Design and Performance of DDS-based Middleware for Real-Time Control Systems

Tarek Guesmi, Rojdi Rekik, Salem Hasnaoui and Houria Rezig

SysCom Laboratory, National School of Engineering of Tunis, Tunisia

Summary
Data-centric design is emerging as a key tenet for building advanced data-critical distributed real-time and embedded systems. These systems must find the right data, know where to send it, and deliver it to the right place at the right time. Data Distribution Service (DDS) specifies an API designed for enabling real-time data distribution and is well suited for such complex distributed systems and QoS-enabled applications. It is also, widely known that Control Area Networks (CAN) are used in real-time, distributed and parallel processing.
Thus, the goal idea of this paper is to study an implementation of publish-subscribe messaging middleware that supports the DDS specifications and that is customized for real-time networking. This implementation introduces an efficient approach of data temporal consistency and real-time network-scheduler that schedules network traffic based upon DDS QoS-policies. A simulator has been developed to demonstrate that our implementation fulfills the guarantees predicted by the theoretical results.

Key words:
Publish-Subscribe, data distribution, Real-Time Middleware, CAN Bus, DDS, EDF, Distributed Control Systems.

1. Introduction

In recent years, there has been a growth in a category of performance-critical distributed systems and applications executing in open and unpredictable environments [1]. Examples range from next generation military avionics and ship computing systems to current open systems. In these applications, data produced in one component of the system needs to be shared with other components of the system. Such applications may have stringent deadlines by which the data must be delivered in order to process it on time to make critical decisions. Further, the data that is distributed must be valid when it arrives at its target. That is, if the data is too old when it is delivered, it could produce invalid results when used in computations. A simple solution would be to provide point-to-point or client-server communication to deliver data within the real-time system. However, this communication can become extremely complex when multiple components require the same data at differing rates. Furthermore the communication infrastructure is inflexible. A decoupled solution where publishers of data do not communicate directly with subscribers (consumers) of data is more efficient and flexible. Such a solution would allow the publishers of data to produce the data at a rate that is consistent with the data production, and would allow subscribers (consumers) of the data to receive the data at a rate that consistent with the needs of the application. As theory and practice in distributed real-time computing, data-centric publish-subscribe (DCPS) and middleware networking mature, there is an increasing demand for automated solutions in real-time DCPS middleware to support scheduling end-to-end timing constraints.

Data-centric design is key to systems which exhibit some or all of the following five characteristics: (a) participants are distributed; (b) interactions between participants are data-centric and not object-centric; often these can be viewed as “dataflows” that may carry information about identifiable data-objects; (c) data is critical because of large volumes, predictable delivery requirements and the dynamic nature of the entities; (d) computation is time sensitive and may be critically dependent on the predictable delivery of data, (e) storage is local. In a large class of data-centric systems, real-time availability of information is of utmost importance. Information generated from multiple sources must be distributed and made available to 'interested parties' taking into account Quality of Service (QoS) offerings by information-producers and requests by information-consumers. The problem to solve can be stated as: ‘How to get the right data at the right time at the right place in real-time and mission-critical systems’. In order to address this need the Object Management Group (OMG) defined the Data Distribution Service (DDS) [2] specification which provides a set of profiles that target real-time information availability for domains ranging from small-scale embedded control systems up to large-scale enterprise information management systems.

Data Distribution Service is a popular standard in embedded systems. It uses publish-subscribe communication model, and supports both messaging and data-object centric data models. DDS targets real-time systems; the API and Quality of Service (QoS) are chosen to balance predictable behavior and implementation efficiency/performance. One of the promising approaches is to make an efficient use of QoS mechanisms proposed by the DDS specification when adopting real-time network

Manuscript received December 5, 2007
Manuscript revised December 20, 2007

messaging.
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 on throughput, RTCS focus on predictability of communication. Controller Area Network (CAN) bus [3] provides advanced built-in features, which make it particularly suited to implement a publisher-subscriber model of communication. Some of these features are priority-based, multiparty bus access control using carrier sense multiple access with Arbitration on Message Priority (CSMA/AMP), bounded message length, message filtering, efficient implementation of positive/negative acknowledgment, and automatic fail-silence enforcement with different fault levels. These characteristics make it very challenging to run real-time data-centric publish-subscribe applications.
DDS provides the DEADLINE QoS Policy, LATENCY_BUDGET Qos Policy, TRANSPORT_PRIORITY QoS Policy and other policies specifically targeted to minimum latency, predictable real-time operation in high-performance distributed data-critical systems. However DDS specification is less explicit about the scheduling mechanisms that should be used to coordinate these policies and to make best benefit when exploiting the underlying facilities of the real-time network. This paper presents a solution to the data distribution when adopting the real-time communication systems. The solution consists on the design of new comprehensive scheduling strategy that provides:

- an algorithm that determines scheduling parameters to ensure that data that is delivered will be valid when it is used.
- a unique real-time network-scheduler that schedules network-traffic based upon DDS QoS-policies such as 'Deadline', ‘LifeSpan’ and ‘Transport_priority’.

