

SIMHOL – A GRAPHICAL SIMULATOR FOR THE JOINT SCHEDULING OF MESSAGES AND TASKS IN DISTRIBUTED EMBEDDED SYSTEMS

Mário J. Calha

Politechnic Institute of Castelo Branco - EST
Av. Do Empresario
6000 Castelo Branco, Portugal

José Alberto Fonseca

Dep. Electronic and Telecommunications
Of University of Aveiro
Campus Santiago
3800 Aveiro, Portugal

Abstract: In this paper a simulator to preview the timeliness of the transmission of messages and of the execution of tasks in a distributed system is presented. The simulator, called SIMHOL, builds on previous work by the authors in which a simple mechanism to dispatch tasks and messages was proposed for CAN-based distributed systems. The inputs to the simulator are the so-called data streams, which include the producer tasks, the correspondent messages and the tasks that use the transmitted data. Using the worst-case execution time and transmission time, the simulator is able to verify if deadlines are fulfilled in every node of the system and in the network. Besides discussing the simulator construction and operation, the paper presents some examples of distributed systems requirements and the results obtained by using the tool to analyze the respective timeliness. This is also used to illustrate the outputs of the simulator. One important issue also discussed is the easy way, due to the object oriented approach chosen, to extend the simulation to cover different networks and to use different scheduling techniques either at the task level or at the message level. *Copyright © 2003 IFAC*

Keywords: Simulators, Scheduling, Algorithms, Fieldbus.

1. INTRODUCTION

In previous works we've proposed a simple mechanism for distributed systems based in nodes with low computational power that allowed the dispatching of tasks and messages in a time-triggered manner (Calha and Fonseca 2002; Fonseca, *et al.*, 2002). These works led to data flow analysis and to the derivation of requirements to the tasks and messages parameters.

The aim of this work is to demonstrate the validity of the proposed analysis and requirements presenting a simulator.

The proposed simulator, SimHol, is an experimental platform for a centralized task and message scheduling. The supported architecture is based in a distributed control system with a centralized dispatcher. A central node dispatches both tasks and messages to other nodes.

For this work, other simulators were studied.

Henderson, *et al.*, (1998) have developed a design tool, Xrma, that supports the schedulability analysis of hard, uni-processor and distributed real-time systems. Xrma automates the analysis and supports the performance verification of diverse real-time systems composed of tasks executing on multiple processors which communicate using the CAN fieldbus.

Vector Informatik GmbH has developed CANalyzer/DENalyzer (Vector Informatik GmbH, 2002) that is a universal development tool for bus systems. CANalyzer/DENalyzer conforms to CAN specification V 2.0 B and makes it easy to observe, analyze and add to bus traffic on as many as 32 channels.

Our approach is different because this simulator supports different architectures and scheduling algorithms. The SimHol uses the flexible time-

triggered (FTT) paradigm for scheduling both tasks and messages.

This paper is organized as follows. Section 2 details some background information about the use of the FTT-CAN to the joint dispatching of tasks and messages. Section 3 explains the SimHol internals. Section 4 explains how to use the simulator and describes some experiments. In the last section some conclusions are drawn and new directions are presented.

2. USING FTT-CAN TO THE JOINT DISPATCHING OF TASKS AND MESSAGES

In a typical distributed control system a process is controlled by a closed loop, where a signal is acquired, some data is processed and an actuator is activated. Each node in a system 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 a real world system there are, usually, several tasks communicating. 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. In a distributed system, the interaction between tasks can be classified according to 2 basic types: stand-alone and 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.

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; Phi – 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:

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

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 maximum 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.

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. An example of data streams is shown in Fig. 1. The analysis of the data streams leads to the definition of the relative phasing, Ph , of each task and message. For this calculus the C_i , D_i , C_m and D_m must be considered. For each stream these task and message parameters must be summed resulting in an accumulated execution and transmission times. This accumulated execution time is the worst-case time because the deadlines are considered.

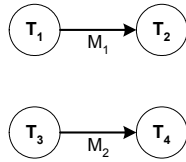


Fig. 1. Examples of data streams.

In a data stream a task may communicate with several tasks with the same message. This means that either the message should be consumed by every consumer task, or it must be buffered in every node with a consumer task for the message.

