# Using Task Scheduling Algorithms for Fieldbus Traffic Scheduling

Carlos Cardeira*[1] Zoubir Mammeri[2]

[1]CRAN (CNRS URA 821) [2]CRIN (CNRS URA 262)
Ensem, 2 avenue de la Forêt de Haye
F-54516 Vandœuvre-les-Nancy Cedex
{cardeira, mammeri}@loria.fr

***Abstract***

In real-time systems the deadlines of each task must be met. A pre-run-time schedulability analysis becomes necessary to prove that the existing software and target hardware will meet the real-time application constraints. In a real-time distributed system, the messages transmitted through the network are also time constrained. However, some new problems arise when one applies existing task scheduling algorithms to schedule the network traffic. The main goal of this paper is to define the boundaries between these two domains of scheduling. After an introduction to fieldbuses and real-time systems, we present an equivalence between tasks and messages as well as between processors and networks, which are much different in practice but have strong similarities from the scheduling point of view. Finally, we analyse the new constraints introduced by the presence of Smart Transducers/Transmitters in fieldbus applications and we define the scheduling algorithms adapted for this type of applications.

***Keywords****: Task scheduling, message scheduling, real-time systems, communication systems, time constraints.*

* On leave from Instituto Superior Técnico, 1096 Lisboa Codex

## 1. Introduction

Real-time systems are those systems that are able to synchronise themselves with external events. Moreover, they must produce correct responses in a specified time, because a result produced out of time is useless even if it was correctly computed [Sta 91], [Hoo 91]. A result produced after its deadline is a wrong result and may lead to severe consequences.

A current practice to build a centralised real-time system is to choose a robust and last generation not cumbersome PLC from the huge variety available on the market (PLCs have benefited from the evolution of microprocessors and became much less voluminous, more robust and with increasing performances). All PLCs manufacturers propose a large variety of modular I/O cards which offer a sufficiently large number (8, 16, 24, etc.) of I/O ports per card. The user "just" need to connect each sensor/actuator to the corresponding I/O port and the software installed on the PLC will control the process as wished. Another advantage is that connections are more or less standard like the 4..20 mA current loop which is, in practice, the standard connection for the transmission of analog signals.

Moreover, most PLC manufacturers propose cards to interconnect their devices, allowing them to exchange variables, usually by a protocol which guarantees the consistency of data belonging to specific pages of each PLC memory. Each PLC may hence read/write this memory region and the protocol ensures that each change will be "automatically" updated on the other memories.

In conclusion, a common architecture of a "so called" distributed system that we may see in most plants or factories is composed by a set of controllers with a more or less large number of I/O cards connected to sensors/actuators by centralised point-to-point connections. Each controller is responsible for its own physical process (a motor, for instance) controlled by the installed software, or using dedicated hardware cards (PID cards, for instance) if the processing speed of the controller is not enough to ensure the process stability. The controllers communicate with each other by so called standard protocols which authorises a certain level of distributed control.

A question arises : why to change this such stable situation where everything seems to work fine ?

## 1.1. Motivations for a standard fieldbus

One of the main disadvantages of the previously presented situations is that wiring costs are one of the most important part of the distributed system cost. They require a lot of handwork and they are difficult to reconfigure. The total wiring length attains easily several miles and the cost to install or reconfigure them becomes important regarding the overall system cost.

This was the main motivation for the emergence of networks to directly interconnect sensors, actuators and controllers in a bus topology which is much more cost effective than point-to-point connections. These networks are called fieldbuses, as they are very closed to the physical process.

Besides this economical advantage, the introduction of a standard fieldbus has some more advantages [Gal 84], [Car 92], [Tho 93].

- Increased noise immunity.

  The transmitted information over a fieldbus is digital, not analog, and hence it has an increased immunity to the inevitable noise. Long point to point analog connections are always subject to undesirable interferences which affect the values read from sensors or sent to actuators. An analog value, after A/D conversion may be transmitted over the bus covering the necessary distances without any degradation of the digital information. If distances to be covered are so large that even digital information would be plenty of noise, it is always possible to use *repeaters* which actually regenerate the digital information.

- Easier reconfiguration

  As the equipments (sensors, actuators or controllers) are connected to the network by a single interface, it becomes easier to reconfigure the connections. If the central controller has to be moved away, we just need to reconnect it on any point of the fieldbus. This is a strong advantage comparing to a situation in which the controller was connected to all the sensors by point to point connections. In the former only one connection is to be moved, whereas in the later one has to reconfigure all the existing connections related to the moved controller.

- Accessibility to the information.

All the information sent over a fieldbus is broadcast all over the bus and may be read by any node connected to the fieldbus, independently of the node on which they where produced. Moreover, the protocol may guarantee that all nodes compute with the same values.

- Decreasing hardware costs.

The connection of a sensor/actuator to a fieldbus has a certain cost. Besides the sensing element one has to introduce at least a processor to perform the acquisition of the signal, some indispensable memory and the fieldbus interface circuits. This has a given cost and this is the main reason why we still have to wait till the emergence of common market cheap standard sensors/actuators. However, it is well known that hardware costs are always reducing and if a standard fieldbus would emerge, this additional costs would be hardly reduced in a near future.

- Transducers become smart !

Smart is maybe a bad chosen word but it is true that a sensor/actuator with some processing capabilities and a standard interface network may provide much more services than a classical one. These standard interfaced sensors / actuators are called "Smart Transducers / Transmitters". The main idea to make them "smart" was to take advantage of the processing capabilities of the probably under loaded processors of smart transducers/transmitters to process some tasks that usually were done by the controlling PLCs. Examples of these tasks are variable pre-processing tasks like, linearisation of the produced value, filtering and so on. If these tasks are distributed among the processors of smart transmitters/transducers instead of being all allocated to the central controlling device, this will actually free a non negligible part of the central controller load.

