The Flexible Time-Triggered (FTT) Paradigm: an Approach to QoS Management in Distributed Real-Time Systems

Paulo Pedreiras*, Luis Almeida
pedreiras@alunos.det.ua.pt; lda@det.ua.pt
DET/IEETA
University of Aveiro

Abstract

Real-time distributed systems are becoming pervasive, supporting a broad range of applications such as avionics, automotive, adaptive control, robotics, computer vision and multimedia. In such systems, several activities are executed on different nodes and cooperate via message passing. One increasingly important concept is that of Quality-of-Service (QoS), i.e. a system performance metric from the application point-of-view. Concerning the communication system, the QoS delivered to the application is a function of communication parameters such as the rates of message streams. In previous work, the authors have developed two network protocols, FTT-CAN and FTT-Ethernet, which allow on-line changes to the set of message streams under guaranteed timeliness. In this paper, the specific network protocols are abstracted away leading to a generic communication paradigm named Flexible Time-Triggered (FTT), which supports on-line QoS management, with arbitrary policies, in distributed real-time systems. Two possible QoS management policies are referred, priority-based and another one based on the elastic task model, and their use is illustrated with a case study.

1. Introduction

Due to continued developments, along the last decades, in the integration of processing and communications technology, distributed architectures have progressively become pervasive in many real-time application domains, ranging from avionics to automotive, adaptive control, robotics, computer vision and multimedia. These architectures present several advantages: they favor dependability through easy replication of nodes and definition of error-containment regions; composability since the system can be built by integrating nodes that constitute independent subsystems; scalability through easy addition of new nodes to support new functionality; and maintainability due to the modularity of the architecture and easy node replacement.

In these systems, there has also been a trend towards higher flexibility in order to support dynamic configuration changes such as those arising from evolving requirements and on-line Quality-of-Service (QoS) management [6]. These features are generally useful to increase the efficiency in the utilization of system resources [5] since typically there is a direct relationship between resource utilization and delivered QoS. In several applications, assigning higher CPU and network bandwidth to tasks and messages, respectively, increases the QoS delivered to the application. This is true, for example, in control applications [3], at least within certain ranges [12], and in multimedia applications [10]. Therefore, managing the resources assigned to tasks and messages, e.g. by controlling their execution or transmission rates, allows a dynamic control of the delivered QoS. Efficiency gains can be achieved in two situations: either maximizing the utilization of system resources to achieve a best possible QoS for different load scenarios or adjusting the resource utilization according to the application instantaneous QoS requirements, i.e. using only the resources required at each instant.

Both situations referred above require an adequate support from the computational and communications infrastructure so that relevant parameters of tasks and messages can be dynamically adjusted. In our case, we handle this problem from the communications perspective, only, considering an autonomous communication system that manages streams of messages, very much like a processor executes tasks. This approach is more robust and particularly adapted to distributed real-time systems with fault-tolerance requirements [9].

In previous work, the authors have contributed to two new communication protocols that aim specifically at supporting dynamic communication requirements and QoS management in distributed real-time systems, FTT-CAN [2] and FTT-Ethernet [13], which are based on CAN and Ethernet, respectively. Both protocols share the basic architecture and model. In this paper, the underlying network is abstracted away, leading to a general communications model, the Flexible Time-Triggered (FTT) paradigm, which is well suited to support dynamic QoS management with arbitrary policies. Moreover, this paper discusses the use of two possible QoS management policies, one that is priority-based and another that is based on the elastic task model. The use of both policies is illustrated by means of a simplified case study based on a mobile robot.

[1]This work was partially supported by the Portuguese Government through grant PRAXIS XXI/BD/21679/99 and by the European Comission through accompanying measure ARTIST IST-2001-34820.

2. Motivation

Distributed real-time systems depend heavily on the underlying communication subsystem capabilities, since it is this subsystem that provides the services that support the cooperation between system nodes, which is performed exclusively via message passing. Moreover, some of the activities carried by distributed real-time systems have strict deadlines that must be met in all anticipated circumstances, thus, the communication subsystem must not only deliver the messages but also, when required, do it within specific time boundaries. For this reason, distributed real-time systems are based on specific communication subsystems, able to convey time constrained message exchanges, which are usually known as real-time communication networks.

As discussed in Section 1, dynamic QoS management implies on-line changes to the traffic characteristics, such as addition, removal and adaptation of message properties. On other hand, some of the message streams have real-time QoS constraints, arising for example from control and monitoring requirements, which must be always fulfilled. Unfortunately, flexibility and timeliness have typically been considered separately and most of the real-time networks available today favor either one aspect or the other [15], i.e., either time-constrained services are guaranteed sacrificing flexibility or such guarantees are sacrificed in exchange for higher flexibility. Thus, most of the existing communication protocols are not well suited to support the flexibility requirements presented by systems that implement dynamic QoS management functionalities. On other hand, protocols such as IBM Token Ring, FDDI and ATM have support for such QoS requirements, but are not broadly used in distributed real-time systems because of outdated technology or high cost.

A debate that has been around for a while is the one opposing time to event-triggered architectures. The former ones are synchronous and favor determinism and temporal isolation between different streams, which supports composability with respect to the temporal behavior[8]. They are particularly well suited to support periodic messages. However, current examples are either static regarding the time-triggered traffic definition, such as TTP/C, TT-CAN and FlexRay, and/or they are considerably inefficient in handling sporadic (even-triggered) traffic, e.g. TTP/C, WorldFIP and Foundation Fieldbus-H1. On the other hand, event-triggered architectures are asynchronous, handle sporadic traffic efficiently as well as dynamic communication requirements, but cannot enforce temporal isolation between different streams and thus do not support composability with respect to the temporal behavior. As examples we can refer Profibus, P-Net, DeviceNet.

