

Using RT-CORBA Scheduling Service And Prioritized Network Traffic to Achieve End-to-End Predictability

Tarek GUESMI
Laboratoire SYSCOM
Ecole Nationale d'ingénieurs de
Tunis
1002 Tunis, Tunisia
Tarek.guesmi@isecs.rnu.tn

Salem HASNAOUI
Laboratoire SYSCOM
Ecole Nationale d'ingénieurs de
Tunis
1002 Tunis, Tunisia
Salem.hasnaoui@enit.rnu.tn

Houria REZIG
Laboratoire SYSCOM
Ecole Nationale d'ingénieurs de
Tunis
1002 Tunis, Tunisia
Houria.rezig@enit.rnu.tn

Abstract— *Computing systems are increasingly distributed, real-time, and embedded (DRE) and must operate under highly unpredictable and changeable conditions. To provide predictable mission-critical quality of service (QoS) end-to-end, QoS-enabled middleware services and mechanisms have begun to emerge. It is also, widely known that Control Area Networks (CAN) are used in real-time, distributed and parallel processing which cover manufacture plants, humanoid robots, networking fields. We show how prioritization of messages over the physical CAN network can be achieved, when adopting the use of RT-CORBA distributed scheduling service which implements a dynamic scheduling policy to achieve end-to-end predictability and performance.*

Keywords: Real-time scheduling, Earliest Deadline First, CAN Bus, Intermediate Deadline, Network Priority Mapping.

I. Introduction

In recent years, there has been a growth in a category of performance-critical distributed systems executing in open and unpredictable environments [10]. Examples range from next generation military avionics and ship computing systems to current open systems. In these systems, intelligent sensors, actuators and distributed control structures replace the centralized computer. This leads to a modular system architecture in which smart autonomous objects cooperate to control a physical process. As theory and practice in distributed computing and in real-time computing matures, there is an increasing demand for automated solutions for dynamic distributed real-time middleware to support scheduling end-to-end timing constraints. The latest version of Real-Time CORBA (RTC), known as RTC1.2 (formerly known as RTC 2.0) [1], defines the Distributable Thread (DT) primitive to support real-time computing in dynamic distributed middleware systems. RTC1.2 provides a flexible means for expressing and propagating scheduling information across node boundaries in a distributed system. However,

Real-Time CORBA is not immediately applicable to embedded real-time control systems for several reasons:

- Real-Time CORBA implementations have excessive resource demands. A first step to solve this problem is the *minimumCORBA* [12] specification, which is a cut-down version of CORBA specified by the OMG.
- Often Real-Time CORBA implementations are built on the top of unpredictable off-the-shelf soft- and hardware and do not support typical real-time communication systems.

A real-time communication system (RTCS) constitutes the backbone for distributed control applications. RTCS substantially differ in many respects from general purpose communication systems. In general, while the goals of general purpose communication systems center around throughput, RTCS focus on predictability of communication. Predictability means that the system exhibits an anticipated behaviour in the functional and the temporal domain. Controller Area Network (CAN) bus [15] provides advanced built-in features, which make it suitable for complex real-time applications. Some of these features are priority-based, multiparty bus access control using carrier sense / multiple access with collision avoidance (CSMA/CA), bounded message length, efficient implementation of positive/negative acknowledgement, and automatic fail-silence enforcement with different fault levels. These characteristics make it very challenging to run Real-Time CORBA applications on a CAN-based distributed platform. To exploit the advantages of CAN for RT-CORBA, we designed an inter-ORB protocol within the context of Data Acquisition from Industrial Systems (DAIS) use [2]. RT-CORBA preserves end-to-end priorities by the mapping of the importance of the Distributable Thread to the corresponding operating system priorities and propagating these priorities across the network as the DT spans multiple hosts; however RT-CORBA specification is less explicit about the communication transport and the

underlying network. A promising approach is QoS enhancement by preserving the priority of the client when sending a request and accessing the communication support by giving an efficient mechanism to map the DT global priority, assigned by the distributed scheduling service and the CAN network-based priority. Contributions of this paper are as follows:

- Describing the interaction between the Distributable Thread (DT) and distributed scheduling service (DSS) when sending a request or a reply.
- Developing an efficient model using tasks and subtasks to describe the execution of the DT over the distributed system and especially the communication over the CAN bus.
- Implementing a new technique for calculating the deadline of the CAN message, and hence its priority.