- Distribution of the system intelligence.

This advantage is a consequence of the generalisation of the previous one. If the central controller is free of all the pre-processing tasks, why not to do the same to the rest of the tasks ? These tasks may be distributed among all the processors of the smart transducers/ transmitters and there would be no more need of a centralised controller ! All the tasks become distributed among the several

transducers, as an actual distributed system. In such a system, tasks co-operate with each others (by means of message exchange) to achieve the overall goal, without any centralised hierarchical controller. Moreover, tasks may be allocated dynamically allowing the possibility of migration which strongly enhances the system fault-tolerance.

### 1.2. Fieldbus drawbacks

In spite of the above advantages, the introduction of fieldbuses brings up some disadvantages too. Let us see some of them :

- The fieldbus is a communication bottleneck !

Classical sensors are connected to the controllers by dedicated connections. The physical medium of a fieldbus is no more dedicated to each sensor but must be shared by all the connected devices. The controllers may no more read/send the information whenever they want. On the contrary, they must wait to have the access to the medium (this depends on the type of MAC protocol) to perform the desired input/output operations. The network traffic must hence be carefully scheduled, to guarantee that information will arrive to the consumers within a given time window to meet the timing constraints. Thus, we should not use MAC protocols in which all possible traffic generators may transmit messages at random times, provoking collisions or other sort of delays, if there is no guarantee that the time window to pass a message is bounded. In other words, protocols like CSMA/CD should not be used for fieldbuses.

- Lack of consistency

A fieldbus has the usual problems of distributed systems. As it has no central memory, protocols to guarantee that the copies of each variable have the same value must be provided [Bay 92]. As it has no single clock, clock synchronisation must be provided to ensure a consistent time stamping of variables and events occurrences. Nowadays a lot of algorithms exist ensuring a more or less accuracy of the synchronised clocks, depending mostly on the drift of each clock and the period between resynchronisation operations.

- Fault-Tolerance

A real-time system must be intrinsically fault tolerant (i.e., the system must continue to work correctly, even in the presence of local faults). Concentrating all the information on a single support may be a strong disadvantage when the medium becomes faulty. For instance, if the bus is accidentally cut, no more messages may be transmitted, conducting to a global fault of the system. This problem is solved by duplicating the physical supports.

From the above discussion, it seems that even if a fieldbus may pose new problems, the presented advantages do largely compensate the disadvantages.

But how can we guarantee that a fieldbus-based real-time system will meet the applications timing constraints ?

## 2. Schedulability analysis

To guarantee that the application timing constraints will always be met, one must do a pre-runtime schedulability analysis of the system. From the scheduling point of view, distributed real-time systems impose much more problems and constraints than centralised systems. Whereas in centralised monoprocessor based systems the main problem is to schedule tasks on a single processor, in distributed systems the problem is not only to decide "when" executing a task, but also to decide "where" should the task be executed. It is up to the global scheduler to decide which is the most suitable node (or nodes) to execute a given task and to give a warranty that it will be able to schedule all the tasks and network traffic, meeting all the application timing constraints [Cas 88].

There are some fundamental differences concerning message scheduling and task scheduling. Messages are somewhat different from tasks, nevertheless a lot of analogies do exist between task scheduling and message scheduling. They have many points in common, as shown by the following non-exhaustive list [Car 93], [Kop 86]:

1. Scheduling tasks on a processor means to serialise their execution on a processor. Similarly, it is possible to consider the message scheduling through a network as serialising the message transmission through the physical medium.

2. Two tasks can't be executed on a same processor at the same time, as well as two messages can't be transmitted through a physical support at the same time (otherwise a collision would occur).
3. A task has a computation time depending on the number of executed instructions. Similarly, a message has a transmission time depending on the number of bits to transmit.

This equivalence seems to be a departure point for a further analysis between message scheduling and task scheduling. The OSI reference model was introduced to clearly identify and isolate the communication problems that may exist between two stations. A complete equivalence between messages and tasks should hence be done taking into account all the seven layers of the OSI model. However, in practice, real-time communication systems do not need the services proposed by all the seven layers. Let us summarise each layer services and analyse their importance from the scheduling viewpoint.

The first layer, the physical layer, is responsible for bit transmission over the physical medium. It defines the features of the physical support and specifies electrical and mechanical specifications of the physical interface plugs and sockets. In other words, this layer is irrelevant from the scheduling viewpoint.

The Medium Access Control (MAC) sub layer defines the policy how nodes access to the physical medium. As the medium is a resource shared by a set of tasks, this layer is of extreme importance from the scheduling viewpoint.

The Logical Link Control (LLC) sub layer defines the procedure for opening and closing connections, flow control and error management. From the scheduling viewpoint this is equivalent to a fault tolerance technique similar to the "watch dog" technique used for tasks. The main difference is that in message communication the fault tolerance technique is almost inevitable (most network protocols support some sort of acknowledgement to prevent the inevitable noise over the medium), whilst for processors, fault tolerance techniques are not inevitable, they are used only for applications where dependability is a must.

The Network layer is responsible for routing messages among different networks. This protocols are used mainly in highly interconnected networks (like telecommunication networks). Fieldbus protocols do not include this layer because they are usually composed by a single segment.

The Transport layer is responsible for end-to-end flow control. It is more or less equivalent to the LLC sub layer, regarding that the LLC sub layer ensures reliable

communication among directly connected stations whilst the Transport layer does the same function but between end-to-end stations which may be connected passing through a huge number of intermediate stations and networks. Another function of the transport layer is to segment messages to keep them in a reasonable size. From the scheduling point of view this is equivalent to limit each task execution time in order to allow other tasks to execute in the meanwhile. This is a common practice for real-time operating systems called time sharing.