In this context, the FTT paradigm, which is described next, appears as an attempt to jointly fulfill the requirements for timeliness, flexibility and efficiency. It was specifically designed to exhibit the following features: time-triggered communication with operational flexibility; support for on-the-fly changes both on the message set and on the traffic scheduling policy; on-line admission control of the real-time traffic; support for different types of traffic with temporal isolation (time and event-triggered, hard, soft and non real-time); and finally efficient use of network bandwidth.

3. The FTT paradigm

The FTT paradigm uses an asymmetric synchronous architecture, comprising one master and several slave nodes. The master node is responsible for the management and coordination of the communication activities and it may also execute application software. The slave nodes execute the application software as well as the network protocol.

The master node implements the centralized scheduling concept, in which both the communication requirements, message scheduling policy, QoS management and on-line admission control are localized in one single node. Such concentration supports a complete knowledge of the instantaneous system requirements as well as the possibility to make atomic changes over them. This feature facilitates considerably the implementation of on-line admission control and QoS management. On the other hand, it is considered by many as inadequate to applications with safety and availability requirements due to the single point of failure formed by the master. However, the single point of failure can be eliminated by using redundant backup masters with appropriate election and synchronization mechanisms.

The scheduling decisions taken in the master are broadcast to the network using a special periodic control message called trigger message (TM). Slaves decode the TM and transmit their messages if instructed to, in a master-slave fashion. The typical overhead of master-slave communication is substantially reduced by using one single TM to trigger the transmission of several slave messages, possibly from distinct slaves. The timeliness of the system depends essentially on the timeliness of the master. We call this scheme master/multi-slave transmission control.

By using centralized scheduling and consistent interfaces between the scheduler, dispatcher, QoS manager and admission control, together with the distribution of the schedule decisions by means of the trigger message, the system gets a high degree of flexibility since:

- Changes on the message set properties, resulting for instance from the admission or removal of message streams, are performed internally on the master node and distributed by the network nodes via the trigger message, thus the synchronization of the update among the network it is intrinsically guaranteed.
- The master holds enough information to know the demands of real-time traffic and how much leeway the system has, therefore can safely allocate bus bandwidth to other kinds of traffic without jeopardize the timeliness of real-time traffic.
- The station nodes don't need to be aware of the particular scheduling policy in use, since they strictly follow the schedule conveyed in the trigger message.

3.1. The Elementary Cycle

In the FTT paradigm the bus time is slotted in consecutive fixed duration (E) time-slots, called Elementary Cycles (ECs).

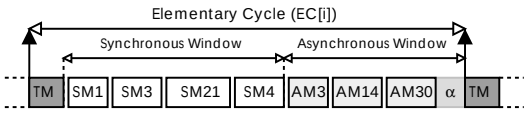

Figure 1. The Elementary Cycle structure

The EC starts with the reception of the TM, and all nodes are synchronized by the reception of this message. Within each EC are defined two consecutive windows, synchronous and asynchronous, that correspond to two separate phases (Figure 1).

The synchronous window conveys the time-triggered traffic, specified by the trigger message. The length of the synchronous window ($lsw(i)$) can vary from EC to EC, according to the number and size of messages scheduled for each particular EC. It is however possible to impose a limit to the maximum size of the synchronous window (LSW), and thus grant to the asynchronous window a minimum guaranteed bandwidth share. The time-triggered traffic is subject to admission control and thus all messages accepted by the system have its timeliness guaranteed (dynamic planning-based scheduling).

The asynchronous window has a duration ($law(i)$) equal to the remaining time between the EC trigger message and the synchronous window. It is used to convey event-triggered traffic, herein called asynchronous because the respective transmission requests can be issued at any instant, by the application software. Unlike the synchronous traffic, the arbitration within the asynchronous window is not resolved by the master node. The only information supplied in the trigger message (either implicitly or explicitly, depending on the particular implementation) is the duration of the asynchronous window. A suitable protocol must thus be used to perform the message serialization within this window. The asynchronous traffic is handled according to a best-effort policy. However, the use of deterministic medium-access policies combined with the possibility to define a minimum guaranteed bandwidth to the asynchronous traffic allows, when required by the application, to pre-analyze its requirements and compute whether a given set of real-time asynchronous messages can met its deadlines in worst-case conditions. This feature is usually required only by asynchronous messages related to alarms or other similar real-time events.

In order to maintain the temporal properties of the time-triggered traffic, such as composability with respect to the temporal behavior, the synchronous window must be protected from the interference of asynchronous requests. A strict temporal isolation between both phases is enforced by preventing the start of transmissions that could not complete within the respective window. Since the message lengths are not correlated neither with the EC duration neither with the synchronous and asynchronous window durations, a short amount of idle-time (α) may appear at the end of the asynchronous window.

The communication services of the FTT paradigm are delivered to the application by means of two subsystems, the Synchronous Messaging System (SMS) and the Asynchronous Messaging System (AMS), that manage the respective type of traffic. The SMS offers services based on the producer-consumer model [16] whilst the AMS offers send and receive basic services, only. The components of each of these services are spread among the master and the station nodes, and presented in the following sections.