This scheduling strategy is implemented within the context of run-time simulator developed using real-time java to ensure that the delivered data is valid and on time.
The remainder of this paper is organized as follows: In Section 2, we discuss the related work, comparing our approach with some existing solutions. Section 3 summarizes the technical backgrounds of this work and describes basic principles of Real-Time DDS, Controller Area Network and some related real-time basic knowledge. Sections 4 and 5 describe the proposed architecture and mechanisms for supporting the temporal data validity and network traffic scheduling. In section 6, we describe the simulations for performance evaluation and discuss the simulation results. Finally, in Section 7, we conclude the paper with a summary.

2. Related Works

Real-time data distribution has recently emerged as an important area of research. There was a workshop dedicated to the topic (The First Workshop on Data Distribution for Real-Time Systems [4]) in May of 2003. The Object Management Group (OMG) contributes to the research efforts by standardizing data distribution in a middleware service. Developing dynamic scheduling strategies within the context of data-centric publish-subscribe systems running over real-time networks is a very challenging research topic and during the last years, several teams and companies have prominently worked on these systems. In [5], the problem of scheduling the broadcast of real-time data is considered. It provides an approximate version of the Longest Wait First heuristic that reduces overhead. Similar work [6] describes a Broadcast on Demand technique that schedules the broadcast using earliest deadline first, periodic or hybrid scheduling algorithms. The work described in [7] is a speculative data dissemination service that uses geographic and temporal locality of reference to determine which data to be disseminated. These techniques take into account the deadline timing constraints of the clients, but do not consider nor the data temporal consistency neither the use of underlying real-time networks.
An application area that has provided various research efforts towards data distribution is embedded sensor networks [8,9,10,11,12]. While all of the work described here provides valuable insights into solving the problem of data distribution in sensor networks, none considers real-time characteristics of the data or of the applications. That is, neither deadlines on data delivery nor temporal consistency of the data is supported.
A large amount of real-time data distribution research has been done at the University of Virginia (UVa) in the context of wireless sensor networks [13,14,15,16]. This work does address the deadlines of requests. Also, temporal validity is considered in the sense that data values are reported before they expire, but with corresponding confidence values. However, it does not provide assurance that the data is temporally valid when it arrives at the requestor.
DDS being an API only specification does not specify a transport model. However, DDS does not depend on reliable and ordered delivery of messages, there is not enough works dealing with implementing DDS upon real-time networks. There are several commercial products that support multiple network-interfaces for transparent forwarding of DDS-data and/or fault tolerant communication paths between nodes. PrismTech[17] has a product called OpenSplice[18] which is compliant with real-time networking. DDS implementation compliant with CAN-based networks have not been treated yet, but similar works can be mentioned such as ROFES[19]. In

the context of ROFES platform, S. Lankes, A. Jabs and T. Bemmel describe the implementation of a CAN-based connection-oriented point-to-point communication model and its integration into Real-Time CORBA; but this project hadn't been extended to support data distribution service.
These research works has been enforced by several commercial products which are working on becoming compliant with the OMG's Data Distribution specification. Real-Time Innovations [20] has a product called NDDS that provides publish-subscribe architecture for time-critical delivery of data. Thales Naval Nederland [21] has a product called SPLICE [22] that provides a data-centric architecture for mission-critical applications. Both of these products provide valuable real-time features in data distribution. But neither guarantees data temporal deadlines nor real-time network support.

3. Technical Backgrounds

3.1 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.
CAN bus is based on the arbitration scheme Carrier Sense Multiple Access/Arbitration on Message Priority (CSMA/AMP). 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/AMP 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. We argue that for embedded control systems built around data-centric publish-subscribe paradigm the CAN-Bus is particularly suited to implement a publisher-subscriber model of communication.

The presented approach associates advantage of the built-in CSMA/AMP access protocol of CAN bus and the temporal QoS policies proposed within DDS mechanisms to compute the priority of the CAN message. The comprehensive scheduling approach, we presented in this paper, determines the message priority (also called the network priority) that reflects the urgency of the requestor – the subscriber in the context of the DDS communication.

3.2 Basic DDS Features

DDS targets real-time systems; the API and Quality of Service (QoS) are chosen to balance predictable behavior and implementation efficiency/performance. The DDS specification describes two levels of interfaces:

- A lower level Data-Centric Publish-Subscribe (DCPS) that is targeted towards the efficient delivery of the proper information to the proper recipients.
- An optional higher-level Data-Local Reconstruction Layer (DLRL), which allows for a simpler integration into the application layer.

