EDF message scheduling on a CAN network[1]

F. Rodríguez, J.C. Campelo
Dept. of Computer Engineering, Technical University of Valencia (SPAIN)
{prodrig, jcampelo}@disca.upv.es

In this paper, Earliest Deadline First (EDF) scheduling algorithm has been translated to a CAN network by the use of a slightly modified CAN controller, called EDF controller. With this controller, the network can be modelled as a single, prioritised queue of messages. The messages use their time-to-deadline as their priority level. Using EDF scheduling on the network guarantees message transmission times. This can be also used in conjunction to the task scheduling algorithms in the CPU nodes to obtain a global scheduling policy related to the whole system. The information needed to apply a dynamic end-to-end scheduling algorithm can be automatically delivered into CAN messages when using EDF controllers.

Introduction

CAN networks ([1, 2]) are becoming more and more widespread used in the industrial environment. The main characteristics of this environment are:

- Short messages are sent across the network. The information carried is usually the data collected from distributed sensors, reference values to control algorithms, actuators control signals and so on.

- Low bandwidth needs compared to general-purpose networks.

- All processes in the system must meet hard real time constraints. This imposes time constraints on the communication network, so the system schedule can be guaranteed.

- Low cost is a main factor in the protocol/element selection. One of the most important cost factors in these networks is cabling cost.

CAN restricts the maximum data length to 8 bytes, and uses two wires to transmit balanced data with data rates up to 1 Mbps. In this network, 1s are called *recessive* bits and can be overwritten by 0s, called *dominant* bits. The bus can be seen as a large, wired-AND gate. If, at any given time, a node transmit a recessive bit and a second node transmit a dominant bit, the resultant bit seen on the network is a dominant bit. This is the base for contention resolution.

When several nodes are connected to a bus, some access mechanism must be used. In a CAN network, any node listen the channel until it becomes idle; then, if there is a pending message, transmission begins. This can cause that several nodes compete for the bus transmitting different messages at the same time.

However, collision resolution does not destroy the message sent. Any node transmitting a message still listen the bus to compare the bits being transmitted with those listened. At any time while transmitting the identifier filed, if a recessive bit is sent and a dominant bit is listened, means that the node has lost arbitration and must end transmission immediately.

As CAN identifiers must be unique across the network, it is impossible that two nodes try to transmit data messages with the same identifier. At the end of the first field of the message, the identifier field, the arbitration process has selected one

[1]This work is supported by the Spanish *Comisión Interministerial de Ciencia y Tecnología* under project number CICYT-TAP96-1090-C04-01

node for transmission. That node continues sending bits to the network, while all the rest keep listening.

Thus, the identifier field of the CAN message is also the priority of the message. This field is sent MSB first. When competing for the bus, dominant bits win arbitration over recessive bits, so the lower identifier, the higher priority.

In this article, a simple method to a dynamic priority assignment to CAN messages is described. It uses the time-to-deadline (also known as *time laxity*) to assign message priority. The resulting scheduling policy is the Earliest Deadline First (EDF). It is a well-known priority assignment that leads to optimal scheduling [7] when applied to a set of tasks competing for the CPU time. Here, this is translated to CAN messages competing for bus access.

To obtain EDF message scheduling, it is necessary to modify a standard CAN controller to accommodate the needs of the scheduling algorithm. This modified version of a CAN controller is called EDF controller. The approach used here also allows applying the priority assignment to the tasks that communicate over the network. In fact, end-to-end scheduling is achieved with the use of dynamic priorities related to the time-to-deadline attribute of every pair of communicating tasks.

Related work

To use CAN as a real-time industrial area network, message time constraints must be guaranteed.

In [11, 12], a deadline monotonic priority assignment is (statically) used to assign fixed priorities (message identifier values) to the messages sent from a set of communicating tasks within CAN. These are sent by a set of tasks that have their period, deadline and length of messages transmitted described in a standard workload defined in [3] for the automotive industry and known as the SAE benchmark. A theoretical bound is obtained to calculate the worst-case response time of any message, and the SAE workload is proven to be schedulable when using data rates over 125 Kbps.

Another static scheduling is given in [8] for the SAE benchmark using bandwidth reservation to guarantee message deadlines.