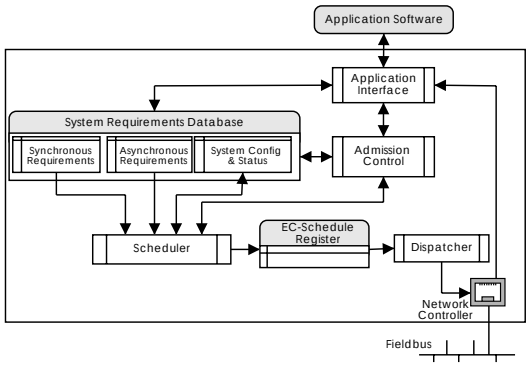

Figure 2. FTT master internal architecture

3.2. Master node architecture

The master node plays the role of system coordinator and provides an interface to allow system management, maintain a local database holding the system communication requirements, build schedules according to the traffic scheduling policy in use and broadcast this schedules at appropriate time instants, encoded in the TM. Figure 2 depicts the internal architecture of an FTT master.

The **Application Interface** provides a set of services that are used by the application software to perform the system management, such as system configuration (set-up of the EC duration, bus speed, network topology, etc.), message management (addition and elimination of messages, as well as modification of properties) and system status retrieval (information about the system performance, e.g. jitter figures and delays). As depicted in Figure 2, API services are accessible both locally, by the application software running in the master node, and remotely by the application software executing in slave nodes. The FTT protocol defines a set of dedicated asynchronous messages that support full remote access to the master's API.

The **System Requirements Database** (SRDB) holds the properties of each of the message streams to be conveyed by the system, both real-time and non-real-time, as well as a set of operational parameters related to system configuration and status. This information is stored in a set of three tables.

The Synchronous Requirements Table (SRT) includes the properties of the N_S synchronous messages carried in the system (Definition 1).

$$SRT \equiv \{SM_i(DLC_i, C_i, Ph_i, P_i, D_i, Pr_i, *Xf_i),\ i = 1..N_S\} \quad (1)$$

where for each message SM_i, DLC_i is the data length in bytes, C_i is the respective transmission time (including all overheads), Ph_i allows to define an initial phase, P_i is the period or minimum inter-arrival time, respectively for periodic and sporadic messages, D_i is the deadline and finally Pr_i is a

fixed priority. The basic time unit in the FTT paradigm is the elementary cycle, thus both Ph, P and D are expressed as integer multiples of the EC duration (E). Synchronous message identification is related to the message contents and not with the particular sender or consumer(s), i.e., source addressing.

Besides the basic properties defined above, the SRT also provides a custom data structure ($*Xf$ in Definition 1) that can be used to extend the system functionalities. For instance, if it is required to support message streams with different levels of acceptable Quality of Service (QoS) concerning the respective bandwidth, the SRT can be extended with an admissible period range (Minimum, Nominal and Maximum) as well as other parameters required for the particular QoS management policy. On the other hand, this mechanism also allows to implement security constraints, supporting restrictions to the operations that can be performed over the message streams. For example, some flags can be used to indicate which messages can or cannot be deleted or if the QoS manager can automatically update their properties.

The Asynchronous Requirements component is composed by the reunion of two tables, the Asynchronous Requirements Table (ART) and the Non-Real-Time Requirements Table (NRT). The ART (Definition 2) is used to store the properties of the asynchronous messages conveyed by the system that, despite being asynchronous, may or may not have timeliness requirements. For example alarm messages usually have hard timeliness requirements while messages used to remote diagnosis or configuration do not have such timeliness constraints.

$$ART \equiv \{AM_i(DLC_i, C_i, mit_i, D_i, Pr_i),\ i = 1..N_A\} \quad (2)$$

This table is similar to (1) except for the use of mit_i, minimum inter-arrival time, instead of period, and the absence of initial phase Ph_i, since there is no phase control between different asynchronous messages. As in the case of the synchronous messages, the asynchronous message exchange is based on the producer-consumer co-operation model, therefore it uses also source addressing.

The non-real-time traffic is handled strictly according to a best-effort policy. Since no timeliness guarantees are provided, the master node only needs to keep track of which stations produce this kind of traffic, and, for each of them the size of the respective longest non-real-time message, as required to enforce the temporal isolation between synchronous and asynchronous traffic.

$$NRT \equiv \{NM_i(SID_i, MAX\_DLC_i, MAX\_C_i, Pr_i), i = 1..N_N\} \quad (3)$$

The NRT contents is described in Definition 3 and characterizes the communication requirements of each one of the N_N nodes producing non-real-time messages. More specifically, SID_i is the sender's identifier , MAX_DLC_i is data length in bytes of the longest non-real-time message sent by the node, MAX_C is the respective maximum transmission time, including all overheads and Pr_i is the node's non-real-time priority, which can be used to implement an asymmetric distribution of the bus bandwidth among the different nodes with respect to this type of traffic only.

The **Scheduler** uses the information provided by the SRDB to build the EC-schedules for the synchronous traffic. More specifically, the Scheduler scans the SRT and ART as well as the system configuration information stored in the SCSR register, and, based on such data, decides which synchronous messages should be transmitted in the following EC, according to the particular scheduling algorithm implemented. The result of such computation is placed in the EC-Schedule register.