The remainder of this paper is organized as follows: section 2 describes the related work; section 3 summarizes the technical backgrounds of this work and describes basic principles of Real-Time CORBA, Controller Area Network and some related real-time basic knowledge. Section 4 describes the proposed architecture and mechanism for supporting the network priority mapping. In section 5, we evaluate the latency time introduced by the CAN network transmission. Some general assessments of the lessons learned are provided and some conclusions are drawn in section 6.

II. Related work

Developing Distributed Real-Time Embedded (DRE) platforms based on RT-CORBA middleware running over real-time networks is a very challenging research topic and during the last years, several teams have prominently worked on these platforms. One of these is a team of the Software Architecture Lab, Seoul National University headed by Kimoon Kim [6] [7]. In their paper [6] Kim and others present their design of CAN-CORBA, an environment specific CORBA for CAN-based distributed control systems. Their ORB core supports the classical connection-oriented point-to-point communication of CORBA and additionally subscription-based group communication. They implement a new Inter-ORB protocol customized for the CAN bus called Embedded Inter-ORB Protocol (EIOP). Nevertheless EIOP was first inter-ORB protocol designed for embedded systems, it provides no support for Real-Time CORBA specifications and therefore it provides not translation between the priority handling of CAN and Real-Time CORBA.

In their paper [4], S. Lankes, A. Jabs and T. Bemmerl describe the implementation of a CAN-based connection-oriented point-to-point communication model and its integration into Real-Time CORBA in the context of ROFES [13] platform. The main idea presented in their work is to make efficient use of the advantages of CAN by means of smaller message headers for CAN and mapping the CAN priorities to a band of RT-CORBA priorities.

However the idea of mapping priorities in both level applications and network is very interesting and fundamental to enforce end-to-end predictability, the way this mapping is done in ROFES is too simple and this for many reasons. First, the protocol is based on CAN 2.0A and thus only uses 11-bit CAN identifiers, and the priority field is encoded in 2 bits. This choice imposes severe restrictions of the number of network priorities comparing to the number of CORBA priorities. Second, the technique used for priority mapping, maps each individual CAN priority to a contiguous range of CORBA priorities and thus badly expresses the real-time requirements of each DT when communicating over the CAN bus.

Recently, Douglas C. Schmidt and al. [8] present an interesting approach, which describes how priority and network QoS management mechanism can be coupled with standards-based, off-the-shelf distributed object computing (DOC) middleware to better support dynamic DRE applications with stringent end-to-end real-time requirements. This work is very interesting since it provides TAO [5] extensions to support mechanisms to map RT-CORBA priorities to DiffServ network priorities. This enhancement allows RT-CORBA middleware to manage network resources using DiffServ, which is priority based but provides no support to other prioritized network traffic like CAN-based networks.

The main idea discussed within this work is how to use a task model to calculate CAN message priority relatively to the global CORBA priority and thus by defining a well adapted architecture using the distributed RT-CORBA Scheduling Service.

III. Technical Backgrounds

A. Basic CAN Features

The Controller Area Network (CAN) is an ISO defined serial communication bus. It was originally developed during the 80's by the Robert Bosch GmbH for the automotive industry. The CAN bus works according to the Producer-Consumer-Principle: messages are not sent to a specific destination address, but rather as a broadcast (aimed at all receivers) or a multicast (aimed at a group of receivers). A CAN message has a unique identifier, which is used by devices connected to the CAN bus to decide whether to process or ignore the incoming message. Two variants of the CAN protocol exist. The main difference between the first (CAN 2.0A) and second variant (CAN 2.0B) is that the former uses 11 bits to uniquely identify each message, while the latter uses 29 bit identifiers. For correct operation of the CAN bus, the identifiers of two messages sent at the same time must never be the same, consequently CAN 2.0B offers a greater variety and scope for concurrent message Id's.

The CAN bus is based on the arbitration scheme Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA). During arbitration process, any node willing to send a CAN message starts sending bit by bit the 11 or (in case of CAN 2.0B) 29 identifier bits. Each time a bit is applied to the bus, the sending node checks whether the bus really is at the corresponding voltage level—high for an applied logical 1 and low for an applied logical 0. As a

common resource, the CAN bus has to be shared by all computing nodes. Access to the bus has to be scheduled in a way that distributed computations meet their deadlines in spite of competition for the communication line. Since the scheduling of the bus cannot be based on local decisions, a distributed consensus about the bus access has to be achieved. The CSMA/CA protocol is comparable with a priority-based dispatcher. Due to this analogy, it is possible to express scheduling decisions for the CAN-bus resource by dynamic priority orders. The presented approach associates advantage of the built-in CSMA/CA access protocol of CAN bus and the Real-Time CORBA dynamic scheduling service to realize the EDF access regulation.