The DCPS model builds on the idea of a "global data space" of data-objects that any entity can access. Applications that need data from this space declare that they want to subscribe to the data, and applications that want to modify data in the space declare that they want to publish the data. A data-object in the space is uniquely identified by its keys and topic, and each topic must have a specific type. There may be several topics of a given type. A global data space is identified by its domain id, each subscription/publication must belong to the same domain to communicate.
Figure 1 illustrates the overall data-centric publish-subscribe model, which consists of the following entities: DomainParticipant, DataWriter, DataReader, Publisher, Subscriber, and Topic. All these classes extend Entity, representing their ability to be configured through QoS policies, be enabled, be notified of events via listener objects, and support conditions that can be waited upon by the application. Each specialization of the Entity base class has a corresponding specialized listener and a set of QoSPolicy values that are suitable to it.

Publisher represents the objects responsible for data issuance. A Publisher may publish data of different data types. A DataWriter is a typed facade to a publisher; participants use DataWriter(s) to communicate the value of and changes to data of a given type. Once new data values have been communicated to the publisher, it is the Publisher's responsibility to determine when it is appropriate to issue the corresponding message and to actually perform the issuance (the Publisher will do this according to its QoS, or the QoS attached to the corresponding DataWriter, and/or its internal state).

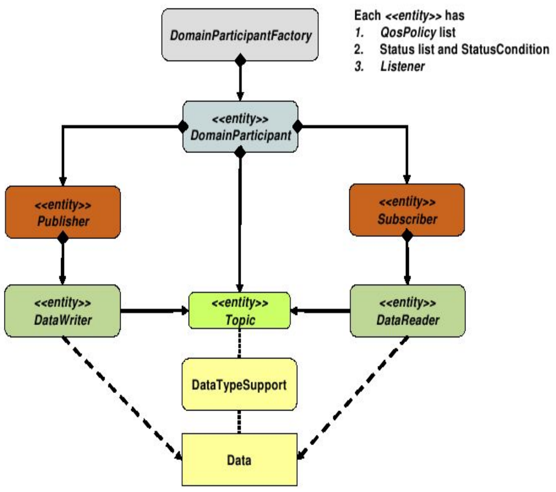


Fig. 1 UML diagram of the DDS data-centric publish-subscribe interfaces.

A Subscriber receives published data and makes it available to the participants. A Subscriber may receive and dispatch data of different specified types. To access the received data, the participant must use a typed DataReader attached to the subscriber. The association of a DataWriter object (representing a publication) with DataReader objects (representing the subscriptions) is done by means of the Topic. A Topic associates a name (unique in the system), a data type, and QoS related to the data itself. The type definition provides enough information for the service to manipulate the data (for example serializes it into a network-format for transmission). The definition can be done by means of a textual language (e.g. something like "float x; float y;") or by means of an operational "plugin" that provides the necessary methods. The DDS middleware handles the actual distribution of data on behalf of a user application. The distribution of the data is controlled by user settable Quality of Service (QoS). A Quality of Service is a set of characteristics that controls some aspects of the behavior of the DDS Service. Below, we mentioned some QoS policies that deeply impact the architectural design of our scheduling strategy:

- The DEADLINE QoSPolicy expresses the maximum duration (deadline) within which a DataReader expects a data-object instance to be updated. If a sample is not received within the deadline, a listener method is called.
- The LATENCY_BUDGET QosPolicy provides a hint as to the maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.
- The LIFESPAN QosPolicy, on a DataWriter and Topic, which specifies how long the data written by a DataWriter is considered valid ("time to live").
- The TIME_BASED_FILTER QosPolicy specifies a minimum_separation value that allows a DataReader to specify that it interested only in (potentially) a sub-sampled set of the values for a data-object instance.
- TRANSPORT_PRIORITY QoSPolicy, in a DataWriter, which allows a DDS application to take advantage of transports that are capable of sending messages with different priorities.
- The RELIABILITY QosPolicy, on a DataWriter, DataReader, or a Topic. This policy determines whether a message should be sent best effort (send once without expecting acknowledgments) or reliably (resent until positively acknowledged).

With these QoS policies, and many others, the DDS publish-subscribe map well to the real-time communications problem. However, publish-subscribe as we have described it so far is not enough. Real-time systems have several other needs. For example, real-time programs must be able to control the trade-off between delivery reliability and delivery timing. These problems can be solved when using a global scheduling strategy that works with the DDS QoS parameters (considered as end-to-end scheduling parameters) and make efficient use of multicast and prioritized access of CAN-based networks. This global scheduling framework coordinates with local scheduling mechanisms to manage the network access and consequently makes globally sound scheduling decisions for the system. The scheduling strategy must resolve a number of design challenges. In the next sections we examine two challenges:

- Design of an algorithm that computes the data delivery deadline for each published data, in order for DataReaders to read valid data. This computation is based on the DDS QoS parameters, we call it: LifeSpan-based Consistent Data Delivery.
- Determining the CAN message priority using a mapping from the delivery deadline of published data to the message deadline.