The **Admission Control** is based on the schedulability test of the synchronous traffic. The schedulability test depends on the particular scheduling algorithm being used as well as on the information within the SRDB, such as the maximum length of the synchronous window. The admission control is invoked whenever there is a request for a change in the SRT.

The **Dispatcher** reads the EC-Schedule Register, builds the next trigger message with such EC schedule and broadcasts it over the network. Since it is the reception of the trigger message in the remaining nodes that signals the beginning of an EC, it is important that the Dispatcher broadcasts such message regularly, with sufficient precision.

3.3. Slave node architecture

Slave nodes, also known as stations, execute the application software required by the user, eventually requesting the services delivered by the communication system.

The application software interacts with the communication system through a real-time API (RT_API) which enables the applications to: define which messages are locally produced or consumed; update and read the value of such real-time entities; set-up callbacks associated to communication events (e.g. message transmission, reception, deadline misses, etc.). The local communication requirements are kept in the Node Requirements Database (NRDB).

The exchange of synchronous messages is performed with autonomous control, i.e. the transmission and reception of messages is carried out exclusively by the network interface without any intervention from the application software. The message data is passed to and from the network by means of shared buffers. There are two complementary API functions available to the application layer, SMS_produce and SMS_consume, which allow producer nodes to update the local buffers with new data and consumer nodes to read the actual contents of the local buffer, respectively .

The transmission of the real-time asynchronous messages follows the external control paradigm, i.e. the transmission of messages takes place upon explicit requests from the application software. Such requests are issued by means of a basic API service called AMS_send, which is a send function with queuing. The queue is ordered first by priority, according to the message identifiers, and second by request instant (FCFS). The delivery of messages to the application software is accomplished by means of a complementary API basic service called AMS_receive, a receive function that allows waiting for a specified, or unspecified message. More complex and reliable exchanges, e.g. requiring acknowledge or requesting data, must be implemented at the application level, using the two basic services referred above.

The FTT Interface Layer receives and decodes the EC trigger message and transmits messages that carry entities produced locally and requested elsewhere, according to the information of the EC-Schedule. On reception of real-time frames the FTT Interface Layer matches its ID with the list of the locally consumed entities, by checking the NRDB. If the received message is locally consumed, its local buffer/queue is updated with the received data.

3.4. Feasibility tests

Hard real-time systems demand a high degree of predictability, thus the feasibility of the schedule should be guaranteed in advance. Moreover, in flexible on-line scheduled systems like FTT, messages can be created, changed and removed dynamically during runtime. In this case a suitable admission control mechanism is required to assess on-line if such operations can be accepted, that is, if the resulting message set is schedulable.

As discussed in Section 3.1, the scheduling model used for the synchronous traffic does not allow the transmission of messages to cross the boundary of the synchronous window. This is achieved by using inserted idle-time, i.e., whenever a message does not fit completely within the synchronous window of a given EC it is delayed to the next. This same behavior is also enforced in the asynchronous window, despite its implementation being somehow different. Consequently, the EC trigger message is always transmitted regularly, without any blocking. However, the use of inserted idle-time has also a negative impact on the traffic schedulability, since within the synchronous window it corresponds to a reduction on its length, and on the asynchronous window it corresponds to bus time that is wasted, since no messages are transmitted in it.

In [1], *Almeida et al* present several techniques for the schedulability analysis of task sets scheduled with inserted idle-time, in similar conditions to those referred above. The model, named *blocking-free non-preemptive scheduling*, considers tasks periods and deadlines as integer multiples of a basic cycle duration (E), the execution times as being always shorter than E and task activations always synchronous with the start of a cycle. The only difference is that in [1] the whole cycle is available to execute tasks, while in the FTT model the synchronous traffic is restricted to the synchronous window within each EC, with maximum length LSW.

In particular, one of those techniques is based on the adaptation of the existing analysis for preemptive scheduling of tasks with fixed priorities and requires a simple modification to account for the impact of the EC trigger message and asynchronous phase, therein not considered. An important corollary of this theorem is that Liu and Layland's utilization bound for Rate Monotonic [11] can be used with just a small adaptation as part of a simple on-line admission control for changes in the SRT incurring in very low run-time overhead. This is expressed in Condition 4, where X is an upper bound for the inserted idle-time.

$$\sum_{i=1}^{N_s} \frac{C_i}{P_i} < N_s \left(2^{\frac{1}{N_s}} - 1\right) \left(\frac{LSW - X}{E}\right) \Rightarrow \begin{array}{c} SRT\ Sched. \\ RM\ under \\ any phasing \end{array} \quad (4)$$

A similar line of reasoning can be followed to adapt the Liu and Layland's utilization bound for EDF [11]. In this case, the sufficient schedulability condition is expressed by Condition 5.

$$\sum_{i=1}^{N_s} \frac{C_i}{P_i} \leq \frac{LSW - X}{E} \Rightarrow \begin{array}{c} SRT\ Schedulable\ with \\ EDF\ under\ any phasing \end{array} \quad (5)$$

Both of the analysis presented above are pessimistic, because they consider that the inserted idle-time always has its maximum value, thus leading to an analysis that is sufficient, only. However, it must be recalled that these schedulability tests are to be executed on-line. In systems that are highly dynamic, with frequent changes to the message set or in which the system's response to change requests must be prompt, schedulability tests must have low computational complexity. Both schedulability tests presented above have a complexity of $O(n)$, similar to the one of the original Liu and Layland's analysis [11].