B. Basic Real-Time CORBA 1.2 Features

To understand the model presented within the context of this paper, which associates the scheduling mechanisms of both CAN bus and Real-Time CORBA scheduling service, this section explains the necessary features of the Real-Time CORBA specification. A more detailed description of the Real-Time CORBA specification is given in [3] and [11]. Real-Time CORBA is a QoS enabled extension of CORBA middleware. The Real-Time 1.1 specification is designed for static distributed system where the number of tasks and their scheduling parameters are known a priori. The Real-Time CORBA 1.2 specification extends RTC1.1 to encompass both static and dynamic systems. In a dynamic system, tasks enter and leave the system at times that cannot be calculated a priori. In order to effectively manage the dynamic task set, RTC1.2 introduces the Distributable Thread scheduling primitive. A DT is an abstraction of a chain of method calls by multiple threads at multiple processors. According to its definition, a DT can span nodes boundary and carry scheduling parameters to each node in the chain. At each local node, the correspondent local scheduler will schedule that DT based on its scheduling information. Each distributable thread in RTC2 is identified by a unique system wide identifier called a *Globally Unique Id (GUID)*. A distributable thread may have one or more execution scheduling parameters, e.g., priority, time-constraints (such as deadlines), and importance. These parameters specify the acceptable end-to-end timeliness for completing the sequential execution of operations in CORBA object instances that may reside on multiple physical endsystems. Below we describe the key interfaces and properties of distributable threads in the RTC2 specifications:

Scheduling segment. A distributable thread comprises one or more *scheduling segments*. A scheduling segment is a code sequence whose execution is scheduled according to a distinct set of scheduling parameters specified by the application. For example, the worst-case execution time, deadline, and criticality.

Scheduling points. An application and ORB interact with the RTC2 dynamic scheduler at pre-defined points to schedule distributable threads in a DRE system. These scheduling points allow an application and ORB to provide the RTC2 dynamic scheduler information about the competing tasks in the system, so it can make scheduling decisions in a consistent and predictable manner. Scheduling points 1-3 in Figure 1 are points where an application interacts with the RTC2 dynamic scheduler.

Scheduling points 4-7 are points where an ORB interacts with the RTC2 dynamic scheduler, i.e., when remote invocations are made between different hosts. The ORB interacts with the RTC2 dynamic scheduler at points where the remote operation invocations are sent and received. Client-side and server-side interceptors are therefore installed to allow interception requests as they are sent and received.