Two dynamic scheduling algorithms are used in [13, 14] and [9, 10] over a CAN network. Both make use of a global clock (through the use of a synchronisation protocol). Only those messages with tight deadlines have their waiting time improved in the first one.

In [9, 10] hard real-time messages reserve time-slots to guarantee their deadlines; however, before the reserved slot arrive, these messages compete with soft real-time messages for bus access to improve transmission times. Soft real-time messages use their time laxity as priority.

However, priority updates are performed by the communication subsystem, increasing the overhead of the CPU nodes. For this overhead to be acceptable, the time granularity resulted of an order of magnitude larger than the period of the events it is supposed to help scheduling (message transmissions).

End-to-end scheduling

In the SAE benchmark, two sets of tasks are described, periodic and sporadic. A periodic task is a repetitive task with period P that sends a message of length L that must be transmitted before the deadline (D time units after the period begin) is reached. It is assumed that deadline value is less than or equal to the corresponding period. A sporadic task is similar except that has no period but a minimum inter-arrival time.

A periodic task can be described by figure 1. After activation at time t_a (the sending task becomes ready to execute), the message must be transmitted before D time units have elapsed (at time $t_a + D$).

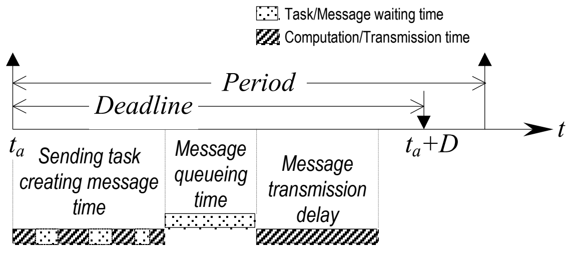
Task/Message waiting time
Computation/Transmission time
Period
Deadline
t_a
t_a+D
t
Sending task creating message time
Message queueing time
Message transmission delay

Figure 1. Periodic task attributes

The time from task activation to message transmission can be divided into 3 parts. First, sender task must compute some value that will be delivered on the message; this time is called T_S (sending task time) and includes the time spent by the task waiting for CPU service (T_S^W) and the execution time (T_S^X). Second, the message built has to wait into the controller message queue while higher priority messages are sent; this is called T_Q. And finally, the message is physically transmitted over the network; this time is called T_T.

From the previous three time delays, only the last one is known a-priori. Given the length of the message and the data rate, the time T_T needed to transmit the message bits over the network wires can be easily obtained. The task computation time T_S depends on the task code, the data being processed, and the overall workload of the CPU and its scheduling policy. The queuing delay T_Q is the time needed to transmit all the messages in the network with higher priorities and depends on the bus load and the message own priority.

When the receiving task is taken into account, a fourth time must be considered (see figure 2). This is the time the receiving task needs to process the message. This time is called T_R (receiving task time) and, like T_S, includes the execution time (T_R^X) and the time the task is waiting for the CPU (T_R^W). To guarantee that receiving task will process the message before the deadline is reached, scheduling policies which takes all these times into account must be applied.

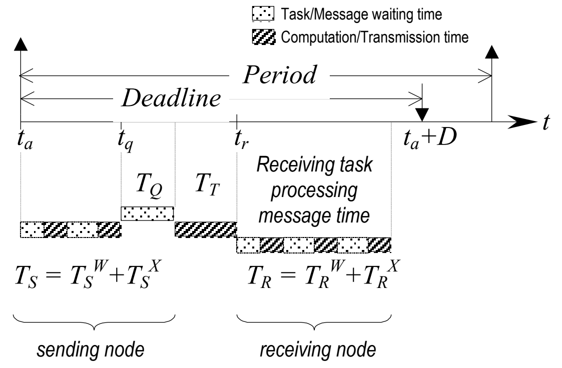

Figure 2. Sending and receiving tasks

To obtain an end-to-end scheduling policy, the transmitting node scheduler must take T_S^X, T_T and T_R^X into account to assign the priority of the task when competing for the CPU. In EDF, the priority of a task at any given time is the time left to the task deadline (also known as *time laxity*).

If we call *R(t)* the remaining time at time *t*, $R(t_a) = D - (T_S^X + T_T + T_R^X)$. At activation time, T_S^X and T_R^X can only be estimated with the *worst case execution time* (*wcet*) of send and receive tasks. If the sender begins execution at any point in time after $R(t_a)$ time units from activation, the scheduling will fail.