4. QoS management based on the FTT paradigm

As discussed in Section 3, the scheduling activity is performed on-line, based on the current message properties stored in the SRDB (Figure 2). This mechanism is the source of the operational flexibility exhibited by the FTT architecture, concerning the synchronous traffic. When the message set is changed, in its next activation the Scheduler will use the updated values, and thus the following EC-Schedules include the new communication requirements.

In its most basic functionality level, the FTT paradigm requires change requests to be handled by an on-line admission control. The purpose of this mechanism is to assess, before commitment, if the requests can be accommodated by the system i.e., if the message set that would result of the incorporation of the requested changes would be still schedulable. In this case, the changes can be safely committed to the SRDB, and consequently the request is accepted. Conversely, if the change request would result in an unfeasible message set, it is rejected and the SRDB is kept unchanged.

From this point of view the master node can be seen as a QoS server, in the sense that when a message is admitted or changed, the master node verifies if its associated requirements (memory, network bandwidth, message deadline and jitter, etc.) can be fulfilled, and in this case also reserves these resources in a way that they will be strictly available in the future, assuring that all the accepted messages will receive the requested QoS.

As discussed in Section 2, many recent real-time applications exhibit a high degree of flexibility. Concerning particularly QoS requirements, in some cases such applications require and in other cases benefit from, the definition of ranges of acceptable QoS levels. In these cases, some of the system activities can vary their requirements during the system lifetime, in response to environment changes. To handle these requirements efficiently, the communication protocol should not only guarantee that the minimum requirements will be fulfilled in all anticipated conditions, but also grant at all instants

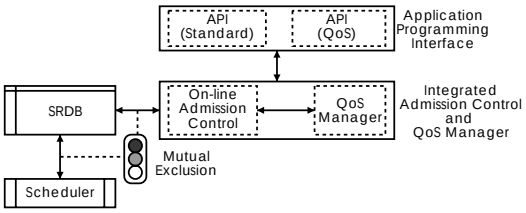

Figure 3. Adding QoS management to FTT

the higher QoS possible to all the activities. Moreover, it can be required to support different levels of importance for these activities, implying that some of them can be favored with respect to the others, according to some well defined policy.

4.1. Adding a QoS manager

The FTT paradigm can provide support for advanced QoS management methodologies by aggregating a QoS manager to the on-line admission control block. With this architecture, the on-line admission control still decides about the acceptance of change requests based on the minimum requirements of the existing message streams. This will eventually generate some spare resources, e.g. spare bandwidth, that will be distributed by the QoS manager according to a pre-defined policy.

As described in Section 3.2, the master node holds in the Synchronous Requirements Table the properties of the synchronous message set. It was also referred in that section that besides the basic message properties (Period, Deadline, etc.), the SRT also provides room for additional fields, Xf in Definition 1. The QoS manager can use this field to store the relevant properties for each of the synchronous messages. Examples of such properties is the specification of the admissible QoS ranges, relative importance, criticalness, etc.

The FTT paradigm is based on a modular design, with well defined interfaces between the system components. The scheduler bases its decisions on the actual contents of the SRT, so the QoS manager must map the communication requirements into standard message properties, such as periods (for an RM scheduler) and deadlines (for an EDF or DM scheduler). Moreover, SRT updates cannot be performed while the Scheduler is reading its contents for building the following EC, therefore it is necessary to enforce mutual exclusion between the SRT updates and the scheduling activity. If both of these properties are enforced, the operation of the Scheduler becomes completely independent not only of the existence of a QoS manager but also from the particular QoS management policy used. With respect to the Application Interface, the aggregated on-line admission control and QoS manager must implement the standard SRDB management functions (add, remove and change message properties), but can also extend the API to provide QoS management user-level functions, which are specific to the particular QoS management policy implemented and allows the application to request the appropriate QoS in response to environment changes.

4.2. Priority-based QoS management

Many real-time systems are composed by sets of activities with distinct levels of importance concerning the behavior of the system. In these cases, QoS should be granted strictly according to the relative importance of these activities, with the more important ones receiving the highest QoS possible. A methodology to deal with this situation consists in assigning a QoS priority parameter to each of the activities. Then the QoS manager sorts the activities according to the QoS priority and distributes the required QoS to each one, when possible.

As discussed in Section 2, the concept of QoS is broad. Concerning specifically real-time communications, usual requirements are bandwidth, response-time and jitter.

To cope with this framework the SRT (Definition 1) should be extended to incorporate the above referred properties.

$$Xf_i \equiv (V_i, T_{i_{min}}, T_{i_{max}}, Rt_i, J_i),\ i = 1..N_S \tag{6}$$

where V_i specifies the relative message importance, the minimum $T_{i_{min}}$ and maximum $T_{i_{max}}$ periods bound the bandwidth required by each message, Rt_i, indicates an upper-bound to the admissible message's response-time and J_i the upper-bound for the admissible message's jitter. The QoS priority field is required, while the remaining fields are used to specify one or more of the QoS requirements, and thus can be used all or only a few, depending on the particular requirements.

4.3. Elastic Task Model based QoS management

One of the characteristics of the priority-based QoS manager above presented is that the spare resources are distributed among the messages in a strict priority order. This might be restrictive when, for example, it is desirable to do a more equitable distribution of the spare resources. In this case, the Elastic Task Model QoS manager is more adequate since it allows a tighter control over the way the spare resources are distributed.