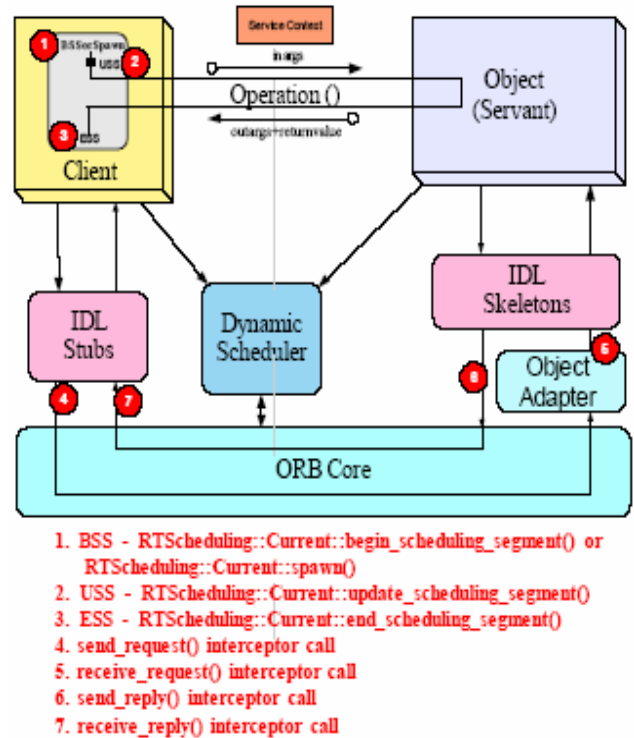


Figure 1. RTC2 Scheduling Points

As depicted in figure 1, the distributable thread interacts with the RTC 1.2 dynamic scheduler, which responsible for the allocation of CPU resources, to meet the QoS needs of the application that share ORB endsystem. The problem with such architecture, as we can notice here, that all scheduling decisions are assumed to be local – that is a local scheduler on each endsystem uses the same propagated scheduling information to make local scheduling decisions. These local schedulers do not have a global view of the overall system. This could lead to local enforcement decisions that fail to achieve maximum possible global system performance. Another problem is that the RTC1.2 dynamic scheduler is not able to regulate the access to communication support when sending request or reply. These problems can be solved when using Distributed Scheduling Service (DSS) framework that works with application specified end-to-end scheduling parameters and with local scheduling mechanisms to make globally sound scheduling decisions for the system. To manage network access, an RTC1.2 framework must resolve a number of design challenges. Below we examine two challenges:

- Designing interactions between DTs and DSS when sending a message on the CAN bus.
- Determining the message priority using a mapping from global CORBA priority of the DT to CAN priority.

IV. Proposed architecture

A. Task Model

In this section, we define the manner the distributable thread interacts with the distributed scheduling service when sending a request or a reply, and thus to improve the network behavior. The problem here is, how can the DSS set the network priority of the message sent over the CAN bus using the scheduling parameters of the DT. To solve this problem, we are brought to develop a tasks and subtasks model that describes the execution of the distributable thread over the distributable system. In our model the distributable thread DT is modeled with real-time Task (T), the scheduling parameter elements of the distributable thread represent real-time attributes of the task. Two scheduling parameter elements will be taken in consideration in the model: 1) distributable thread deadline and 2) distributable thread execution time. At any given instant, each distributable thread has only one execution point in the whole system, i.e., it executes a code sequence consisting of nested distributed and/or local operation invocations. This code sequence executing in one node of the distributed system is mapped to a subtask of the original task mapped to the distributable thread. We believe that this model encompasses the communication subsystem, i.e., the CAN bus. That is sending messages can be viewed as another type of subtask. For instance, let T be a task composed of two subtasks T_1 (running on node N_1) and T_2 (running on node N_2), say that after T_1 completes, it is necessary to send a message from N_1 to N_2 (*send_request()*) containing the inputs for T_2 . Then after T_2 finishes it is necessary to ship the final result to some other site (*send_reply()*). The two transmissions can be seen as two additional subtasks, T_a, T_b , so that the global task is really $T=T_1, T_a, T_2, T_b$. Below we describe the basic formalism of our task model. A distributed real-time system consists of several nodes representing system components. Each node manages one or more resources, for example, a database, a cycle server, or a communication channel. At each node, there is a real-time scheduler prioritizing tasks according to some real time queueing discipline, e.g., earliest deadline first (EDF).

We consider two categories of tasks: local and global. A task that visits only one node and is submitted to a scheduler only once is termed a local task. On the other hand, a global task is a set of several related subtasks or stages, to be executed in series. The deadline of a global task is the time by which the last subtask must complete. Each subtask is associated with an execution node to which it is submitted for execution. Associated with each task X (whether it is local, global, or a subtask) are five attributes denoted by the following functions:

- $ar(X)$ = arrival (or submission) time of X,
- $sl(X)$ = slack of X,
- $dl(X)$ = deadline of X,
- $ex(X)$ = real execution time of X,
- $pex(X)$ = predicted execution time of X.

The first four attributes are related by the following equation:

$$dl(X) = ar(X) + ex(X) + sl(X) \quad (1)$$

We assume that the deadline and the execution time of the task are known, since they are member of the scheduling parameters elements, introduced with the distributable thread mapped to that task. The slack can be computed using the above equation. The section will introduce the main components interacting when sending a message on the CAN bus and how the former model can be useful to evaluate the network priority of this message.

B. System Architecture

Figure 2 depicts our overall system architecture. There are five essential components in this framework, Distributed Thread (DT), Local Scheduler, DSS, Local Info Collection and System Info Repository. These components are independent and coordinated with each other.