In a CAN network, all messages are sent through broadcast. It is then possible that several tasks (in the same or different nodes) receive and process a single message. When this is the case, the most stringent case (the maximum T_R^X) must be used.

After the task has finished and built the message to be sent, this is queued into the communication controller at time t_q. The remaining time at this moment $R(t_q)$, must take into account the elapsed time since activation, the message transmission delay, the receiving task computation time and the time to deadline, so $R(t_q) = D - (T_S^W + T_S^X) - (T_T + T_R^X)$. Again, T_R^X can only be estimated by its wcet. However, T_S^X is the actual execution time of the sending task because at t_q the sending task has ended.

The expression $T_S^W + T_S^X$ is the time used by the sending task to compute and queue the message. This can be rewritten as t_q -

t_a, or *now* - t_a at the moment of message queuing. The expression $T_T + T_R^X$ is the minimum time needed for the message to traverse the network and be processed by the receiving task.

The operating system of the sender node is responsible to activate the task every *P* time units. Almost every OS include a service to obtain the actual time (*now*), and it should be easy to add a service to obtain the activation time of the current task (t_a). The final expression for $R(t_q)$ is then $R(t_q) = D - (now - t_a) - (T_T + T_R^X)$.

When the message is sent over the network it reaches the receiving node at time t_r. The remaining time then is $R(t_r) = D - (t_r - t_a) - T_R^X$.

The use of the value t_a for the receiver scheduling imposes several problems. First, t_a is the activation time of the sender task executed in the sender node, and this information must be delivered to the receiver somehow. Second, t_a is a point of time measured from the sender node that has meaning in the receiver node only if both nodes share the same time clock (t_a is an "absolute" time value).

EDF requirements

There are three scheduling stages in the end-to-end communication. First, the transmitting task competes for the CPU at the sender node. Second, the message to be sent competes for the bus. Third, the receiving task competes for the CPU at the receiving node. If all these stages use EDF scheduling and time information is delivered from stage to stage, the whole system will be EDF scheduled.

At the sender node, T_T can be computed as depends on the time length and network data rate, which are known values. T_S^X and T_R^X can be estimated with worst-case execution times of sender and receiver tasks, respectively. All this information can be introduced into the scheduler at the sender node to schedule tasks using EDF policy. When the sending task becomes ready, its laxity time and priority is $R(t_a) = D - (T_S^X + T_T + T_R^X)$.

For the bus to use EDF, the laxity time for any real-time message must be used to assign the message priority within the network. To accomplish this requirement, the laxity time at t_q is directly used as the message priority. In fact, as the lower the CAN message identifier the higher the priority, if this identifier is the remaining time of the message, those messages with less laxity time will be delivered first. At queuing time, it is easy to compute the laxity time $R(t_q)$ as $R(t_q) = D - (now - t_a) - (T_T + T_R^X)$ and use this as the message priority.

However, once the message has been queued and compete with other messages for the bus, this priority must be dynamically updated as time goes.

Suppose that a message m1 is queued at time 5 with 7 time units as its laxity time and is waiting for the bus to be idle to begin transmission. At time 10 a second message m2 is queued with 4 time units as the remaining time and bus becomes idle at time 11. This example is illustrated in the following figure.

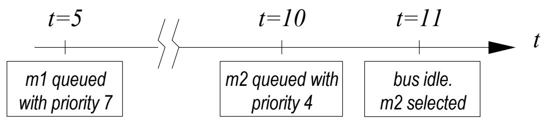

Figure 3. Fixed priorities example

If fixed priorities are used (e.g. deadline monotonic as in [11]), message m2 will be transmitted first. But the message with lower laxity time at t = 11 is message m1, so it should be transmitted first. To avoid this kind of priority inversion (a lower priority message being transmitted while a higher priority message is waiting), priorities should be dynamically updated. If the time-to-deadline for each message is periodically updated, message m1 will have priority 1 and m2 priority 3 at time 11. The message transmitted will be the lower remaining time, m1, as shown in the following figure.