According to the elastic model proposed in [4], the utilization of a task is treated as an elastic parameter, whose value can be modified by changing the period within a specified range. Each task is characterized by five parameters: a worst-case computation time C_i, a nominal period T_{i_0}, a minimum period $T_{i_{min}}$, a maximum period $T_{i_{max}}$, and an elastic coefficient E_i. Thus an elastic can be denoted by:

$$\tau_i(C_i, T_{i_0}, T_{i_{max}}, T_{i_{max}}, E_i)$$

The elastic coefficient specifies the flexibility of the task to vary its utilization for adapting the system to a new feasible rate configuration: the greater E_i, the more elastic the task. Thus, from a design perspective, elastic coefficients can be set equal to values which are inversely proportional to tasks' importance.

Admission of new tasks or requests of variations in the properties of existing ones are always subject to an elastic guarantee and are accepted only if there exists a feasible schedule in which all the other periods are within their range.

In [4] it is proposed to scheduled tasks by the Earliest Deadline First algorithm [11], hence, if $\sum \frac{C_i}{T_{i_{max}}} \leq 1$ the task set is schedulable.

Whenever a feasible schedule exists, if $\sum \frac{C_i}{T_{i_{min}}} \leq 1$, all tasks can be created at the minimum period $T_{i_{min}}$, otherwise the elastic algorithm is used to adapt the tasks' periods to T_i such that $\sum \frac{C_i}{T_i} = U_d \leq 1$, where U_d is some desired utilization factor. The elastic algorithm consists first in computing by how much the task set must be compressed ($U_0 - U_d$) and then to determine how much each task must contribute to this value, according to its elastic coefficient, as follows:

$$\forall i,\ T_i = T_{i_{min}} - (U_0 - U_d)\frac{E_i}{E_v} \tag{7}$$

where U_0 is sum of nominal task utilizations and $E_v = \sum_{i=1}^{n} E_i$.

However, due to the period constraints ($T_{i_{min}} \leq T_i \leq T_{i_{max}}$) the problem of finding the values T_i can require an iterative solution, since during compression one or more tasks may reach their maximum period. In this case the additional compression has to affect only to the remaining tasks. In [5] it is shown that, in the worst case, the compression algorithm converges to a solution (if there exists one) in $O(n^2)$ steps, where n is the number of tasks.

The Elastic Task Model was originally developed for task scheduling, which presents some important differences with respect to message scheduling. For example, tasks are usually preemptible while message transmissions cannot be suspended and resumed later. Moreover, in FTT periods, initial phasings and deadlines are expressed as integer multiples of the EC duration, while in the original elastic task model the resolution can be arbitrarily small.

Despite these differences, the elastic task model can be easily applied to the FTT paradigm, as presented by the authors in [14] for the FTT-Ethernet protocol, which is based on the FTT paradigm.

To cope with this framework the SRT (Definition 1) should be extended to incorporate the above referred properties.

$$Xf_i \equiv (T_{i_{min}}, T_{i_0}, T_{i_{max}}, E_i),\ i = 1..N_S \tag{8}$$

5. Case study: a mobile robot

5.1. Communication requirements

To illustrate the use of the FTT paradigm to provide dynamic QoS management, this section presents an hypothetical case study based on the requirements of a mobile robot that uses a distributed embedded control system. The robot should navigate autonomously within a delimited geographical area, and must exhibit the following behaviors: obstacle avoidance, path following and beacon tracking. The desired global robot behavior is determined by a subsumption architecture that arbitrates among the existing behaviors, deciding which is the active one. The behavior arbitration is carried out as follows: whenever an obstacle is detected, avoid it; in the absence of obstacle, follow a path indicated by a line on the floor; in the absence of obstacle and line, track a beacon and move towards it, otherwise move randomly.

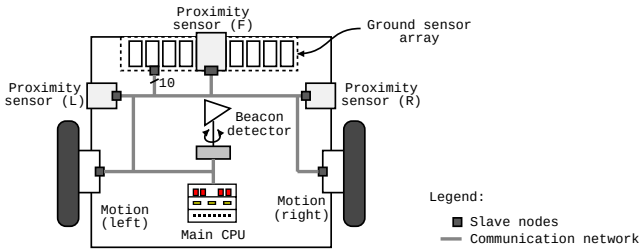

Figure 4. Robot components

Source	Signal name	Data Bytes	# of Mesgs	Period(ms) Min	Period(ms) Max
Obstacle sensors	OBST $_{1..3}$	1	3	10	50
Line sensors	LINE $_{1..10}$	1	10	10	1000
Beacon sensor	BCN_INT	1	1	200	2000
	BCN_ANG	1	1	50	200
Main CPU	SPEED $_{1..2}$	1	2	10	150
Motors	DISP $_{1..2}$	2	2	20	500

Table 1. Message set and properties

To support the desired behaviors the robot is equipped with two independent motors, a set of three proximity sensors to detect nearby objects, a beacon detector, a line sensor made of an array of 10 individual sensors and a main CPU to execute the high level control and planning software (Figure4). These elements are interconnected by a shared broadcast bus over which the FTT paradigm has been implemented. The FTT master is implemented in the main CPU, jointly with application tasks. The sensor readings are produced by the respective sensors and consumed by the main CPU. On the other hand, the main CPU produces the speed set-points that are consumed by the motor controllers, which execute closed-loop speed control. These controllers also produce displacement measures that are consumed by the main CPU to support trajectory control.