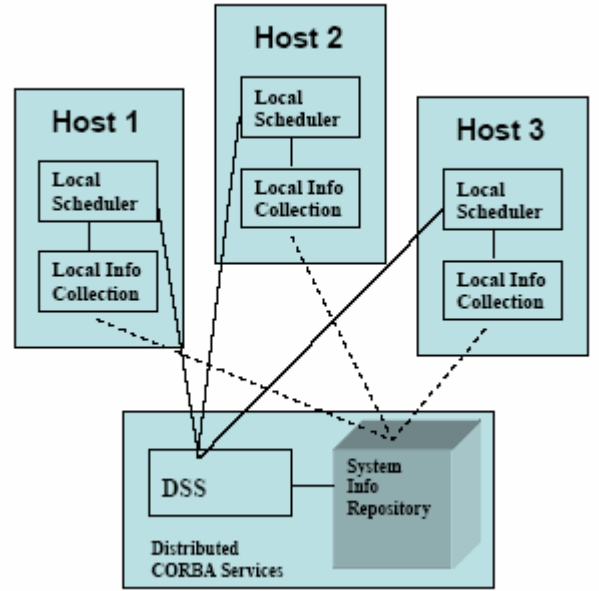


Figure 2. System Architecture

A *Distributable Thread* (DT) is the schedulable entity in our system architecture. When a DT is spawned by the application it carries its specified scheduling parameters, including its end-to-end deadline, along as it traverses the nodes in its path. The Local Scheduler is defined in RTC1.2 to manage the local portion of a DT. In our architecture, we extend the definition and allow the local scheduler to interact with the DT so that the local scheduler can obtain and use global information. When a DT is spawned by an application, the DT communicates with the DSS to determine if it is schedulable alongside the existing DTs in the system, and receives from the DSS its globally sound scheduling parameters. In RTC1.2, the interface specifies that the DT pass its scheduling parameters to the Local Scheduler. We have preserved this interface, and extended the Local Scheduler to allow it to send these parameters to the DSS, which returns the globally sound scheduling parameters to be returned to the DT. A DT first sends its scheduling information to a local scheduler whenever the DT makes a request to begin, update or end a scheduling segment. The parameters passed along are determined by the scheduling discipline chosen by both DT and the local scheduler such as RM, DM, EDF and

MUF. If the DT spans multiple nodes our design dictates that it must pass an end-to-end deadline and a sequence of subtasks to the local scheduler. On each endsystem, a local scheduling component schedules access to resources within that endsystem, and a local information collection component records a variety of status information such as CPU utilization, progress of application activities, and success or failure of tasks in meeting their deadlines. This local status information is distilled into higher-level information such as predictability of local tasks in meeting intermediate deadlines toward timely completion of end-to-end activities. The higher level information is sent to a distributed information collection service called the *system information repository*.

The components cited above cooperate to enforce the end-to-end task scheduling. The DSS sets intermediate deadlines for an EDF local scheduler; it uses the end-to-end deadline and subtask execution times to calculate an intermediate relative deadline for each subtask.

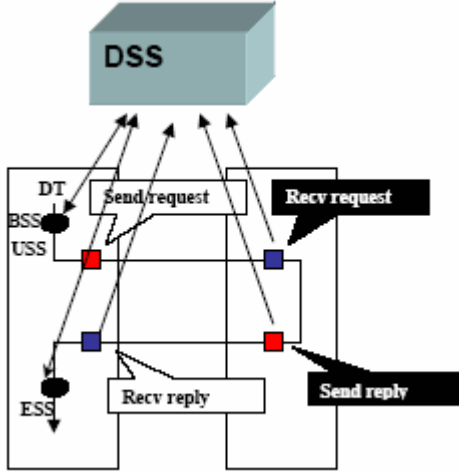


Figure 3. RTC1.2 Scheduling Points and DSS

In RTC1.2 scheduling points are the points in time and/or code at which the local scheduler is invoked by the application, which may result in a change in the current schedule. Figure 3 shows that all seven scheduling points which may have interactions with our DSS. The seven scheduling points are Begin_Scheduling_Segment (BSS), Update_Scheduling_Segment (USS), End_Scheduling_Segment (ESS), *send_request*, *receive_request*, *send_reply*, and *receive_reply*. In our current implementation, we use two of the seven scheduling points, *send_request* and *send_reply* to interact with the DSS, to calculate request and reply CAN priority and thus to optimize the network behaviour. At the *send_request* scheduling point, the DT sends all of its scheduling parameters to the DSS. These scheduling parameters include a system wide unique name of the DT, its execution time and its end-to-end deadline. The question here is how to set the network priority for the CAN message send when calling the *send_request()*. The DSS, using the task model we introduced previously, calculates the intermediate deadline of the network message and then mapped this deadline to a CAN priority.