The session layer is responsible for maintaining associations between two or more application entities. In the case of loss of a communication connection, the session layer may provide check pointing to allow recovery from a known state. This is equivalent to some task error recovery procedures used in fault-tolerant systems.

The Presentation layer is responsible for solving differences in information representation between application entities and provides services like data transformation, formatting, structuring, encryption and compression. For instance, this layer is necessary to provide a standard representation for integers, reals and characters codes among heterogeneous computers [Slo 87]. From the scheduling viewpoint this layer is irrelevant.

The Application layer is responsible for protocols corresponding to end to end applications like electronic mail, file transfer protocols, etc. This layer is relevant for task scheduling, since user tasks are located at this layer.

From this analysis we conclude that the problem is too complex to be analysed for all the seven layers. From the scheduling viewpoint all the layers are more or less important but the one which plays the main part on message scheduling is the MAC sub layer. Hence, from now on, we'll only consider the influence of this sub layer, as this is the one responsible for the police for allocating the shared resource. The influence of the other layers is not considered, unless otherwise stated.

To test if the analogy between task scheduling and message scheduling holds, we present next the meaning of task scheduling algorithms features when applied to traffic scheduling.

We make a classification of existing work about scheduling algorithms regarding the following criteria: temporal aspects, priorities, preemption, end of tasks execution, precedence constraints, resource constrains, scope of the algorithm (global versus local), execution of the algorithm (on-line versus off-line), type of hardware (monoprocessor versus multiprocessor), optimisation criteria, fault

tolerance, and the resolution technique used by the algorithm [Car 93b] [Car 94]. Let us see these features one by one and see their meaning when applied for message scheduling.

### 2.1. Temporal aspects.

Existing task scheduling algorithms consider tasks with the following timing constraints [Xu 91], [Car 94b]:

- ready time (date before which it's impossible to start a task),
- deadline (date at which the execution of the task must be finished),
- execution time (time necessary for one task to be executed by the processor without preemption),
- maximum execution time (maximum interval between the start time and the finish time of the task)

The previous task features may be applied to messages without change. Messages may be transmitted after a given ready time and/or before a given deadline, as well as they have a transmission time (similar to task execution time) which is the time they take to be transmitted without interference through the medium. Critical messages must be transmitted within a maximum transmission time which is the maximum effective time to transmit the message considering all the interferences that other messages may provoke.

The activation of a task may be:

- at a fixed date (for instance "at 8:00 PM …"),
- periodic (for instance "every 20 ms … "),
- sporadic (for instance "when temperature is greater than 90° …").

It is the same for messages. Hence, the scheduling of traffic over a network must take into account fixed-date, periodic and sporadic messages.

### 2.2. Priorities

One way to schedule tasks is to assign priorities to them (statically or dynamically). Tasks are then scheduled obeying to the previously established priorities. The way how messages may be scheduled through a network may also be

conducted by dynamic or static priorities. In fact, this is already done by some MAC protocols like FFDI [Ans 87], [Ros 89], token ring [Iso 86], token bus [Iso 87], etc.

Priorities assigned to tasks are often computed by the scheduling algorithm, and seldom defined by the user. In message scheduling, priorities assigned to messages are not defined at the MAC level, (i.e., the message schedulers at MAC level do not introduce priorities, they only have to follow the ones defined by the upper level).

### 2.3. Preemption

Preemption is the operation of interrupting a task to execute another one(s) and resuming the execution of the first task some time afterwards. Preemptive scheduling has improved performance (it enhances the processor utilisation) than non-preemptive scheduling. However, if tasks have resource constraints, preemption may lead to deadlocks or priority inversions [Raj 91].

At first sight, it seems that message scheduling is non-preemptive. In fact, a message, once being transmitted may not be preempted by another one and be resumed after. This is true for most existing protocols. Nevertheless, even if we may conclude that message scheduling is non-preemptive there are some exceptions. For instance, the Register Insertion protocol [Rea 75], [Haf 75], is preemptive. because a message being transmitted over a ring may be preempted by a message from another station and be resumed thereafter.

However, preemptive algorithms, even if they are not suited to message scheduling for most existing protocols, may still be used if the transmission times become insignificant regarding the messages deadlines. This is often true for fieldbus traffic which is composed by a large quantity of short messages. For instance, a message containing a one bit information (a simple sensor corresponding to a switch indicating open or closed), even if the frame to code the message adds some more 31 bits, will have a 32 bit message length which corresponds to 32 microseconds in a common 1Mbps network, whilst message deadlines are measured in milliseconds (as it is the case in FIP). When the execution time compared to the deadline goes to zero, the difference between preemptive and non-preemptive tends to disappear.

### 2.4. End of tasks execution

Most existing task scheduling algorithms consider that a task has to be executed till the end which sounds logic. But there are some tasks which results are produced by successive iterations and may have a big (or infinite) execution time. However, after some iterations the solutions is quite good and there is no need for continuing to execute the task. Examples of this kind of tasks are, for instance, the classical example of a task calculating the exact • value or, more realistically, a pattern recognition task which result is more and more accurate when its execution time grows. These tasks are called incremental tasks. Algorithms to schedule this type of tasks minimise the overall error produced by an application composed by a set of incremental tasks.

Like for the preemption this type of algorithms is not suitable for message scheduling because a message is always bounded in length, and an "incremental" message has no sense.

### 2.5. Precedence constraints

Till now we only dealt with constraints of a task, considered isolated from the others. However, in a distributed system tasks co-operate each other to achieve an overall goal. By this we mean that tasks aren't independent, they must be executed in a defined order, they share critical resources, etc., i.e. they have precedence, resource and allocation constraints. Is it the same for messages ?

Two tasks have precedence constraints if the execution of one task must be achieved before starting another one. Messages may have precedence constraints too, as it is for instance, the case of connection request messages which must be scheduled before scheduling the acknowledge messages.