Table 1 characterizes the communication requirements, i.e. the message set and respective properties. Basically, each sensor will produce a 1-byte message with the respective reading except for the motor controllers that will produce a 2-byte message with the displacement information. The QoS requirements are expressed in terms of admissible ranges for the production rates of each message. Since specified periods are integer multiples of $10ms$, this value has been used to define the EC duration. Moreover, the synchronous window share was restricted to 80% of the EC duration. The remaining 20% were left for the trigger message as well as for possible asynchronous traffic, not defined here.

In order to derive tangible values, we assume an implementation over CAN [7], operating of 100Kbps. Table 2 shows the resulting minimum and maximum network utilizations when the minimum and maximum QoS requirements are used, respectively.

Considering that an EDF scheduler is used, and applying the analysis presented in Section 3.4, the upper bound for

Signal name	Tx time (μs)	# of mesgs	Period(EC)		Utilization(%)	
			Min	Max	Min	Max
OBST $_{1..3}$	650	3	1	5	3.90	19.50
LINE $_{1..10}$	650	10	1	100	0.65	65.00
BCN_INT	650	1	20	200	0.03	0.33
BCN_ANG	650	1	5	20	0.33	1.30
SPEED $_{1..2}$	650	2	1	15	0.87	13.00
DISP $_{1..2}$	750	2	2	50	0.26	6.50
Total utilization (%)					6.07	106.63

Table 2. Message set network utilization

guaranteed traffic schedulability is 73.5%. Recall that only 80% of the network bandwidth is available for synchronous traffic. This upper bound is well above the minimum required utilization but also well below the respective maximum requirement. This means that it is not possible to transmit all the messages at the respective highest rates but, on the other hand, if the lowest rates are used, there is a significant spare bandwidth. This gives room for QoS management in order to assign the spare bandwidth to specific message streams, increasing the respective QoS delivered to the application.

To better understand the use of dynamic QoS management, notice that the robot needs permanently updated information from all sensors but it executes only one behavior at a time (subsumption architecture). Therefore, the communication system should deliver the highest QoS to the active behavior, increasing the rate of the respective messages. Conversely, inhibited or latent behaviors, may be given lower QoS levels assigning lower transmission rates for the respective messages.

For instance, whenever the robot is following a line on the ground, line sensors should be sampled at the highest rate for accurate control. Obstacle detection must still be monitored in order to avoid possible obstacles near the line but, if no near obstacles are detected , lower sampling (transmission) rates can be used. Beacon detection is not relevant in this case. If a near obstacle is detected, the robot must switch the active behavior to obstacle avoidance, assigning highest QoS to this behavior and changing the transmission rates of the respective messages.

In the following sections we will show how the QoS management policies referred before can be applied to this case.

5.2. Using the priority-based QoS manager

In the case of priority-based QoS management, spare resources that remain after fulfilling the minimum resource requirements are distributed among the messages following an order of decreasing QoS priority. These priorities are message parameters that reflect the respective importance in the current robot state. In this dynamic situation, the QoS priorities must also be dynamic, deduced from the actual sensor readings and taking into consideration the referred hierarchy of behaviors as referred above.

In this particular case, a specific task running in the main CPU analyzes the received sensor readings, runs the behavior arbitration to define the active behavior and generates the QoS priorities. Whenever the relative priorities change, they

Signal name	Obstacle avoid.		Path follow.		Beacon tracking	
	QoS Priority	P_i	QoS Priority	P_i	QoS Priority	P_i
OBST$_{1..3}$	1	1	3	5	5	1
LINE$_{1..10}$	4	3	1	1	6	3
BCN_INT	4	20	5	20	4	20
BCN_ANG	4	5	5	9	1	5
SPEED$_{1..2}$	2	1	2	4	2	1
DISP$_{1..2}$	3	2	4	50	3	2
Utilization	63.29%		73.50%		63.29%	

Table 3. Message set utilization: priority-based QoS manager

are supplied to the QoS manager that calculates new effective message periods and applies them to the SRT in the FTT master structure. The rules to generate these QoS priorities are straight forward: the active behavior has highest one, the remaining behaviors are given priorities proportional to the excitation level of the respective sensors.Table3 shows the QoS priorities that were obtained in three different situations with three different active behaviors. The table also shows the results generated by the QoS manager, i.e. the granted transmission periods for each message, as well as the total bandwidth utilization. This utilization is always close to the maximum allowed (73.5% as referred before), meaning that the system is efficiently exploring its resources, i.e. network bandwidth in this case. The fact that the maximum utilization is not attained is due to the coarse time granularity used in the FTT paradigm (EC length), which causes step variations in the total utilization.

5.3. Using the Elastic Task Model QoS manager

As referred before, the Elastic Task Model uses two independent parameters per message[4], the nominal period and the elastic coefficient. The former ones allow to define the optimum periods within the allowable range. The latter ones define the flexibility given to the QoS manager to change the effective periods in the vicinity of the nominal ones. Again, in our case study we would like to adjust these parameters according to the instantaneous application needs or, in other words, according to the current sensor readings.

Therefore, a task running on the main CPU is also used to analyze the sensor readings, determine the active behavior and generate the QoS parameters. In this case, the generation of the parameters is done in the following way: for the active behavior, the nominal period of the respective messages is set to the minimum values, or close, and the elastic coefficient to one, or slightly higher, forcing a high QoS; for the remaining behaviors, the respective messages get a nominal period equal to the maximum values and the elastic coefficient is set proportionally to the respective sensor readings. In this latter case, when the excitation level of the sensors increases, the coefficients become larger thus increasing the chance of the respective behavior receiving higher QoS.