The figure below depicts the way the distributable thread, the DSS and the system repository interact in order

to set the intermediate deadline of the message and thus its priority.

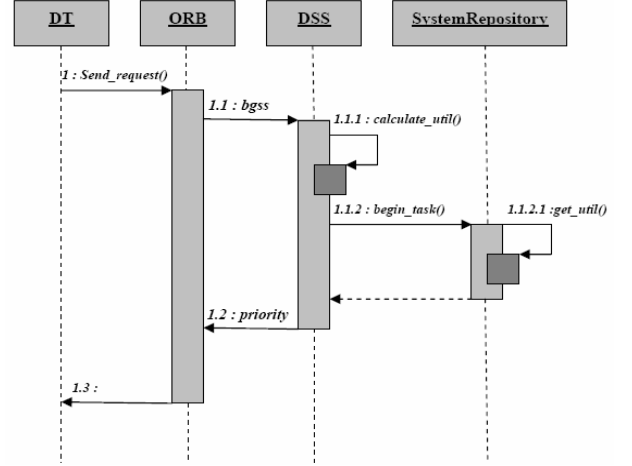


Figure 4. Scheduling Point – *send_request()*

C. Setting the network priority

As we indicated above, the CSMA/CA access protocol, used to regulate the access to the CAN bus, is comparable with a priority based dispatcher. Due to this analogy optimal scheduling of soft real-time communication can be achieved by EDF scheduling strategy. The first step done by the DSS is to calculate the message transmission deadline using the subtask deadline assignment [9]. The second is to map this deadline into the message priority.

1) Transmission deadline assignment

We consider the distributable thread as a global task T that consists of n subtasks T_1, \dots, T_n . The message transmission on the CAN bus can be seen as a subtask T_i and its deadline can be calculated using the Equal Slack strategy (EQS) [9]. Using this strategy, each subtask (including message transmission) should have its fair share of its global task's slack and this can be done when dividing the total remaining slack equally among the remaining subtasks:

$$dl(T_i) = ar(T_i) + pex(T_i) + \left[dl(T) - ar(T_i) - \sum_{j=1}^n pex(T_j) \right] / (n - i + 1) \quad (2)$$

- $ar(T_i)$: arrival time of T_i , i.e., the point of time when the ORB makes the *send_request()* or *send_reply()*.
- $pex(T_i)$: predictable execution time of T_i , corresponds to the time taken by the message transmission over the CAN bus. $pex(T_i)$ can be assumed to the longest time taken to transmit message m (C_m), based on bounding the number of bits sent on the bus for this message. For CAN networks we have the fellow expressions:

$$C_m = \left\{ \left(\frac{34 + 8S_m}{4} \right) + 47 + 8S_m \right\} \tau_{bit} \quad (3) \quad \text{For CAN 2.A and}$$

$$C_m = \left\{ \left(\frac{54 + 8S_m}{4} \right) + 57 + 8S_m \right\} \tau_{bit} \quad (4) \quad \text{for CAN 2.B.}$$

The term S_m is the number of bytes in payload field of the message and τ_{bit} is the bit time of the bus (i.e. $1\mu s$ at a bus speed of 1 MBPS). This time delay includes the 47 bit overhead per message and 34 bits of the overhead added to the message content, both are subjected to bit stuffing. Recall that the stuffing consists on an additional bit of opposite value added after 5 successive bits of identical value. The same reasoning can be made for CAN 2.B.

- $dl(T)$: Corresponds to the global deadline of the task, i.e., the distributable thread.

- $\sum_{j=i}^n pex(T_j)$: The predictable remaining execution time of the task. This expression cannot be directly evaluated. To solve this problem we propose to deduce it from the current execution time of the global task and its global execution time. As indicated above, the local information collection component records the CPU utilization for the subtasks on each node, these information are sent to the system information repository, witch aggregate the subtasks execution times for each global task to get its current execution time when T_i is submitted ($curr_ex(T, ar(T_i))$).

We implement `SystemRepository::get_utilization()` to evaluate this variable. The predictable remaining execution time of the task can be written:

$$\sum_{j=i}^n pex(T_j) = ex(T) - [curr(T, ar(T_i))] \quad (5)$$

After evaluating all terms of equation (2), the DSS calculates the intermediate deadline of subtask T_i , which corresponds to the message transmission deadline. Below we describe how to map this deadline to CAN message priority.