In order to guarantee a global (tasks and messages) synchronization a reliable mechanism to dispatch both tasks and messages is required. In (Calha and Fonseca, 2002), a distributed architecture and a dispatching scheme were proposed to solve the problem.

The architecture uses a special node, known as the Master, which is added to the remaining nodes (Stations) that form the distributed control system (Fig. 2). The Master triggers the execution of tasks in the Stations and the exchange of messages in the network, in a time-triggered manner.

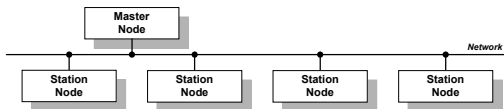


Fig. 2. Distributed system architecture showing the Master and Stations.

Each Station node acts upon a trigger event and, in accordance, dispatches any task or message. The Stations can have a variable number of tasks to be executed and can produce both synchronous and asynchronous/sporadic messages (Fig. 3). The network acts as a triggering vehicle for tasks and messages.

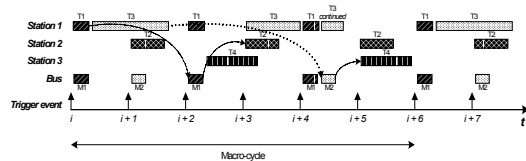


Fig. 3. Triggering of task execution and message sending.

Currently this architecture can be implemented using the Controller Area Network (CiA, 1994; Tindell, *et al.*, 1994) and the FTT-CAN protocol (Almeida, *et al.*, 2002) to trigger the dispatching. It is also possible to use FTT-Ethernet (Pedreiras, *et al.*, 2002) for the same purpose.

A common property of the possible implementations is the fixed periodicity of the trigger events. The period value, usually

designated by EC – Elementary cycle, imposes restrictions to some of the parameters of tasks and messages. This is the case of the periods (T_i and T_m), the deadlines (D_i and D_m) and the relative phasing (Φ_i and Φ_m) which must be integer multiples of the EC.

A Simulator, SimHol, that has as input a system description using the above entities (nodes, tasks and messages) and allows the task and message scheduling is presented in the next chapter.

3. SIMHOL

The main concern for the development of this simulator was to use an object oriented approach due to its well known advantages. Two sets of requirements were identified, namely: external and internal.

The external requirements were:

- Simple interface;
- Input scenarios using plain text files;
- All simulation data should be output to plain text files;
- Fast execution time.

The internal requirements were:

- To allow the integration of different node interconnection technologies, currently Controller Area Network (CAN) is already built-in;
- To allow the integration of different algorithms for the scheduling of tasks and messages, currently Rate Monotonic (RM) and Earliest Deadline First (EDF) are already built-in;
- Function operation as close to actual working system as possible;
- Clear separation between the core elements and the user interface, allowing an easy porting to other platforms;
- Exception handling, providing a good fault coverage resulting in a smooth operation.

The static diagram is organized in four areas:

- Entities,
- Algorithms,
- Architectures and
- Simulator

The entities are: nodes, tasks and messages.

The static diagram showing simplified classes (stripped of more internal attributes and functions) is depicted in Figs. 4, 5 and 6.

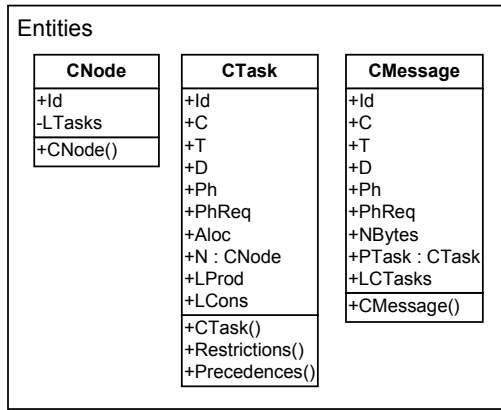


Fig. 4. Static diagram of entities.

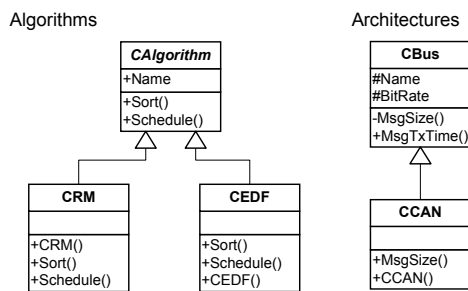


Fig. 5. Static diagram of algorithms and architectures.