### 2.6. Resource constraints

To be executed, tasks may require a set of resources (processors, memory, I/O ports, etc.). Similarly, to be transmitted, messages require a set of resources (physical medium, memory, etc.).

Resource constraints hardly complicate the scheduling problem, leading to undesirable situations like deadlocks [Raj 91]. In the case of tasks, a deadlock arises,

for instance, when a task uses a resource R1 and requires a resource R2 which is being used by another task requiring the resource R1: the two tasks are blocking each other.

In a multiple-connection network, the same situation may arise. For instance, if one message has to be broadcast over two channels at the same time, the sender may fall in a deadlock if another sender tries to do the same operation, but reserving the channels in an opposite order.

A special class of resource constraints is known as allocation constraints; tasks may not be placed on any processor of the network because of several reasons, like micro code compatibility. Similarly, sending messages may require the use of a specific network (for instance, for confidentiality reasons).

### 2.7. Monoprocessor versus multiprocessor

A node may be based on a monoprocessor or a multiprocessor architecture, and a task located on this node may be executed on any processor. Similarly, a network may be composed of a single physical medium or several redundant ones and a message may be transmitted over any physical medium of this network.

As we made an equivalence between a physical medium and a processor, multiprocessor task scheduling corresponds to message scheduling over networks with redundant physical medium and monoprocessor task scheduling corresponds to message scheduling over single medium networks. In the next section we'll see an extension of this analogy for highly interconnected networks where, to transmit a message, several paths are allowed.

### 2.8. Local scheduling versus global scheduling.

A real-time system may be composed of one node or several interconnected nodes. Similarly a communication network may be composed of one or several interconnected elementary networks (for example, a token bus network connected to a token ring network by a router). In a communication system composed of several elementary networks, one must distinguish internetworking devices (bridges, gateways, routers) from other nodes.

Task scheduling algorithms may be local or global. Local scheduling concerns scheduling on just one node[1]. When nodes communicate with each other, scheduling becomes global and schedulers must answer to questions like where to execute a given task and when do it (note that local schedulers only answer to the last question).

In the case of an elementary network, one MAC protocol is responsible for the local message scheduling over this network. If several elementary networks are interconnected, then each local MAC protocol schedules its own traffic, and it's up to the internetworking devices (which support several protocols) to cooperate to schedule messages across different networks.

In conclusion, we may assimilate local task scheduling to message scheduling over an elementary network, and global task scheduling to message scheduling over several interconnected networks.

### 2.9. On-line versus off-line.

An on-line scheduling algorithm is executed whenever a task execution request is recorded. An off-line algorithm defines the schedule before runtime, using the information specifying the future behaviour of the tasks. Off-line scheduling is well suited to periodic task scheduling as well as on-line scheduling algorithms are more suited to handling aperiodic requests. Message scheduling is defined off-line for some network protocols (for instance, the cyclic traffic of FIP [Tho 93]) or on-line for others (for instance token bus [Iso 87]).

### 2.10. Optimisation criteria.

The quality of the result of a scheduling algorithm is an important feature. There are algorithms which guarantee the constraints satisfaction and an optimal proof (an algorithm is optimal in the sense that if the algorithm fails no other algorithm would be able to provided a schedule satisfying the constraints). Moreover, some algorithms may still optimise some criteria. Of course, some of these criteria may be important for message scheduling too.

[1]A node means any device connected to the network (a PLC, a minicomputer, a smart transducer/transmitter, etc.).

Let us see some of the most used criteria and translate their meaning when talking about message scheduling.

### *2.10.1. Minimisation of the schedule length.*

Some task scheduling algorithms optimise the schedule length, i. e., the elapsed time between the activation of the first executed task and the end of the last task. The main idea is to use the processor during a minimal time period. This is an interesting criterion to message scheduling too, as it may reduce the delay to transmit a set of messages.

### *2.10.2. Load Sharing.*

An optimisation criterion of global scheduling is the load sharing among the nodes. In multiple-network message scheduling, load sharing means to equilibrate the traffic among the different existing networks,

### *2.10.3. Minimisation of the communication load.*

How may message scheduling benefit from task scheduling algorithms which minimise the messages among the nodes ? Let us resume the analogy made above : "Global scheduling for messages means the schedule of messages among different networks". In this case what we are minimising is no more than the number of messages crossing different networks. As messages passing through different networks will be naturally delayed by the bridges or gateways it becomes natural that message "placing" should be done minimising the messages that cross different networks.

### *2.10.4. Minimisation of the number of tasks that do not meet the timing constraints.*

If the scheduler cannot meet the timing constraints then, one the criterion to optimise may be the number of tasks that do not meet their timing constraints. It is the same for messages, in the sense that if the network bandwidth isn't enough to transmit all the time constrained messages. Then, we may be interested in a algorithm that maximises the number of successfully transmitted messages.

### 2.11. Fault tolerance

In a distributed system, fault tolerance may be obtained by the execution of several copies of the same task in different processors. The result is chosen by vote or consensus. Similarly several clones of the same messages may be sent from source to the destination using different networks or alternative mediums of the same network.

### 2.12. Resolution techniques for finding a solution.

Most of existing techniques used to elaborate exact or approximate solutions for task scheduling problems may still be used by message scheduling algorithms. Among existing techniques, we may cite: graph theoretical [Sto 77], [She 85], heuristics [Zha 87], [Ram 89], queuing theory [Shi 89], simulated annealing [Aar 88], [Tin 92], genetic algorithms [Hol 85], [Tal 91], fuzzy logic [Zad 65], [Ish 92] and neural networks [Hop 85], [Car 94b].

## 3. Brief presentation of existing algorithms

Most work begun by Liu and Leyland [Liu 73] which derived conditions for scheduling periodic tasks in a monoprocessor environment when preemption is allowed. Their results were extended to handle deadlines [Der 74] and precedence constraints [Che 90]. Chung [Chu 90] and Dean [Dea 88] dealt with scheduling of imprecise results tasks.