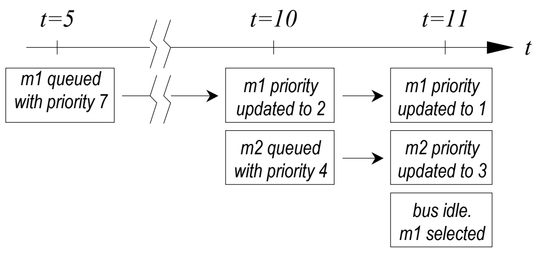

Figure 4. Dynamic priorities example

For this update to be effective, the update frequency must be higher than the maximum frequency of the events on the bus. In the preceding example, if the update period is larger than 6 time units, the schedule will fail. This frequency requirement invalidates any update mechanism that uses the same CPU that executes tasks and queues messages to be sent over the bus.

When the message arrives at the receiving node, the message priority (the CAN identifier) carries the time left to deadline $R(t_r)$. This is the laxity time for the activation of the receiver task. The message priority has taken T_T and the wcet of T_R^X into account, so the value of the priority field when the message arrives at the receiver node is the maximum time the receiver task can delayed.

If the receiving task begins execution at any point of time $R(t_r)$ time units after message reception, the scheduling will fail.

EDF controller design

This section is devoted to reveal the differences between a standard CAN controller and a controller to achieve EDF scheduling, which will be denoted as EDF controller.

First, queuing a high priority message while a low priority message is waiting should not help a lower priority message in another node gaining access prior to the high priority message. This problem has been shown in [12] for the 82c200 CAN controller, with a single transmission buffer [4]. To queue the high priority message, the lower priority message must be released first from the transmission buffer. Between the time the lower priority message is released and the time the higher priority message is copied into the controller transmission buffer, no message can begin transmission for this node. It is then possible that a lower priority message from other node gains access to the CAN network. This is avoided when the CAN controller offers more than one transmission buffer, because no buffer release is needed to queue the high priority message.

This problem persists even on new controller designs, like the SJA1000, described in [5].

Second, message ordering in the queue of the controller should be based on the priority field of the message. Interestingly, only some the newest controller designs (e.g. the Motorola's TouCAN controller [6]) use this approach, and as "added" feature. Message ordering inside a CAN controller with more than one buffer is usually done through the buffer slot number. Message stored in the first slot is transmitted first and competes with its priority to gain access to the bus with the rest of messages of other nodes. This breaks the EDF rule, and must be avoided. In [13], this problem is avoided ensuring that no node has more messages to transmit than transmission slots in a CAN controller.

Finally, once a message is built and ready for transmission, its priority must be periodically updated at a frequency higher than maximum message transmission frequency. This can be achieved by specialised hardware only, a down counter associated with each message identifier on the queue. As the laxity time used as message priority is a time interval value (a count of time units) and not an “absolute” time, there is no need for the clocks of the nodes to be synchronised for this purpose.

Not all the bits in the message identifier field can be used to store the laxity time for the message. The CAN standard requires message identifiers to be unique across the network. In [13], eight bits are used to store the laxity time of the message. This value is encoded and stored into the most significant bits of the CAN identifier.

However, deadline values for the set of tasks described by the SAE benchmark range from 5ms to 1000ms. To use a direct encoding (as proposed in [13]) the time granularity of the encoded time laxity must be 4ms or more. With a set of tasks that execute and send messages every 5ms, this time granularity is impractical.

The approach taken with the EDF controller is a two-level encoding, as shown in figure 5. Time granularity is reduced to the maximum for tight deadlines, while a larger time period (2^K times the time granularity) is used for larger deadlines. Those deadlines encoded with the former time granularity are called fine-grained deadlines; when the latter is used they are called coarse-grained deadlines.

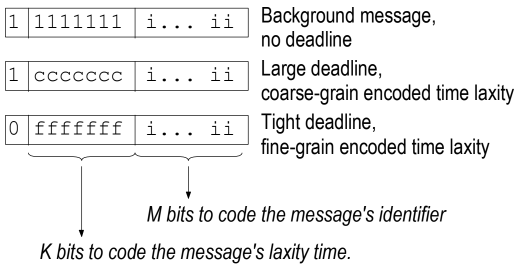
1 1111111 i... ii
Background message, no deadline
1 ccccccc i... ii
Large deadline, coarse-grain encoded time laxity
0 fffffff i... ii
Tight deadline, fine-grain encoded time laxity
M bits to code the message's identifier
K bits to code the message's laxity time.