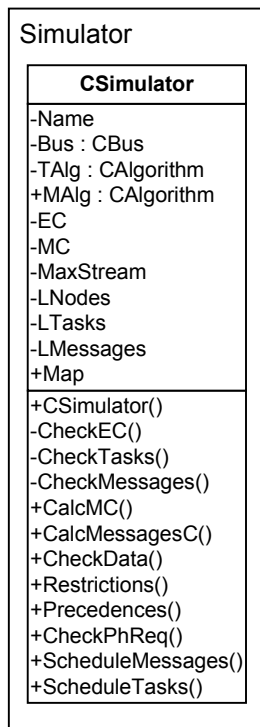


Fig. 6. Static diagram of the simulator.

2.1. Object oriented language

Due to the speed advantage, the simulator was developed using the C++ programming language instead of Java, but a future Java implementation is being considered due to its benefits in what concerns portability. All the features of exception handling were used to cover the critical areas of the code.

2.2. Scheduler

The scheduler is invoked every EC. At every invocation the scheduler considers all entities to be scheduled. This can be either the tasks of a node or the messages to be transmitted. The simulator offers the possibility of using features such as:

- To impose deadlines greater than the periods, and to set the maximum number of delayed tasks/messages;
- To set priority inversion on/off;
- To break the task execution for several ECs, that may not be adjacent;
- To break the message transmission for several ECs, that may not be adjacent.

With this simulator it is possible to test the system allowing priority inversion, or not. If priority inversion is allowed then the simulator tries to use the remaining time in each EC to fit whatever message is waiting to be transmitted, independently of its priority. If priority inversion is not allowed then if the highest priority message, which is ready, can't fit in the remaining time in the EC, this interval is left unused (inserted idle time).

If a task is allowed to be executed across several ECs, than the MayBreak parameter is set to TRUE. Because of this a task may be pre-empted due to the arrival of another task with higher priority. In this situation the context switch overhead must be considered.

If a message is allowed to be transmitted across several ECs, than the MayBreak parameter is set to TRUE. In this situation the kernel has to be aware of this possibility and break the message in smaller pieces. This overhead has to be considered. This technique may be acceptable when the majority, and most frequently used, messages are short and a few, and rarely used, are much longer. In this case the EC might be chosen to best suit the needs of the short messages but without this possibility the transmission of long messages might become impossible.

The flowcharts for the task scheduling are depicted in Fig 7.


```

class CAlgorithm
{
protected:
    AnsiString Name;

public:
    AnsiString GetName() { return Name; }
    virtual void Sort(TList*) {};
    virtual unsigned Schedule(TList*, unsigned*,
        unsigned*, unsigned*, unsigned, unsigned, char*, char,
        char) {return 0;}
};

```

One of the algorithm classes, in this case the CRM (refers to the Class Rate Monotonic), is defined as:

```

class CRM : public CAlgorithm
{
public:
    CRM() { Name="RM"; }
    void Sort(TList*);
    unsigned Schedule(TList*, unsigned*, unsigned*,
        unsigned*, unsigned, unsigned, char*, char, char);
};