4. The Proposed Architecture

4.1 System Architecture

To illustrate the utility of our Comprehensive Scheduling Strategy (CSS), we have chosen to work within a platform of a telecom equipment interconnection. In this system, a set of network processors subsystems produces routing data. This data must be distributed along a chain of components so that it can be used by the signaling processors and the switch fabric to perform an optimal switching/routing process. In previous works [23, 24, 25], we attempted to implement this system using the CORBA client/server architecture. This kind of implementation introduced complexity and inefficiencies.
The main goal of the proposed architecture is to reduce this complexity when adopting the data-centric publish-subscribe paradigm -according to the DDS specification - which is more adapted for the data representation and communication. The framework architecture [26] is a set of nodes connected via a Real Time Transport protocol. In each node is embedded a Real Time Operating System, a middleware, and a Publish-Subscribe interface according to DDS specification. This is depicted in figure 2.

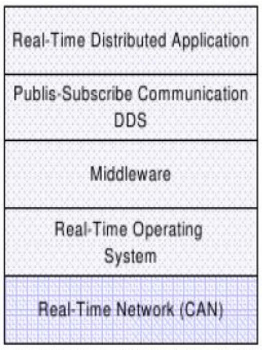

Real-Time Distributed Application
Publis-Subscribe Communication DDS
Middleware
Real-Time Operating System
Real-Time Network (CAN)

Fig. 2 Architecture of distributed real-time system using a publish-subscribe paradigm.

The communication between nodes is achieved due to publish subscribe interface via the Global Data Space that is represented by a relational data model.The middleware has to keep track of the data objects instances, which are considered as rows in a table. Each data object is identified by the combination of a topic and a topic specified key. Figure 3 depicts our overall system architecture.

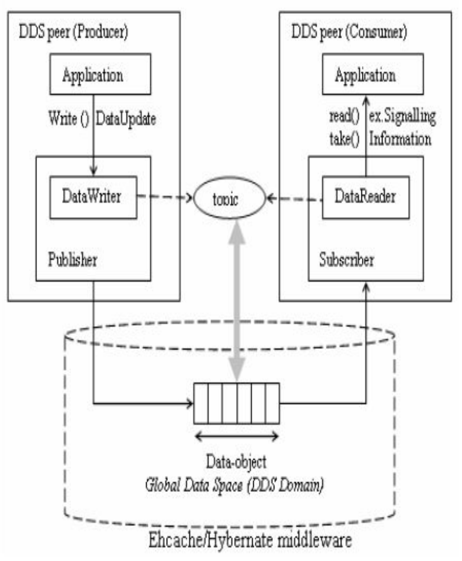


Fig. 3 Matching the topic with the adequate data-object via the middleware

This data-centric middleware allows the application to identify “data-objects” to the communication. The “data-objects” are unique in the ‘global data space’ of the distributed system across all participants. Each participant is regarded as having a local cache of the global data-object. A message on a topic is regarded as an update to the data-object that can be identified and managed by the middleware. Local changes to a data-object are propagated by the middleware; the middleware can distinguish between messages or update samples from different data-objects and manage their delivery to the interested participants on a per data-object basis. This scenario, in the proposed implementation, is achieved by a clustered distributed database based on ehcache and hibernates. The first element of the proposed scheduling solution is to develop the major components of the comprehensive scheduling framework. Figure 4 illustrates the design of the framework.
There are five essential components in this framework, schedulable jobs (DataWriter/DataReader), Local EDF Scheduler, Global Scheduling System (GSS), Local Info Collection and System Info Repository. These components are independent and coordinated with each other. DataReaders and DataWriters are the schedulable entities in our system architecture; we referred to them by schedulable jobs.

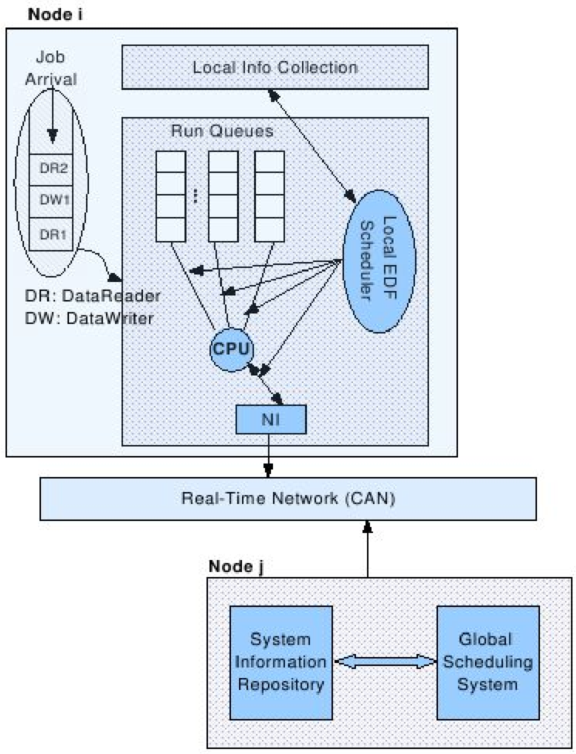


Fig. 4 Comprehensive Scheduling Framework.