Figure 5. Priority encoding

When a coarse-grain deadline reaches its last period, it is automatically translated into a fine-grain deadline. When a fine-grain deadline reaches 0, the message has lost its deadline and an interrupt is immediately raised to inform the system. To ensure those tight deadlines (coded with fine-grain time granularity) will have always the highest priority, they are preceded by a dominant bit.

As shown in the figure above, the CAN identifier is divided into 3 parts. The MSB bit is a dominant/recessive bit to distinguish between tight and large deadlines. The next K bits are used to encode the time laxity of the message. This leaves M = L - K - 1 bits for the message identifier, being L the total number of bits into the CAN identifier field (11 bits on standard frames, 29 bits on extended frames). This arrangement ensures that messages will have a unique CAN identifier and that messages with lower laxity times will have higher priority over large laxity time messages.

Each coarse-grain period is 2^K times a fine-grain period. With K=7, and a fine-grain of 64 μs, each coarse-grain period is greater than 8 ms. The larger time laxity than can be encoded is greater than 1000 ms. This ensures the whole range of deadlines of the SAE benchmark is covered. Tight deadlines (laxity times up to 8 ms) are updated every 64 μs while larger deadlines are updated every 8192 μs.

Design considerations for background messages

Not all the messages in a network have time constraints. There are also background messages that should be sent only when the bus is idle and no time-constrained message is waiting. These messages are usually related to added functionality of the system, not needed for the basic function.

The EDF controller is able to work in a different way for these messages. As background messages, no laxity time is used and the priority is never updated. One control bit of the transmission buffer is used to enable the priority update. If this update is disabled and the laxity time field is filled with recessive bits, these messages never reach the priority levels of any time-constrained message.

EDF controller design details

In this section, a detailed explanation of the controller design trade-offs is shown.

The main objective of this work is to modify a standard controller design as little as possible, resulting into reasonable silicon costs.

To achieve this goal, the easiest way is to use two global clock lines (fine and coarse clocks) to update priorities of all the messages into the controller, as shown in figure 6.

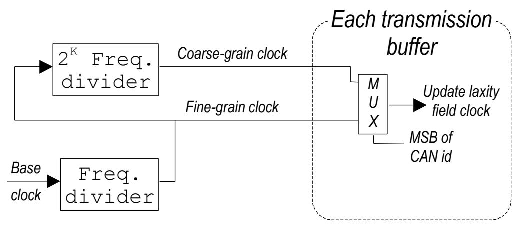

Figure 6. Two global clock lines

This will minimise the silicon costs. However, this approach leads to large quantization errors when the application maps the laxity time $R(t_q)$ within the two-level encoding mechanism used by the EDF controller that must be taken into account.

As both clock dividers are asynchronous to message queuing events, using this design will lead to a mapping worst case error of $2^{(K+1)}\ T_{fg}$, being T_{fg} the fine-grain period.

This worst case comes from a coarse-grain message with a time laxity of $((N+1)2^K - \Delta)T_{fg}$, $\Delta \to 0$, queued just before the coarse-grain period expires.

The application maps the message's laxity into $2^K T_{fg}$ blocks (coarse-grain periods), giving N. As the coarse period is about to expire, this laxity is immediately updated to N-1. The mapping error is ε = *actual laxity - laxity mapped value* = $((N+1)2^K - \Delta)T_{fg} - (N-1)2^K T_{fg} = 2^{(K+1)}T_{fg} - \Delta T_{fg}$.

With $\Delta \to 0$, the error is $\varepsilon = 2^{(K+1)}T_{fg}$.

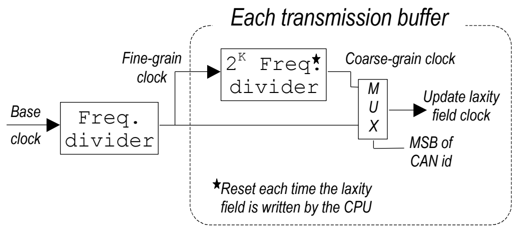

Figure 7. EDF controller design detail

To reduce this mapping error, the controller design (see figure above) uses a single, programmable frequency divider to obtain the fine-grain clock from a base clock. Each transmission buffer has its own clock divider to obtain the coarse-grain period; this is re-started each time the CPU writes the laxity field of the CAN identifier.