2) Priority Mapping

In this section, we describe how the message transmission deadline is mapped to real-time message priority on the CAN network.

2-bit Protocol Type	6-bit Priority Level	7-bit Physical Node_ID	6-bit TX_Port Number	8-bit TX_CAN Object_ID
---------------------------	----------------------------	------------------------------	----------------------------	------------------------------

Figure 5. Partitioning of a CAN-message identifier

As depicted in figure 5, the CAN-ID is divided into five fields. In [14] we described deeply the partitioning of the CAN-ID, in this paper we are specifically interested in the priority level field. In the priority field of a real-time message, the time remaining until its transmission deadline is encoded. The *transmission deadline* is a point of time specified by the sending application object, when a message must be completely transmitted to receiving nodes. As long as a sending node is pending for the bus, its

communication subsystem checks and updates the transmission deadline of the ready message periodically.

Each value of the transmission laxity is mapped to a portion of future time, a *priority tick* Δt_p . At the end of each priority tick, a pending transmitter increases the priority of its real-time message by decrementing its transmission deadline field. The priority ticks are time intervals of a fixed length, with the first one beginning at the present time. Since there are only a limited number of different priorities ($2^6=64$ priority levels), only a limited number of priority ticks are visible. Now the question is: how to map the deadline transmission message, calculated before, to its CAN priority?

Having a range $\{P_{min} .. P_{max}\}$ for the priority field, a deadline ΔL is mapped to a priority P , where $P = \lfloor \Delta L / \Delta t_p \rfloor + P_{min}$ if $\Delta L < (P_{max} - P_{min}) * \Delta t_p$ and $P = P_{max}$ if $\Delta L \geq (P_{max} - P_{min}) * \Delta t_p$. The period Δt_p is called the priority slot and:

- P_{min} is the highest priority = lowest binary value for real-time priorities.
- P_{max} is the lowest priority = highest binary value for real-time priorities.

We denote P as the network priority of the message and network priority mapping, the function having, and global CORBA priority of the distributable thread as input and the network priority as output.

v. The network Latency-time Evaluation Methodology

The performance evaluation of the latency time is based on the evaluation of both the *worst-case queuing delay* in the CAN MAC and physical sublayers and the longest time needed to transmit a message. We ignore factors with low probabilities that affect the latency time, such as message retransmission at Upper Layer Protocols-ULP, operating system scheduling uncertainties, etc. We limit the study of the latency time only to the CAN physical and MAC sublayers. Let J the queuing jitter of a message from the ULP to medium access control layer. We assume this jitter J is constant. The analysis of the worst-case response time can be derived from tasks scheduling theory and real-time scheduling algorithms that can be applied by the transmitter and by the receiver to guarantee a minimum latency time for the exchanged frames.

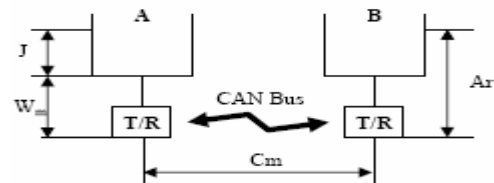


Figure 6. Definition of Latency time

The latency time is defined as the difference of time between the instant indicating the beginning of the transmission request and the real beginning of the action generated by this one.

Let W_m be the time taken by a station to gain access to the medium, C_{mes} the longest time needed for transmitting successfully a CAN message and Ar the time taken by a receiver station to analyze the transmitted MAC frame.

$$T_{lat} = J + W_m + C_{mes} + Ar \quad (6)$$

In what follows, we assume that the jitter J and Ar have constant values. Knowing that C_m is the longest time taken to successfully transmit a CAN messages, then we have:

$$C_m = \left\{ \left(\frac{54 + 8S_m}{4} \right) + 57 + 8S_m \right\} \tau_{bit}$$

W_m is the worst-case queuing delay of message m due to both higher priority messages pre-empting message m , and a lower priority message that has already acquired the bus [16].

$$W_m = B_m + \sum_{\forall j \in l(m)} \left[\frac{W_j + J_j + \tau_{bit}}{T_j} \right] C_j \quad (7)$$