```

2.3.3 Connecting with other software

Due to the plain text and normalized data output to files, it's easy to develop tools that grab this data and make further analysis allowing a simple integration on a broader software suite. The Fig. 9 shows an extract of an output file containing simulation data.

```

--- Nodes List ---
N Id: 1
N Id: 2
N Id: 3
N Id: 4

--- Tasks List ---
T Id: 1 C: 520( 1) T: 3120( 2) D: 3120( 2) PhReq: 0( 0)
T Id: 2 C: 1040( 1) T: 3120( 2) D: 3120( 2) PhReq: 0( 0)
T Id: 3 C: 1248( 1) T: 3120( 2) D: 3120( 2) PhReq: 0( 0)
T Id: 4 C: 390( 1) T: 3120( 2) D: 3120( 2) PhReq: 0( 0)
T Id: 5 C: 2184( 2) T: 4680( 3) D: 6240( 4) PhReq: 0( 0)
T Id: 6 C: 1404( 1) T: 4680( 3) D: 4680( 3) PhReq: 0( 0)

--- Messages List ---
M Id: 1 C: 520( 1) T: 3120( 2) D: 1560( 1) PhReq: 0( 0)
M Id: 2 C: 520( 1) T: 3120( 2) D: 1560( 1) PhReq: 0( 0)
M Id: 3 C: 520( 1) T: 4680( 3) D: 1560( 1) PhReq: 0( 0)

--- Scheduling Map ---
EC T T N T T N T T N H H H Bus(%)
0 1 5 100 0 0 0 0 0 0 0 0 0 0
1 0 5 73 0 0 0 0 0 0 0 0 0 0
2 1 0 33 0 0 0 0 0 0 0 1 0 0 33
3 0 5 100 2 66 0 0 0 0 0 0 0 0
4 1 5 73 0 0 0 0 0 0 0 1 0 3 66
5 0 0 0 2 66 0 0 0 6 90 0 2 0 33
6 1 5 100 0 0 3 80 4 0 25 1 0 0 33
7 0 5 73 2 66 0 0 0 0 0 0 2 3 66
8 1 0 33 0 0 3 80 4 6 100 1 0 0 33
9 0 5 100 2 66 0 0 0 6 15 0 2 0 33
10 1 5 73 0 0 3 80 4 0 25 1 0 3 66
11 0 0 0 2 66 0 0 0 6 90 0 2 0 33

--- Execution Window Map ---
EC T T N T T N H H H
0 1 5 0 0 0 0 0 0 0 0
1 255 5 0 0 0 0 0 0 0 0

```

Fig. 9. Example of an output file.

The output file is organized in two main areas: entities and maps. The first includes the characteristics of all nodes, tasks and messages in the system. The second includes the scheduling map and the execution window map (described in section 4). These maps are organized in the following way. The first column shows the current EC. The node

columns (N) relate the sum of the tasks execution time in the current EC to the EC time. The last column (Bus) shows the bus utilization that relates the sum of the messages transmission time in the current EC to the EC time. The task columns (T) are organized according to the node where they have been allocated and are displayed to the left of their node.

In the next chapter a simple example is used to demonstrate the SimHol operation.

4. EXPERIMENTS

In order to begin a simulation a scenario file has to be loaded. This file includes the details of the simulation and the characteristics of every node, task and message. See an extract in Fig. 10.

```

# Bus type
# B Type BitRate(b/s)
# - Type: (CAN)ControllerAreaNetwork or ...

B CAN 1000000

# Elementary cycle duration
# E TDuration

E 1560

# Scheduling algorithm
# A Tasks Messages
# - Tasks: (RM)RateMonotonic or (EDF)EarliestDeadlineFirst
# - Messages: (RM)RateMonotonic or (EDF)EarliestDeadlineFirst

A RM RM

# Entity declaration
#

# The time unit is microseconds
# The greatest allowed time value is aprox. 35 minutes (2147483647)

# Node declaration
# N Id
# - Id: 0<Id<255

N 1
N 2
N 3
N 4

# Task declaration
# T Id C T D PhReq Allocation NodeId
# - Id: 0<Id<255
# - T: Only valid for independent tasks
# - PhReq: Phase Requested
# - Allocation: (F)ixed o (A)llocable

T 1 00520 0 0 0 F 1

```

Fig. 10. Extract of an input file.

The input file has two main areas: system characterization and entity declaration. The system characterization area is used to configure the simulation selecting the network architecture, the elementary cycle and the scheduling algorithms used for task and message scheduling. The entity declaration is used to define the nodes, the tasks, the messages and the transactions.

After this the simulator determines the size of the macro cycle, i.e. the interval with a pattern, of tasks and messages, that will be repeated indefinitely, and displays the simulation options, see Fig. 11.

The second step is applying all constraints.

Property	Value
Task Scheduling Algorithm	RM
Message Scheduling Algorithm	RM
Bus Type	CAN
Bit Rate	1000000
Elementary Cycle (EC)	1560
Macro Cycle	6
StartUp Interval	6

Fig. 11. Simulation characteristics.

After this the simulator checks the restrictions imposed by the messages to the tasks, builds the data streams (Fig. 12) and makes the precedence analysis. The start-up interval is also determined, i.e. the time between the first EC trigger message and the EC trigger message corresponding to the highest relative phasing of all tasks and messages.