When a job is spawned by an application, it passes its scheduling parameters to the Local Scheduler. The latter sends these parameters to the GSS, which returns the globally sound scheduling parameters (e.g. delivery deadline). The parameters passed along are determined by the DDS QoS policies. On each node of the system, a local scheduling component (EDF scheduler) schedules access to resources within that node, and a local information collection component record a variety of status information such as CPU utilization, progress of application activities, and success or failure of jobs in meeting their deadlines. This local status information is distilled into higher-level information such as temporal parameters of local tasks. The higher level information is sent to a distributed information collection service called the System Information Repository (SIR). The global scheduling system makes efficient use of the SIR to compute the delivery deadline of each published data and the deadline of message when a transmission occurs.
Some of the proposed system characteristics are summarized in the table below.

Table 1: System Characteristics

System Characteristics	Data Characteristics
dynamic applications	single source of each data item
real-time network	temporally constrained data
static infrastructure	periodic data production
hard/soft real-time	relational data structure
dynamic scheduling policy	asynchronous data production

4.2 Data Distribution Model

In this section we describe the model on which our work is based. Figure 5 depicts the elements of the model. The DataObject represents the data that is being distributed. A topic can identify a collection of DataObject instances. In case a set of instances is gathered under the same topic, different instances must be distinguishable. This is achieved by means of the values of some data fields that form the key to that data set. **(Topic, Key)** is a unique identifier of the data object within the system; we referred to this couple by Data Identifier (DID). **Value** is the value of the data object. This can be a simple atomic value, or a structured value depending upon the granularity of the data. According to the DDS specification, SOURCE_TIMESTAMP (S_TS) is the time (timestamp) at which the object was last updated. LifeSpan (LS) is the object validity, a time interval within which the data object is considered to be valid after its update. When the LifeSpan expires, the data is considered temporally invalid.

DataObject = <DID; Value; S_TS; LS>
DataWriter DW = <WID; Node; DID; SP>
DataReader DR = <RID; Node; DID; SP, CT>
Distribution Dist = <Dist_ID; DID; SP>
Transmission Tr = <Tr_ID; Tr_Deadline; Tr_Priority >
Scheduling Parameters SP = <P; D; R; E>

Fig. 5 Real-Time Data Distribution Model.

The DataWriter is the entity that produces the data that is to be distributed. WID is a unique identifier for the DataWriter. The DataReader is the entity that takes/reads the data. RID is a unique identifier for a DataReader. Node is the computing element on which the DataWriter/DataReader executes. SP is a set of scheduling parameters. P is the period of the task. Recall that our solution addresses the problem space of periodic data distribution. D is a deadline within the period. R is the release time within the period after which the task may start to execute. E is the worst-case execution time of the task. Note that the DataWriter and the DataReader may have different scheduling parameters, but these parameters have to be compatible. CT represents the DataReader

Completion time i.e. the point of time in which the DataReader takes the dataobject.
Dist is a distribution of data from the source node (the node on witch executes the corresponding DataWriter). A distribution has its own unique identifier Dist_ID. It also has its own scheduling parameters that can be determined by the LifeSpan-based Consistent Data Delivery algorithm described in the next section. This algorithm consider the scheduling parameters of the data topic, DataWriters, DataReaders, and the data object validity interval to determine the scheduling parameters of the distribution (especially the distribution deadline). When the new data instance has to be transmitted by the real-time network, the scheduler computes the message priority using the distribution deadline.

4.3 Task Model

In this section, we develop a tasks and subtasks model that describes the execution and communications of the DataWriters and DataReaders over the distributable system. In our model the schedulable entity (DataWriter or DataReader) is modeled with periodic real-time Task (T), the scheduling parameters of this entity represent real-time attributes of the task. Four temporal parameters will be taken in consideration in the model: 1) the task release time, 2) the task deadline, 3) the task period and 4) the task execution time.
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 containing the inputs for T_2. Then after T_2 finishes it is necessary to ship the final result to some other site. 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 queuing discipline, e.g., earliest deadline first (EDF). Associated with each task X (or a subtask) is five attributes denoted by the following functions:

- **R(X)** = arrival (or release) time of X,
- **Sl(X)** = slack of X,
- **D(X)** = deadline of X,
- **E(X)** = real-time execution of X,
- **P(X)** = period of X,
- **Pex(X)** = predicted execution time of X.

The first four attributes are related by the following equation:

$$D(X) = R(X) + E(X) + Sl(X) \tag{1}$$

We assume that the deadline and the execution time of the task are known, since they are member of the scheduling parameters, introduced by the application QoS policy when submitting a DataWriter or DataReader. These QoS policy parameters are mapped to the task attributes. The slack can be computed using the above equation. The next section will introduce the algorithms we developed to evaluate the distribution deadline and the transmission deadline.