Mok and Dertouzos [Der 89] proposed the Least Laxity algorithm for scheduling in a multiprocessor environment.

Sprunt [Spr 89] and Chetto [Che 90] dealt with handling non-periodical tasks requests over a set of periodic tasks already scheduled.

Non-preemptive scheduling is treated by [Jef 91]. Resource constraints are handled by [Gar 75], [Raj 91], [Tin 92].

Concerning global scheduling, solutions exist both for static task placement (sometimes allowing fault tolerance constraints) [Ma 82], [Ban 83] [Tin 92] and for dynamic task allocation (when task migration is allowed) [Sta 85], [Shi 89].

Finally, some tutorials exist concerning the above algorithms [Car 94], [Xu 91], [Cas 88], [Mun 91].

A complete presentation of all the algorithms would be out of the scope of this paper. We prefer summarising (see table 1) those which are suited to message scheduling, either because they may be applied directly or because they may be applied with minor modifications. We decided to include preemptive algorithms, even if they are not suited to message scheduling for most existing protocols because of the reasons already explained in section 2.

The table 1 (which is a subset of the tables presented in [Car 94]) is organised as follows: a • in a cell means that the corresponding algorithm handles the corresponding feature; the algorithms are classified in function of the previously presented criteria; each algorithm is referenced by the its most important bibliographical reference.

| Classification Criteria | | | | Algorithms | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Liu 73 | Der 74 | Che 90 | Ma 82 | Ban 83 | Sta 85 | Zha 87 | Spr 89 | Der 89 | Shi 89 | Ram 89 | Raj 91 | Tal 91 | Jef 91 | Tin 92 | Yu 92 | Ish 92 |
| Task (message) features | Timing Constraints | Deadline | | • | • | | | • | • | • | • | • | • | • | • | | • | • | • | • |
| | | Computation time | | • | • | • | • | • | • | • | • | • | • | • | • | • | • | • | • | • |
| | | Periodic execution | | • | • | | | • | • | | • | • | | | • | | • | | • | |
| | | Sporadic execution | | | • | | | | • | | • | • | | | | | | | • | |
| | Priorities | Static | | • | | | | | | | | | | | | | | | • | |
| | | Dynamic | | | • | | | | | | • | • | | • | • | | • | | | |
| | Preemption | Preemptive | | • | • | | | • | • | • | • | • | | | • | | | • | • | • |
| | | Non preemptive | | | | • | • | | | • | | | | • | | • | • | | | |
| | Intertask relations | Independant tasks (messages) | | • | • | | | • | • | | • | • | • | | | • | • | | • | • |
| | | Resources | Mutual exclusion | | | | | | | • | | | | • | • | | | | | • |
| | | | N access points | | | | • | • | | • | | | | | | | | • | | |
| | | Precedence | | | | • | • | | | | | | | | | | | • | | |
| Distri-bution | Local scheduling | | | • | • | | | | | • | • | • | | • | • | | • | | • | • |
| | Global scheduling | Local strategy | Yes | | | | | • | • | | | | • | | | | | • | | |
| | | | No | | | • | • | | • | | | | | | | • | | | | | |
| | | Placement | Static | | | • | • | • | | | | | | | | • | | • | | |
| | | | Dynamic | | | | | | • | | | | • | | | | | | | |
| Algorithm execution | On line | | | • | • | | | | • | • | • | • | • | • | • | | • | | • | |
| | Off line | | | | | • | • | • | | | | | | | • | | • | | • | |
| Hardware features | Number of processors | Monoprocessor | | • | • | | • | | • | | • | | • | | • | | • | • | • | |
| | | Biprocessor | | | | • | | | | | | | | | • | | | | | |
| | | Multiprocessor | | | | • | | • | | • | | • | | • | • | • | | | | • |
| | Processors speed | Same speed | | | | • | | | • | | | • | | • | • | • | | | | |
| | | Different speeds | | | | | | • | | • | | | • | | | | | | | • |
| Solution quality | Contraints satisfation | Yes | Optimal algorithm | • | • | | | | | | | | | | | | | | • | |
| | | | Non optimal | | | | | | • | • | • | • | • | • | • | | • | • | | • |
| | | No | | | | • | • | • | | | | | | | | • | | | | |
| | Schedulability test | | | • | • | | | | | | | • | | | • | | • | | | |
| Optimised criterion | Number of faulty tasks | Optimal solution | | | | | | | | | | | | | | | | | | |
| | | Non optimal solution | | | | | | | • | | • | | | | | | | | | |
| | Communic. Load | Optimal solution | | | | • | | | | | | | | | | | | | | |
| | | Non optimal solution | | | | | • | | | | | | | | | • | | | | |
| | Load sharing | Optimal solution | | | | | | | | | | | | | | | | | | |
| | | Non optimal solution | | | | | • | • | | | | | • | | | • | | | | |
| Fault Tolerance | | | | | | | | • | | | | | | | | | | • | | |
| Technique for finding the solution | Simulayed Annealing | | | | | | | | | | | | | | | | | • | | |
| | Genetic Algorithmes | | | | | | | | | | | | | | | • | | | | |
| | Fuzzy logic | | | | | | | | | | | | | | | | | | | • |
| | Graph theory | | | | | • | | | | | | | | | | | | | | |
| | Queuing theory | | | | | | | | | | • | | | | | | | | | |
| | Heuristics | | | | | | | | | • | | | | • | | | | | | |

**Figure 1.** *Task scheduling algorithms relevant to message scheduling.*

## 4. Applying existing work to the FIP fieldbus traffic scheduling

The consequences of the schedulability analysis on fieldbus architecture is that we think that for a fieldbus, some of the traffic must be configured off-line as it is done for some tasks in real-time systems. These static traffic and tasks must be pre-run-time scheduled [She 90] [Xu 91] to ensure that hard real-time constraints will be met.