As the coarse-grain period starts when the message is queued, the worst case mapping error is reduced to $\varepsilon = ((N+1)2^K - \Delta)T_{fg} - 2^K N T_{fg} = 2^K T_{fg} - \Delta T_{fg} = 2^K T_{fg}$.

Following the above reasoning, the worst case mapping error for a fine-grain message is $\varepsilon = 2T_{fg}$. These mapping errors must be included into $R(t_q)$ to perform schedulability tests, as follows:

$$R(t_q) = D - \varepsilon - (now - t_a) - (T_T + T_R^X)$$

To summarise, an EDF controller is a CAN controller with these added features: i) it offers more than a single transmission buffer, ii) the messages ready for transmission in the controller are ordered by their priority, and iii) the message priority is periodically updated without CPU intervention and with small mapping errors.

The EDF controller design has been targeted on an Altera's EPF10K10 macrocell-based PLD device. The silicon size increase is well below 5% when a single transmit buffer is taken into account. When the size of the whole controller is used, the silicon costs induced by the EDF requirements are negligible.

Conclusions

In this paper, the CAN network has been shown as a good choice to schedule a set of communicating tasks with hard real-time constraints. A well-known dynamic scheduling algorithm, EDF, has been applied.

Even better, CAN messages deliver automatically-updated, valuable time information from sender to receiver nodes allowing the use of EDF not only in the network but also as the scheduling algorithm for the tasks competing for CPU access in a given node (end-to-end scheduling). The whole system schedule is EDF, an optimal dynamic priority assignment based on the time-to-deadline attribute of the object (task, message) being observed.

A modified CAN controller, called EDF controller, makes use of time laxity of messages to order them across the network. The modifications needed over a standard CAN controller have also been shown to be of a reasonable cost. These are focused on the message queue of the controller, that meets the characteristics of being priority-ordered and periodically updated.

Finally, as not every message in the network has a deadline, background messages are included in the EDF controller with minor modifications.

References

[1] *Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communications*. ISO DIS 11898, February 1992.
[2] *CAN Specification version 2.0*. Robert Bosch GmbH, 1991.
[3] *Class C Application Requirements Considerations*. SAE Technical Report J2056/1, 1993.
[4] *PCA82C200 Stand alone CAN controller data sheet*. Philips semiconductors.
[5] *SJA1000 Stand alone CAN controller data sheet objective specification*. Philips semiconductors.
[6] *MC68336/376 User's Manual*. Motorola Semiconductors, 1996.
[7] H. Chetto and M. Chetto. *Some results of the Earliest Deadline Scheduling Algorithm*. IEEE Trans. on Software Eng., vol. 15, no. 10, October 1989.
[8] H. Kopetz. *A solution to an Automotive Control System Benchmark*. Research report 4/1994, Institut für Technische Informatik, Technische Universität Wien, April 1994.
[9] M.A. Livani and J. Kaiser. *EDF Consensus on CAN Bus Access for Real-Time Dynamic Applications*. Lecture notes on Computer Science, vol. 1388, pp. 1088-1097. Springer-Verlag, 1998.
[10] M.A. Livani, J. Kaiser and W.J. Jia. *Scheduling Hard and Soft Real-Time Communication in the Controller Area Network (CAN)*. 23rd IFAC/IFIP Intl. Workshop on Real-Time Programming, China, 1998.
[11] K.Tindell and A.Burns. *Guaranteeing Message Latencies on Control Area Network (CAN)*. Proc. 1st Intl. CAN Conference, 1994.
[12] K. Tindell, A. Burns and A.J. Wellings. *Calculating controller area network (CAN) message response times*. Control Eng. Practice, vol. 3, no. 8, pp. 1163-1169, 1995.
[13] K.M. Zuberi and K.G. Shin. *Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications*. Proc. 5th Real-Time Technologies and Applications Symposium, 1995.
[14] K.M. Zuberi and K.G. Shin. *Real-Time Decentralized Control with CAN*. Proc. Intl. IEEE Conf. on Emerging Technologies and Factory Automation, 1996.

Technical University of Valencia
Dept. Computer Engineering
Camino de Vera, s/n, E46022 Valencia (SPAIN)

Phone:	+(34) 96 387 75 77
Fax:	+(34) 96 387 95 79
E-mail:	prodrig@disca.upv.es
Homepage:	http://www.disca.upv.es