5. Predictable Delivery Enhancement

DDS provides QosPolicies specifically targeted to minimum latency, predictable real-time operation in high-performance distributed data-critical systems. The DEADLINE QoSPolicy expresses the maximum duration (deadline) within which a DataReader expects a data-object instance to be updated. This section describes the algorithms developed within the context of the comprehensive scheduling strategy. The main goal of these algorithms is to ensure that all DataReaders take their requested data within the specified deadlines, and that all DataReaders read temporally valid data.

5.1 Distribution Deadline Computation

This section describes the LifeSpan-based Consistent Data Delivery algorithm. The main goal of this algorithm is to examine the specified system, and compute the scheduling parameters for the required data distributions. The algorithm considers the periods of the DataWriters and the periods and deadlines of the DataReaders to determine a deadline for the distribution. The following assumptions are made about the environment in which the algorithm works:

1. All nodes, data objects, and scheduling parameters are known a priori.
2. Each DataReader/DataWriter has a local node, where it originates.
3. The period of a DataWriter is always less than its LifeSpan.
4. The “offered deadline period” (DataWriter) is always less then the “requested deadline period” (DataReader).

In order for DataReaders to read valid data, the scheduling parameters of the data distribution must be such that the distribution will finish delivering the data before the DataReaders use it. Further, there can be more than one DataReader that requires the delivery of the same data object, possibly at different rates, with different deadlines.

Thus, the computation of the scheduling parameters for the distribution should be able to consider the deadlines of all DataReaders that require the data. The period of the distribution is the same as the period of the DataWriter of the data. The release time of the distribution should be at the start of its period. Let *d* be the deadline that is computed for a distribution *Dist* from DataWriter *DW* to a set of *m* DataReaders *DR1,…,DRm* for a request of data object *DID*. The period of *DW* (and therefore of *Dist*) is *p*. Let *N* be the least common multiple of the periods of all DataReaders of *DID* and the period of the DataWriter *DW*. Let *n* be the number of periods that should be considered for the analysis, where n is computed as

$$n = N/p$$

We call *N* the superperiod of the distribution because it represents a complete cycle of all DataReaders for the specified data. We define LS_i (LifeSpan) to be the point in time in the i^{th} period of the distribution that the data-object (from the most recent update) becomes temporally invalid. An invalid interval is an interval of time during which the data-object does not have a valid value associated with it, that is, the data-object is temporarily inconsistent. Figure 6 depicts an invalid interval. LS_i is the time within period P_i that the data that was updated during period P_{i-1} becomes invalid. The *d* in the figure represents the deadline of the distribution within its period. The invalid interval is the time between *LSi* and this deadline because after the deadline, a new value of the data will have been delivered.

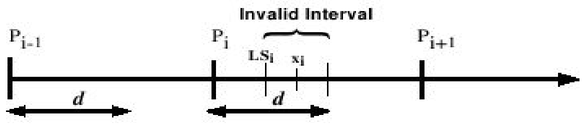


Fig.6 Data Distribution Deadline.

Initially we set the deadline of the distribution equal to its period. The deadline computation process is described by the following algorithm.

Algorithm: LifeSpan-based Consistent Data Delivery

```
1. Input: DID (data Identifier), SP_DW
2. scheduling parameters of the DataWriter,
3. SP_DR_1 ... SP_DR_m (scheduling parameters of the m
4. DataReaders), Output: d (Distribution deadline)
5. Initialization: Dist.deadline = Dist.period =
   DW.period;
6. i := 1;
7. function Distribution_Deadline_Computing
8. for each of the n periods in the superperiod do
8.     P_i := i*P(DW) ;
9.     x := min {R(DR_1), R(DR_2),..., R(DR_m) } = R(DR_k) ;
10.        if (x > P_i + Dist.deadline)
11.          d : = Dist.deadline ;
/* maintaining the last computed distribution deadline */
12.        end if
13.        if (( x > LS(DW)) and (x < P_i + d))
14.        d := x – P_i ;
15.        end if
16.        if (( x < LS(W) ) and (x+E(DR_k) )
17.        d := LS(DW) – P_i ;
18.      end if
19.    i := i + 1;
20.  end for
21. return d ;
```

There are three cases to consider when calculating the deadline: 1) if no DataReader's release time belongs to the invalid interval, the deadline is unchanged, because no DataReader will be using invalid data. 2) If some DataReader is released time x, after *LS(DW)* then the deadline is changed to $x – P_i$. 3) If any DataReader has started before or at *LS(DW)* and continues to execute in the invalid interval, then the deadline is changed to $LS(DW) – P_i$.