In a real-time system, both a variable and the time at which it was produced, should be considered as an atomic entity and so, should always be broadcast

together. This timing information is analysed by consuming tasks to verify their temporal validity [Lor 94].

Fip protocols [Tho 93], support some of the above detailed characteristics. In particular they provide:

- The network traffic in a FIP fieldbus is pre-run-time analysed and configured in order to prevent run-time overloads or collisions.
- Flags are associated to transmitted variables qualifying their temporal validity. These flags indicate if the variable was produced, broadcast and consumed respecting its timing constraints.

Which algorithms may we use to schedule the traffic on the FIP fieldbus ? First, FIP fieldbus traffic is absolutely non-preemptive. Hence, for the static configuration of a fieldbus traffic only algorithms for non-preemptive scheduling can be used. Moreover, in FIP, a bus arbitrator schedules the network traffic according to its sampling table. All variables belonging to the static part of its table are periodic. So we are concerned with algorithms for scheduling periodic non-preemptive tasks on a single processor, which is a known subject. Moreover, as we said, FIP messages have transmission times which are much lower than deadlines which allows the use of strategies used by preemptive algorithms.

Handling non-periodical requests for message transmission is a rather difficult aspect because most algorithms for non-periodical request handling [Spr 89] consider some sort of periodic servers to deal with requests. However, as a fieldbus is a distributed system, the aperiodic requests do not appear necessarily on the bus arbitrator station. Algorithms from [Spr 89] can only be applied to centralised systems. The solution taken by the FIP protocol is to reserve some bits on periodic messages to communicate the aperiodic request to the network scheduler.

## 5. Application to Smart Transducers/Transmitters.

In the previous sections we analysed the use of global scheduling algorithms to fieldbus-based applications in particular to schedule the network traffic. But, as we said in the introduction, the nodes connected to the fieldbus are the so called Smart Transducers/Transmitters, which, from the scheduling viewpoint are somewhat

different from classical devices connected to a network. What new constraints do they lead to and which are the scheduling algorithms most suited to handle them ?

When compared to a classical device, the processing entities of the smart transducer/transmitter are limited in memory size which hardly limits the freedom of global scheduling algorithms. Next, some functions are basic functions (those directly involved with the sensing element) are fixed (in ROM) and not allowed to migrate, which limits the application of dynamic algorithms for task migration.

In conclusion, from the scheduling point of view (problem of task placement), smart transducers/transmitters impose more allocation constraints than classical devices do, but these placement constraints can be handled by current practices for static allocation [Ban 83]. Hence, the class of algorithms most suited to fieldbus task allocation is the class of global static allocation algorithms [Ban 83], [Tin 92].

These additional task allocation constraints induced by smart transducer/transmitters reduce the possibility of finding an optimal solution, but they may increase the performance of the algorithms [Ma 82].

## 6. An example of traffic scheduling : the FIP configuration tool

The FIP configurator from the FIPTOOLBOX package [Let 92] provides a schedulability analysis of the network traffic in order to ensure that no overloads appear during run-time. Each variable is defined by its period and its computation time. In fact the computation time is calculated as a function (no-linear) of the type of variable (integer, long, float, double).

FIPTOOLBOX creates a sampling table used by the bus arbitrator to schedule the network traffic. The length of this table is proportional to the least common multiple of the periods of the periodic variables to broadcast. Moreover, it generates all the declarations of the variables to be linked with the user programs (TurboC, µsoftC) and the linked program can be executed in a personal computer. FIPSTACK allows the extension to the micro-controllers Intel (80C31, 80C51, 80C196), Nec (78C10) and Motorola (series 68xxx, bus G96, SCSI and the OS9 system).

From the scheduling point of view, FIPTOOLBOX generates a static stable according to a non-preemptive version of the Rate Monotonic Algorithm [Liu 73], and then shifts some variables to make the fieldbus load as constant as possible (which is nearly equivalent to the load sharing optimisation criteria).

In conclusion, the FIP configuration tool is a first solution to deal with the message scheduling problem. It only configures the traffic of the network but the same principle can be extended to the task scheduling on the nodes of the distributed system. In this case it would generate, not only the sample table for the bus arbitrator, but also the task activation tables to all the nodes, satisfying all the precedence, resources and timing constraints specified.

The use of on-line algorithms in the bus arbitrator would have the advantage of doing without the bus arbitrator sampling table. The existence of the bus arbitrator table limits the possible variable periods and hardly complicates on-line reconfiguration. With an on-line algorithm on the bus arbitrator, variables would no more need to avoid relative prime periods and reconfiguration becomes straightforward. The inconvenient is that the algorithm execution time is eventually greater than the operation of reading the sampling table. In practice, this inconvenient would be translated by an increased bus arbitrator turnaround time.

## 7. Conclusions

As for the objectives initially presented, we've elaborated an analogy to pass from task scheduling algorithms to message scheduling to take advantage of existing work and not reinvent the wheel.

We may summarise the analogies between task and message scheduling as follows:

- Monoprocessor task scheduling becomes analogous to message scheduling over a single physical support.
- Multiprocessor task scheduling becomes analogous to message scheduling over alternate (or maybe redundant) physical supports.
- Global task scheduling becomes message scheduling over different networks.

In function of the equivalence made and the criteria that are suitable to message scheduling we defined the subset of task scheduling algorithms relevant for traffic scheduling.