Where:

$$B_m = \max_{k \in l(m)} (C_k) \quad (8)$$

Finally, we obtain the latency time expression, introduced by the CAN physical and MAC layer as:

$$T_{lat} = J + B_m + \sum_{\forall j \in l(m)} \left[\frac{W_j + J_j + \tau_{bit}}{T_j} \right] C_j + \left\{ \left(\frac{54 + 8S_m}{4} \right) + 57 + 8S_m \right\} \tau_{bit} + Ar \quad (9)$$

VI. Conclusion

With the work described herein, the CAN bus has been rendered more usable in the field of distributed real-time systems. We tried to design interaction between distributable thread and distributed scheduling service to optimise network behaviour. This interaction aims to integrate the network resources control to high level middleware and thus enabling a new generation of flexible DRE applications that have more precise control over their end-to-end resource. The priority based mechanism and the EDF scheduling strategy used within the context of this work is adapted with soft real-time communication systems.

On promising research direction is to combine priority-based mechanisms in conjunction with reservation mechanisms and to combine this strategy with the hybrid real-time bus scheduling mechanisms for CAN.

VII. References

- [1] Real-time CORBA (Dynamic Scheduling) specification, version 1.2, Nov., 2003, <http://www.omg.org/cgi-bin/apps/doc?formal/05-01-04.pdf>.
- [2] OMG Technical Document – Manufacturing Domain Task/01-09-03. *Data Acquisition from Industrial Systems Specification*, 2002.
- [3] D. C. Schmidt, D. L. Levine, and S. Mungee, “The Design and Performance of Real-Time Object Request Brokers,” *Computer Communications*, vol. 21, pp. 294–324, Apr. 1998.
- [4] S. Lankes, A. Jabs, and T. Bemmerl. Integration of a CAN-based Connection-oriented Communication Model into Real-Time CORBA. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003), 11th Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2003), Nice, France, April 2003.
- [5] D. C. Schmidt, D. L. Levine, and S. Mungee, “The Design and Performance of Real-Time Object Request Brokers,” *Computer Communications*, vol. 21, pp. 294–324, Apr. 1998.
- [6] T.Kim, K.Kim, G.Jeon and S.Hong, “Integration subscription-based and connection-oriented communication into the embedded CORBA for the CAN Bus,” *IEEE Real-Time Technology and Application Symposium, Washington D.C., USA, May 2000*.
- [7] T.Kim, K.Kim, G.Jeon and S.Hong., “Resource-conscious customization of CORBA for CAN-based distributed embedded systems,” *IEEE International Symposium on Object-Oriented Real-Time Computing, Newport Beach, CA, USA, March 2000*.
- [8] D. C. Schmidt and C. O’Ryan. Patterns and Performance of Distributed Real-time and Embedded Publisher/Subscriber Architectures. *Journal of Systems and Software*, 2002.
- [9] B. Kao,H. Garcia-Molina: “Deadline Assignment in a Distributed Soft Real-Time System,” in the proceedings of the 13th International Conference on Distributed computing Systems. 1993.
- [10] G. Blair, G. Coulson, P. Robin, M. Papatomas, “An Architecture for Next Generation Middleware,” Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, London, England, 1998.
- [11] Tejasvi Aswathanarayana, Venkita Subramonian, Deepti Mokkaoti, Harihar Subramanian, Douglas Niehaus, and Christopher Gill, “Design and Performance of Configurable Endsystem Scheduling Mechanisms”, submitted to the 11th IEEE Real-time Technology and Application Symposium (RTAS), San Francisco, CA, March, 2005.
- [12] OMG Technical Document orbos/98-08-04, minimumCORBA – Joint Submission , 1998
- [13] S. Lankes, M. Pfeiffer, and T. Bemmerl. Design and Implementation of a SCI-based Real-Time CORBA. In 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001), Magdeburg, Germany, May 2001.
- [14]] T. Guesmi, S. Hasnaoui, Design and Implementation of a CAN-based Inter-ORB Protocol, Using RT-CORBA, Data Acquisition from Industrial Systems and the underlying Real-Time Control Area Network, In *PDPTA’05 Int. Conf.* Las Vegas, USA, June 2005.
- [15] ROBERT BOSCH GmbH, *CAN Specification Version 2.0*, 1991.
- [16]] S. Hasnaoui, O. Kallel, “A proof of concept implementation of a proposed modification of CAN protocol on a FPGA controller component,”. *AMI Journal*, in press.