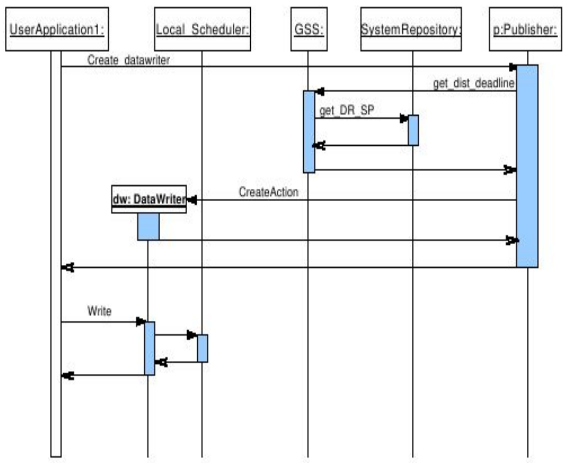


Fig.7 Data distribution deadline evaluation diagram

Figure 7 depicts the way the DataWriter, DataReader, the local scheduler, the global scheduling system and the system repository interact in order to evaluate the distribution deadline.

5.2 Transport Priority Computation

As we indicated above, to achieve the predictable communication, DDS specification defines the transport priority QoS policy. The purpose of this QoS is to allow the application to take advantage of transports capable of sending messages with different priorities. The Transport_Priority policy depends on the ability of the underlying transports to set a priority on the messages they send. Any value within the range of a 32-bit signed integer may be chosen; higher values indicate higher priority.

However, any further interpretation of this policy is specific to a particular transport and a particular implementation of the Service. For example, a particular transport is permitted to treat a range of priority values as equivalent to one another. It is expected that during transport configuration the application would provide a mapping between the values of the TRANSPORT_PRIORITY set on DataWriter and the values meaningful to each transport. This mapping would then be used by the infrastructure when propagating the data written by the DataWriter. The CAN bus is well adapted for such transport. The CSMA/AMP 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 scheduler is to calculate the message transmission deadline using the subtask deadline assignment [27]. The second is to map this deadline into the transport priority.

We consider the Distribution as a global task *T* that consists of three subtasks T_1, T_2, and T_3. The DataWriter (*DW*) and the DataReader (DR_k) are considered, respectively, as the first and the third subtasks. The message transmission on the CAN bus can be seen as the second subtask T2 and its deadline can be calculated using the Equal Slack strategy (EQS) [27].

$$Dist = DW + Tr + DR_k \tag{2}$$

Using the EQS 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. Equation 3 describes the EQS strategy:

$$D(T_i) = R(T_i) + Pex(T_i) + \frac{D(T) - R(T_i) - \sum_{j=i}^{n} Pex(T_j)}{(n-i+1)} \tag{3}$$

In our situation, the global task which is the distribution *Dist* is composed of the DataWriter, DataReader and the transmission subtasks. Thus, equation (3) gives

$$D(Tr) = R(Tr) + Pex(Tr) + \frac{[D(Dist) - R(DW) - (Pex(Tr) + Pex(DR_k))]}{2} \tag{4}$$

- *D(Tr)*: the transmission deadline is a point of time, 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.
- *R(Tr)*: release time of the transmission, i.e., the point of time when the middleware begins the message transmission.
- *Pex(DR_k)*: predictable execution time of the DataReader DR_k. which realizes the condition:

$$R(DR_k) = \min \{R(DR_1), R(DR_2),..., R(DR_m)\} \tag{5}$$

- *Pex(Tr)*: predictable execution time of *Tr*, corresponds to the time taken by the message transmission over the CAN bus. *Pex(Ti)* can be assumed to the longest time taken to transmit message *m (Cm)*, based on bounding the number of bits sent on the bus for this message. For CAN networks we have the fellow expressions:

For CAN 2.A:

$$C_m = \left\{ \left(\frac{34+8S_m}{4} \right) + 47 + 8S_m \right\} \tau_{bit} \tag{6}$$

For CAN 2.B:

$$C_m = \left\{ \left(\frac{54+8S_m}{4} \right) + 57 + 8S_m \right\} \tau_{bit} \tag{7}$$

The term S_m is the number of bytes in payload field of the message and τbit is the bit time of the bus (i.e. 1µs 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. The question is: how to map this deadline to the transport priority defined by the DDS specification?

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. Each value of the transmission laxity is mapped to a portion of future time, a *priority tick* Δt_p. Since there are only a limited number of different priorities (2^6=64 priority levels), only a limited number of priority ticks are visible
- P_{min}: The highest priority = lowest binary value for real-time priorities.
- P_{max}: The lowest priority = highest binary value for real-time priorities.

We denote *P* as the transport priority value; this value will be mapped to the CAN priority of the message transmitted when propagating the data written by the DataWriter. Figure 8 shows how the system components interact to compute the transport priority of the message.