Regarding this survey, our conclusions are the following:

- One criterion that is not taken into account by existing scheduling algorithms is to minimise the jitter of the transmitted variables, which is an important criterion for some control applications.
- The application of on-line monoprocessor non-preemptive priority based scheduling algorithms to the network scheduling in a FIP fieldbus, can increase the performances of the FIP configurator by the following reasons:
  1. It overpasses the need of the sampling table on the bus arbitrator. When this table is too large it is impossible to fit it on the limited bus arbitrator memory.
  2. Reconfiguration would be much easier (over passing the need of complex procedures for on-line sampling table reconfigurations).
- The introduction of on-line scheduling algorithms on the bus arbitrator station brings up some disadvantages too. The challenge is to bring the algorithm execution time to acceptable values (comparing to just reading the table) to reduce the eventually increased bus arbitrator turnaround time.
- We note the need for non-preemptive optimal scheduling algorithms for periodic scheduling. The analogy made to use preemptive algorithms to non preemptive scheduling does not hold if message transmission times become important. Unfortunately, non-preemptive scheduling over a monoprocessor with arbitrary deadlines and start-up times is an NP-Complete problem.

## References


[Aar 88] Aarts, E. and Horst, J., *Simulated Annealing and Boltzmann Machines*, Wiley-Interscience, New York, 1988.

[Ans 87] Ansi, *"FDDI Token Ring Media Access Control,"* ANSI X3T39, 1987.

[Ban 83] Bannister, J. and Trivedi, K., "Task Allocation in Fault-Tolerant Distributed Systems," *Acta Informatica*, 20, 1983, pp. 261-81.

[Bar 92] Baruah, S.,. Koren, G., Mao, D., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D., and Wang, E., "On the Competitiveness of On-Line Real-Time Task Scheduling," *Journal of Real-Time Systems,* 4, 1992, pp. 125-144.

[Bay 92] Bayard, M. and Staroswiecki M., "Coherence of Distributed Applications Under Critical Time Constraints," in Proceedings *of the IFAC/IFIP International Workshop on Real-Time Programming*, Bruges, June 1992, pp. 155-60.

[Car 92] Cardeira, C., Mammeri, Z., and Thomesse, J.P., "Scheduling in Fieldbus Based Real-Time Systems," in W. Halang and A. Stoyenko eds. Real-Time Computing, NATO ASI Series, Springer Verlag, 1992.

[Car 93b] Cardeira, C., Mammeri, Z., and Thomesse, J.P., *"Constraint Specification for Task Scheduling in Real-Time Systems,"* in B. Mayoh, J. Penjam and E. Tyugu (Eds.) Constraint Programming, Tech. Rep. CS 57/93, Institute of Cybernetics, Estonian Academy of Sciences, Tallinn, 1993, pp. 183-197.

[Car 93] Cardeira, C., Siebert, M., and Thomesse, J.P., "Scheduling on Fieldbus with Smart Transducers/Transmitters," in C. Eugène ed., International Symposium on Intelligent Instrumentation for Remote and On-Site Measurements, IBRA-BIRA, Brussels, Mai 1993, pp. 259-265.

[Car 94b] Cardeira, C. and Mammeri, Z., "Neural Networks for Satisfying Real-Time Task Constraints," in *Proceedings of SPRANN'94 IMACS Symposium on Signal Processing, Robotics And Neural Networks*, Lille, April 1994.

[Car 94] Cardeira, C., Mammeri, Z., "Ordonnancement de tâches dans les systèmes temps-réel et répartis," Tech. Rep. CRIN n. 93-R-097, submitted (Mai 93), to *Automatique, Productique et Informatique Industrielle,* revised (December 1993)

[Cas 88] Casavant, T. and Kuhl, J., "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Transactions on Software Engineering*, 14 (2), February 1988, pp. 141-54.

[Che 90] Chetto, H. and Bouchentouf, T., "Dynamic scheduling of real-time tasks under precedence constraints," *Real-Time Systems*, 2, March 1990, pp. 181-94.

[Chu 90] Chung, J., Liu, J., and Lin, K., "Scheduling Periodic Jobs that Allow Imprecise Results," *IEEE Transactions on Computers*, 39 (9), Septembre 1990, pp. 1156-74.

[Dea 88] Dean, T. and Boddy, M., "An Analysis of Time-Dependent Planning," in *Proceedings of the 7th National (USA) Conference on Artificial Intelligence,* 1988, pp. 49-54.

[Der 74] Dertouzos, M., "Control Robotics: the procedural control of physical processes," in *Proc. IFIP Congress,* 1974, pp. 807-813.

[Der 89] Dertouzos, M. and Mok, A., "Multiprocessor On-line Scheduling of Hard Real-Time Systems," *IEEE Transactions on Software Engineering*, 15 (12), December 1989, pp. 1497-1506.

[Gal 84] Galara, D. and Thomesse J.P. "Proposition d'un système de transmission série multiplexée pour les échanges d'informations entre des capteurs, des accionneurs et des automates réflexes," *Groupe de Reflexion FIP,* Ministère de L'Industrie et de la Recherche, 1984.

[Gar 75] Garey, M. and Johnson, D., "Complexity Results for Multiprocessor Scheduling under Resource Constraints," *SIAM (Society for Industrial and Applied Mathematics) Journal of Computing*, no. 4, 1975, pp. 397-411.

[Haf 75] Hafner, E., Nenadal, Z. and Tschanz, M., "Integrated local communications - principles and realization," *Hasler Review,* 8 (2), 1975, pp. 34-43.

[Hol 85] Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[Hoo 91] Hoogeboon and Halang, "The Concept of Time in Software Engineering for Real Time Systems," in *3rd International Conference on Software Engineering for Real Time Systems,* IEEE, 1991, pp. 156-163.

[Hop 85] Hopfield, J., "Neural computation of decisions in optimization problems," *Biological Cybernetics,* 52, 1985, pp. 141-152.

[Ish 92] Ishii, H., Tada, M. and Masuda, T., "Two Scheduling Problems with Fuzzy Due-Dates," *Fuzzy Sets and Systems,* 46, 1992, pp. 339-347.

