arrives.



**Fig. 12 – Task consumes and produces a message**

Considering a producer/consumer task  $T$  that consumes a message  $C$  and produces a message  $P$  (Fig. 12), the restrictions are:

- $T_{Task} = T_{MsgC} = T_{MsgP}$
- IF(  $D_{MsgP} \geq D_{MsgC}$  )
  - $D_{Task} = \text{CeilIEC}( C_{Task} + (T_{MsgP} - \text{CeilIEC}(D_{MsgP})) )$
  - $Ph_{Task} = Ph_{MsgP} - D_{Task}$
- IF(  $D_{MsgP} < D_{MsgC}$  )
  - $D_{Task} = \text{CeilIEC}( C_{Task} + (T_{MsgP} - \text{CeilIEC}(D_{MsgC})) )$
  - $Ph_{Task} = Ph_{MsgC} + D_{MsgC}$

It can be understood from these equations that in dependence of the deadlines of the produced and consumed messages, one set of the equations will apply. This means that the message with the largest transmission window will prevail when defining the execution window of the task.

The relative phasing of the message to be consumed is, logically, inferior to the relative phasing of the task, and on the other hand, the relative phasing of the task is, logically, inferior to the relative phasing of the message to be produced, which can be expressed as:

$$Ph_{MsgC} \leq Ph_{Task} \leq Ph_{MsgP}$$

## 5.2. Precedence analysis

Before the precedence analysis can take place, it's necessary to build all the data streams. As in Fig. 2, these data streams comprise all messages and interactive tasks.

The precedence analysis is the iterative procedure that, on each data stream, defines the relative phasing,  $Ph$ , of each interactive task and message based on the accumulated execution time.

For the example that is being used (Fig. 2, Fig. 4 and Fig. 9) the resulting relative phasings are:

Task/Msg Id	Station	C (EC)	T (EC)	D (EC)	Ph (EC)
T1	1	0.30	2	2	0
T2	2	0.60	2	2	3
T3	3	0.80	2	2	6
T4	4	0.25	2	2	6
T5	1	1.40	3	4	0
T6	4	0.90	3	3	5
M1	-	0.30	2	1	2
M2	-	0.30	2	1	5
M3	-	0.30	3	1	4

Table 1 – Tasks and messages parameters

In this table, for simplicity, all the values are expressed relatively to the EC duration. The tasks and messages are listed by decreasing order of priority. The shaded cells have been calculated.

After this analysis, a comparison between the relative phasing requested by the operator and the relative phasing determined by the precedence analysis has to be done. If there is any mismatch then the operator should be called for intervention.

With the precedence analysis, the need to distinguish between the Start-up interval and the Macro-cycle arises. So the Start-up interval covers the time between the first EC trigger message and the EC trigger message corresponding to the highest relative phasing of all tasks and messages,  $MaxPh$ , as given by the following expression:

$$MaxPh = \text{MAX}( Ph_i, Ph_m ), i=1..N_{st}, m=1..N_{sm}$$

Where MAX is a function that calculates the highest value of a set of values,  $N_{st}$  is the number of synchronous tasks and  $N_{sm}$  is the number of synchronous messages.

The Macro-cycle is an interval with a pattern, of tasks and messages, that will be repeated indefinitely, until one of the tasks finishes or an error occurs.

The relation between the Start-up interval and the Macro-cycle is shown in Fig. 13.

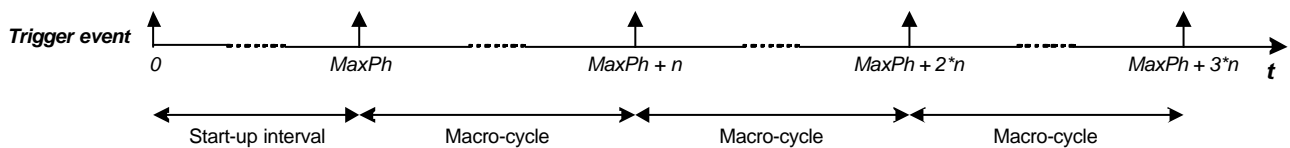


Fig. 13 – Start-up interval and Macro-cycles



## 7. References

- [1] K. Tindell, J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", *Microprocessing & Microprogramming* 40, 117-134, 1994.
- [2] P. Bizzarri, A. Bondavalli, F. Giandomenico, F. Tarini, "Planning the Execution of Task Groups in Real-Time Systems", 8th IEEE Euromicro Workshop on Real-Time Systems (WRTS'96), L'Aquila, Italy, June 1996.
- [3] Almeida L., J. Fonseca, P. Fonseca, "Flexible Time-Triggered Communication on a Controller Area Network", Work-In-Progress Session of 19<sup>th</sup> IEEE Real-Time Systems Symposium (RTSS'98), Madrid, Spain, December 1998.
- [4] J.C. Palencia, M.G. Harbour, "Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems", 20th IEEE Real-Time Systems Symposium (RTSS'99), Phoenix, USA, November 1999.
- [5] Almeida L., J. Fonseca, "FTT-CAN: A Network-Centric Approach for CAN-based Distributed Systems", 4th IFAC Symposium on Intelligent Components and Instruments for Control Applications (SICICA'00), Buenos Aires, Argentina, September 2000.
- [6] P. Chevochot, I. Puaut, "Holistic Schedulability Analysis of a Fault-Tolerant Real-Time Distributed Run-time Support", 7th IEEE International Conference on RealTime Computing Systems and Applications (RTCISA'00), Cheju Island, South Korea, December 2000.
- [7] P. Richard, F. Cottet, M. Richard, "On-line Scheduling of Real-Time Distributed Computers With Complex Communication Constraints", 7<sup>th</sup> International Conference on Engineering of Complex Computer Systems (ICECCS'01), Skovde, Sweden, June 2001.
- [8] M. Richard, P. Richard, F. Cottet, "Task and Message Priority Assignment in Automotive Systems", 4th IFAC Conference on Fieldbus Technology (FET'01), Nancy, France, November 2001.
- [9] E. Martins, P. Neves, J. A. Fonseca, "Architecture of a Fieldbus Message Scheduler Coprocessor Based on the Planning Paradigm", *Microprocessors and Microsystems*, Vol. 26, Issue 3, April 2002.
- [10] L. Almeida, P. Pedreiras, J.A. Fonseca, "The FTT-CAN protocol: Why and How", *to appear in a special issue on factory communication systems*, IEEE Transactions on Industrial Electronics, 2002.