Below is depicted an example scenario with 6 tasks, running in 4 different nodes and that exchange 3 messages.

Property	Abs. Value	Rel. Value (EC)
Node	1	-
C	520	1
T	3120	2
D	3120	2
PhMin	0	0
PhReq	0	0

Fig. 12. Data streams.

The Fig. 12 shows the two data streams. For instance *task 5* sends *message 3* to *task 6*.

The task and message parameters can be changed and a new scenario file, reflecting the changes, can be saved.

The last step is to execute the scheduling.

For the scheduling the user has two alternatives. Either message scheduling is selected, or message and task scheduling is selected. If message scheduling is selected then only messages will be scheduled to the bus, which means that each task is considered to be executing in a different node, see Fig. 13. If on the contrary, message and task scheduling is selected then messages will be scheduled to the bus and tasks will be scheduled at their nodes of execution (according to the allocation node), see Fig. 14. The scheduling takes place at an EC basis, i.e. at every EC messages and tasks are scheduled.

The scheduling is executed until either the simulation terminates or until a task, or message, misses its deadline.

Another option for the simulation is to choose between the requested initial phases (Phreq) or the minimum initial phases (Phmin) that were determined during the analysis.

The area depicted in the Fig. 13 shows the transition from the startup cycle (that finishes in the EC 5) to the first macro cycle. In this figure we can check the bus utilization (or we can check the output file).

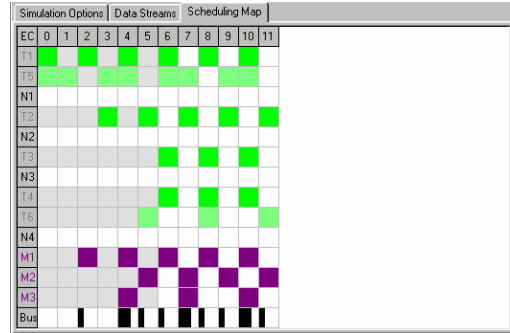


Fig. 13. Area of the message scheduling map.

The Fig. 14 also shows the nodes utilization.

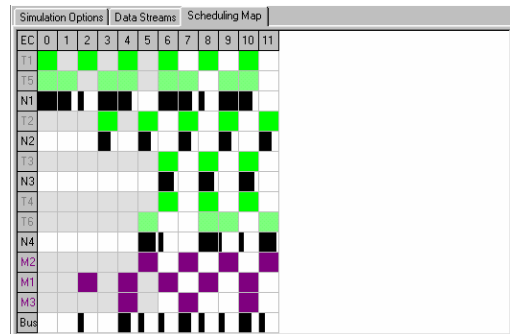


Fig. 14. Area of the task and message scheduling map.

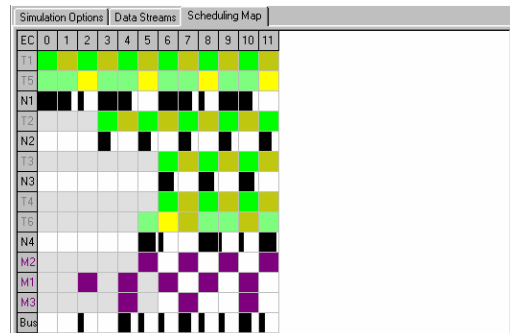


Fig. 15. Area of the task and message scheduling map with display of the execution windows.

An execution window represents the interval, in ECs, between the minimum and maximum allowed time for the task execution. Examples of execution windows can be seen in the Fig. 15.

Even for large macro cycles, the simulation is executed within a very short time frame. An example (shown in Fig. 16) with 10 nodes, 12 tasks and 8 messages, is organized in 4 data streams, and has a resultant startup time of 804 ECs and a macro cycle of 3510 ECs.

The task and message scheduling of this example stops at EC 3534 due to a missed deadline by message 4. An extract of the scheduling map is shown in the Fig. 17.

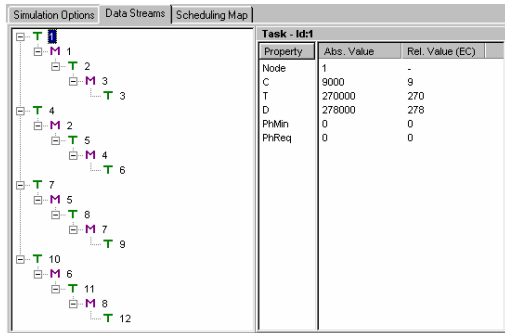


Fig. 16. Data streams.