[Iso 86] ISO, International Standards Organization,"*Local Area Netorks - Token Ring Access Method and Physical Layer Specifications,"* DIS 8802.5, 1986

[Iso 87] ISO, International Standards Organization,"*Local Area Netorks - Token BUS Access Method and Physical Layer Specifications,"* DIS 8802.4, 1987

[Jef 91] Jeffay, K., Stanat, D., and Martel, C., "On non-preemptive scheduling of periodic and sporadic tasks," in *IEEE Real-Time Systems Symposium,* San Antonio, December 1991, pp. 129-139.

[Kop 86] Kopetz, H., "Scheduling in Distributed Real Time Systems," Tech. Rep. Mars nr. 1/86, Austria, January 1986.

[Kor 92] Koren, G. and Shasha D., *"D-Over: an Optimal On-Line Scheduling Algorithm for Overloaded Real-Time Systems,"* Technical Report INRIA n. 138, February 1992.

[Let 92] Leterrier, P. and Valentin, T., *FipToolbox 3.0 - Reference Manual*, Centre de Competence FIP, 3 rue de la Salpêtrière, 54000 Nancy, France, April 1992.

[Liu 73] Liu, C. and Layland, J., "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, 20 (1), January 1973, pp. 46-61.

[Lor 94] Lorenz, P., Mammeri, Z. and Thomesse, J.-P., "A state-machine for temporal qualification of time-critical communication," in *26th IEEE Southeastern Symposium on System Theory,"* Ohio, Mars 1994.

[Ma 82] Ma, P., Lee, E., and Tsuchiya, M., "A Task Allocation Model for Distributed Computing Systems," *IEEE Transactions on Computers*, 31 (1), January 1982, pp. 41-47.

[Mun 91] Muntean, T. and Talbi, E., "Méthodes de placement statique des processus sur architectures parallèles," *Tecnique et Science Informatiques*, 10 (5), Octobre 1991, pp. 355-419.

[Raj 91] Rajkumar, R., "*Synchronization In Real-Time Systems: A Priority Inheritance Approach*," The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Boston/Dordrech/London, 1991.

[Ram 89] Ramamrithan, K, Stankovic J. and Zhao W., "Distributing Scheduling of Tasks with Deadlines and Ressource Requirements," *IEEE Transactions on Computers*, 38 (8), August 1989, pp. 1110-1123.

[Rea 75] Reames, C., and Liu, M., "A loop network for simultaneous transmission of variable-length messages," in *Proceedings os the 2nd annual symposium on computer architecture,* Houston, Texas, January 1975.

[Ros 89] Ross, F.., "An overview of FDDI," *IEEE Journal on Selected Areas in Communications*, 7 (7), September 1989, pp. 1043-1051.

[She 85] Shen, C., Tsai, W., "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minmax Criterion," *IEEE Transactions on Computers,* 34 (3), March 1985, pp. 197-203.

[She 90] Shepard, T. and Gagné, J., "A Model of the F18 Mission Computer Software for Pre-Run-Time Scheduling," in *10th International Conference on Distributed Computing Systems*, May 1990, pp. 62-69.

[Shi 89] Shin, K. and Chang, Y., "Load Sharing in Distributed Real-Time Systems with State Change Broadcasts," *IEEE Transactions on Computers*, 38 (8), August 1989, pp. 1124-1142.

[Shi 89] Shin, K. and Chang, Y., "Load Sharing in Distributed Real-Time Systems with State Change Broadcasts," *IEEE Transactions on Computers*, 38 (8), August 1989, pp. 1124-1142.

[Slo 87] Sloman, M. and Kramer, J., "Distributed Systems and Computer Networks," in C. A. R. Hoare Series Editor, Prentice-Hall National Series in Computer Science, 1987.

[Spr 89] Sprunt, B., Sha, L., and Lehocsky, J., "Aperiodic Task Scheduling for Hard Real-Time Systems," *Real-Time Systems*, 1, January 1989, pp. 27-60.

[Sta 85] Stankovic, J., Ramamritham, K., and Cheng, S., "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems," *IEEE Transactions on Computers*, 34 (12), December 1985, pp. 1130-43.

[Sta 91] Stankovic, J. and Ramamritham, K., "The Spring kernel: a new paradigm for real-time systems," *IEEE Software*, 8 (3), May 1991, pp. 62-72.

[Sto 77] Stone, H, "Multiprocessor Scheduling with the aid of network flow algorithms," *IEEE Transactions on Software Engeneering*, 3(2) 1977, pp. 85-93.

[Tal 91] Talbi, E. and Bassiere, P., "A parallel genetic algorithm for the graph partitioning problem," in *ACM International Conference on Supercomputing*, Cologne, Germany, June 1991.

[Tho 93] Thomesse, J.-P., "Le réseau de terrain FIP," *Réseaux et Informatique Répartie,"* 3 (3), Hermes, 1993, pp. 287-321.

[Tin 92] Tindell, K., Burns, A., and Wellings, A., "Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy," *The Journal of Real-Time Systems*, 4, 1992, pp. 145-65.

[Xu 91] Xu, J. and Parnas, D., "On Satisfying Timing Constraints in Hard-Real-Time Systems," *Software Engeneering Notes*, 16 (5) December 1991, pp. 132-146,.

[Yu 92] Yu, A., and Lin, K., "A Scheduling Algorithm for Replicated Real-Time Tasks," *IPCC'92,* IEEE, 1992, pp. 395-402.

[Zad 65] Zadeh, L., "Fuzzy Sets," *Information and Control,* 8, August 1965, pp. 338-53.

[Zha 87] Zhao, W., Ramamrithan, K., and Stankovic, J., "Scheduling Tasks with Ressource Requirements in Hard Real-Time Systems," *IEEE Transactions on Computers*, 13 (5), May 1987, pp. 564-577.