Signal name	Obstacle avoidance			Path following			Beacon tracking		
	T_{i_0}	E_i	P_i	T_{i_0}	E_i	P_i	T_{i_0}	E_i	P_i
$OBST_{1..3}$	1	1	1	5	10	1	5	5	1
$LINE_{1..10}$	100	8	3	1	1	2	50	20	2
BCN_INT	100	20	20	200	20	20	30	10	50
BCN_ANG	10	20	5	20	20	10	5	1	8
$SPEED_{1..2}$	1	1	1	2	1	1	1	1	1
$DISP_{1..2}$	4	5	2	10	10	2	2	5	2
Utilization	63.29%			73.48%			73.44%		

Table 4. Message set network utilization: ETM QoS manager

The QoS manager is invoked whenever there is a change on the elastic coefficients or nominal periods. However, to reduce the number of invocations and keep the run-time overhead under adequate levels, the mapping between sensor readings and elastic coefficients should be coarse, using large quantization steps. Moreover, it is important to use some level of hysteresis in order to prevent undesired oscillations in changing from step to step.

Table 4 also shows three situations in which the active behavior is different. The respective QoS parameters are shown together with the effective message periods generated by the QoS manager. The overall network utilization in all three situations is close but below the maximum possible (73.5% in this case). The reason is the same as explained in the case of the priority-based QoS manager, i.e. it is due to the coarse time resolution within the FTT paradigm.

6. Conclusions

This paper discussed the benefits and implications of supporting dynamic QoS management in distributed real-time systems, particularly in what concerns the communication network. The paper identified the need for flexible but still predictable communication services and showed that these requirements are not adequately supported in existing real-time communication protocols. Therefore, it proposed and discussed a new communication paradigm (Flexible Time-Triggered paradigm), which has been developed by the authors to address specifically this type of systems. Resulting from its operational flexibility, it is shown that this paradigm is particularly adequated for dynamic distributed real-time systems, which benefit from, or even require, dynamic QoS management. Moreover, the proposed paradigm supports arbitrary QoS management policies, as long as the QoS attributes can be mapped onto standard properties (periods, priorities or deadlines).

The FTT paradigm has been successfully implemented on CAN as well as on Ethernet, leading to the FTT-CAN and FTT-Ethernet protocols. Other implementations based on different networks are being studied. In particular, there is on-going work concerning the implementation over switched Ethernet as well as over the wireless LAN 802.11b.

To illustrate how the FTT paradigm supports dynamic QoS management, the paper also presents a simplified case study using a mobile autonomous robot. Two possible QoS management policies are briefly presented, one that is priority-based and the other based on the elastic task model, and it is shown how they can be used in the scope of the FTT paradigm. The results obtained confirm that using the FTT paradigm in distributed real-time applications can lead to efficiency gains in network bandwidth that arise from the support to dynamic QoS management policies.

References

[1] L. Almeida and J. Fonseca. Analysis of a simple model for non-preemptive blocking-free scheduling. In *Proceedings Euromicro Conference on Real-Time Systems RTS'01)*, Delft, Netherlands, June 2001.

[2] L. Almeida, P. Pedreiras, and J. A. Fonseca. The ftt-can protocol: Why and how. *IEEE Transactions on Industrial Electronics*, 49(6), December 2002.

[3] G. Buttazzo and L. Abeni. Adaptive rate control through elastic scheduling. In *39th IEEE Conference on Decision and Control*, Sydney, Australia, December 2000.

[4] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 286–295, Madrid, Spain, December 1998.

[5] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. In *IEEE Transactions on Computers*, volume 51 of *3*, pages 289–302, March 2002.

[6] J. A. S. et al. Strategic directions in realtime and embedded systems. *ACM Computing Surveys*, 28(4):751–763, 1996.

[7] R. B. GmbH. *CAN Specification version 2.0*. Robert Bosch GmbH, Stuttgart, Germany, 1991.

[8] H. Kopetz. Should responsive systems be event-triggered or time-triggered ? *IEICE Transactions on Information and Systems*, 11(76):1325–32, November 1993.

[9] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[10] C. Lee, R. Rajkumar, and C. Mercer. Experiences with processor reservation and dynamic qos in real-time mach. In *Multimedia Japan 96*, Japan, April 1996.

[11] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), 1973.

[12] P. Marti. *Analysis and Design of Real-Time Control Systems with Varying Control Timing Constraints*. PhD thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, July 2002.

[13] P. Pedreiras, L. Almeida, and P. Gai. The ftt-ethernet protocol: Merging flexibility, timeliness and efficiency. In *Proceedings of the 14th IEEE Euromicro Conference in Real-Time Systems*, Vienna, Austria, June 2002.

[14] P. Pedreiras, L. Almeida, P. Gai, and G. Buttazzo. Ftt-ethernet: A platform to implement the elastic task model over message streams. In *Proceedings of the WCFS '02*, Vasteras, Sweeden, August 2002.

[15] J.-P. Thomesse. The fieldbuses. In *Annual Reviews in Control*, volume 22, pages 35–45, 1998.

[16] J.-P. Thomesse and M. L. Chavez. Main paradigms as a basis for current fieldbus concepts. In *Proceedings of FeT'99 (International Conference on Fieldbus Technology)*, Magdeburg, Germany, September 1999.