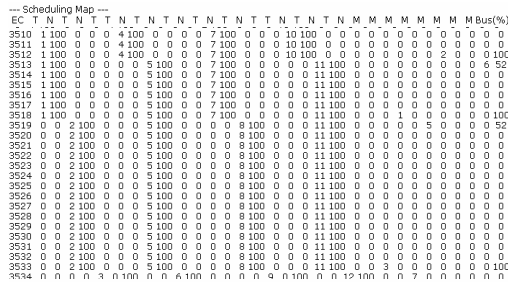


Fig. 17. Area of the output file.

5. CONCLUSIONS

The SimHol offers a suitable experimental platform for the simulation of distributed systems based on the time-triggered (FTT) paradigm. Using a simple interface it allows the simulation of various interconnection architectures, although currently only CAN is supported. Different scheduling algorithms can also be chosen. At this moment SIMHOL operates with Rate Monotonic and Earliest Deadline First. The user can start with a simple message scheduling in order to make a first essay to the system and then proceed to the full scheduling (tasks and messages) that takes into account the load of each node. The scheduling takes place at an EC basis closely resembling an actual system.

This simulator has validated the set of requirements previously derived, namely the data flow analysis and precedence requirements. The migration of the SimHol to a Java implementation is foreseeable due to a possible full integration with the execution environment. Other advantages are: improved portability and the use of standardized XML for the input/output files.

REFERENCES

Tindell, K., J. Clark (1994). *Holistic Schedulability Analysis for Distributed Hard Real-Time Systems*. In *Microprocessing & Microprogramming*, **40**, 117-134.

CiA DS 201-207 (1994). *CAN Application Layer for Industrial Applications*. CiA, CAN in Automation International Users and Manufacturers Group, 1994.

Tindell K., H. Hansson and J. Wellings (1994). *Analysing Real-Time Communication: Controller Area Network (CAN)*. Proc. of RTSS'94 (15th IEEE Real-Time Systems Symposium).

Henderson, W.D., Kendall, D., Robson, A.P. and Bradley, S.P. (1998), *Xrma: An holistic approach to performance prediction of distributed real-time CAN systems*. Proceedings of Can In Automation Conference, San Jose.

Almeida, L., J. Fonseca and P. Fonseca (1998). *Flexible Time-Triggered Communication on a Controller Area Network*. In *Proc. of Work-In-Progress Session of RTSS'98 (19th IEEE Real-Time Systems Symposium)*. Madrid, Spain.

Almeida, L., R. Pasadas and J.A. Fonseca (1999). *Using a planning scheduler to improve flexibility in real-time fieldbus networks*. IFAC Control Engineering Practice, **7**, 101-108.

Henderson, W.D., Kendall, D. and Robson, A.P. (2000). *Improving the Accuracy of Scheduling Analysis Applied to Distributed Systems*. RTS 2000.

Vector Informatik GmbH (2002). *CANalyzer: The Tool for CAN*. Product information at <http://www.vector-informatik.de>

Martins, E., P. Neves and J.A. Fonseca (2002). *Architecture of a Fieldbus Message Scheduler Coprocessor Based on the Planning Paradigm*. In *Microprocessors and Microsystems*, Vol. 26, Issue 3.

Almeida, L., P. Pedreiras and J.A. Fonseca (2002). *The FTT-CAN protocol: Why and How*. To appear in IEEE Transactions on Industrial Electronics.

Calha, M.J., J.A. Fonseca (2002). *Adapting FTT-CAN for the joint dispatching of tasks and messages*. Submitted to WFCS'2002 – IEEE Workshop on Factory Communication Systems.

Pedreiras, P., L. Almeida and P. Gai (2002). *The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency*. To appear in *Proceedings of the 14th Euromicro Conference on Real-Time Systems*. Viena, Austria.

Fonseca, J.A., J. Ferreira, M. Calha, P. Pedreiras and L. Almeida (2002). *Issues on Task Dispatching and Master Replication in FTT-CAN*. Africon'2002 – IEEE International Conference, George, South Africa