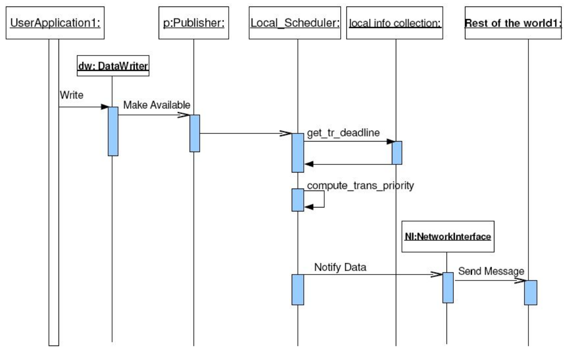


Fig.8 Transport Priority computation diagram

6. Implementation and Evaluation

6.1 Implementation Details

Here we evaluate the performance of the proposed scheduling schemes. Before we present the detailed discussion of the results, we describe the simulation platform, workloads, and the performance metrics used in this study. We implemented the proposed scheduling schemes onto an extensible/unified simulation framework termed ***DDSoverCAN***. The implementation is programmed in Real-Time Java and runs on Jamaica VM, which is a compatible RTSJ Virtual Machine, and Linux Kernel 2.6. The simulator provides users with graphical interfaces to specify the overall system configuration. A set of ASCII files containing descriptions of all of the topics, data, nodes, publishers, subscribers and network configuration, are created and stored. We chose to use real-time Java specification to implement our simulator; because it illustrates the real-time needs expressed by our framework. Real-Time Specification for Java (RTSJ) [28] is designed to support both hard and soft real-time applications. Among its major features are: scheduling properties suitable for real-time applications with provisions for periodic and sporadic tasks, support for deadlines and CPU time budgets, and tools to let tasks avoid garbage collection delays. The key features of the RTSJ are:

- Real time thread: executable entity of work which, at a minimum, characterized by a worst-case execution time and a time constraint.
- Scheduling: the default scheduler guards against priority inversion, it is also possible to define user schedulers.
- Timing: RTSJ defines relative and absolute times, with a Nanosecond precision time values.

Real-time tasks are implemented as real-time threads according to a specific mapping (shown in table 2). The EDF schedulers are defined as an extension of the default scheduling class *(javax.realtime.PriorityScheduler)*. The CAN access is controlled by a set of daemons that implements the network controllers, the message queues and the bus behavior.

Table 2: System Characteristics

Task model	**RT-Java implementation**
periodic real-time task T	periodic Realtime Thread RT_Thrd
D(T)	RT_Thrd.deadline
Pex(T)	RT_Thrd.Cost
R(T)	RT_Thrd.release

The following metrics are defined to evaluate the performance of the proposed scheduling scheme:

- The age of the data: for each data instance, the age represents the difference between the point of time when the DataReader reads the instance (an absolute time) and the point of dime the data is updated by the DataWriter, also called source timestamp (absolute time). Thus, the data age is represented as a relative time (difference between absolute times).

- The delivery rate: Suppose the publishers have published k dataobjects in the super-period, denoted by $d_1, d_2 \ldots d_k$. For a dataobject d_i, let the number of DataReaders interested in it be ts_i, and the number of DataReaders that receive it before the deadline be ds_i. We can define a metric called delivery rate of the system as follows:

$$\frac{\sum_{i=1}^{k} ds_i}{\sum_{i=1}^{k} ts_i} \tag{8}$$

- The completion time: represents the difference between the point of time in which the DataReader's execution cycle takes end and the release time of the DataReader.

$$CT(DR) - R(DR) \tag{9}$$

6.2 Performance Results

In order to demonstrate the effectiveness of our implementation, we have developed an evaluation test scenario. In the simulated network, there are 3 nodes, 2 data source (2 Topics) – each source is associated to a DataWriter – and 4 Subscribers. Each subscriber is associated to 5 DataReaders having the same scheduling parameters as the subscriber. Table 3 shows the parameters and ranges of values used in the test scenario.

Table 3: System Setup

Name	Period (ms)	Release (ms)	Deadline (ms)	Life Span	Cost (ms)
PUB1_T1_N1	80	0	10	150	5
PUB2_T2_N1	140	0	20	170	8
SUB1_T1_N2	80	50	50	-	7
SUB2_T1_N3	130	50	50	-	3
SUB3_T2_N1	150	90	60	-	4
SUB4_T2_N3	200	120	60	-	6

Figure 9 shows the temporal consistency results for Topic1 which is published by Publisher1. The horizontal line in the graph represents the LifeSpan value of the dataobject being distributed. The other points represent the ages of the data objects at the time they were read by the interested DataReaders.

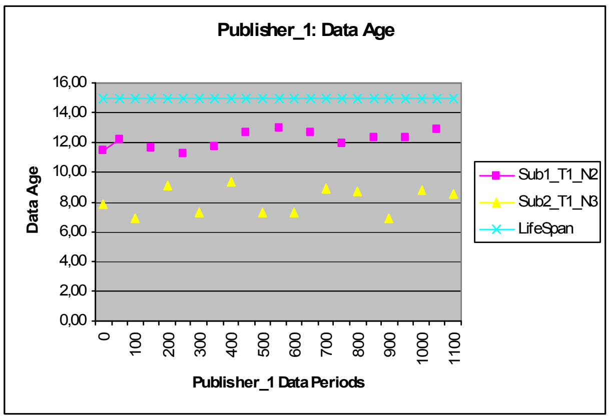


Fig.9 Topic 1 Consistent Distribution

Figure 10 displays the delivery rate of the DataReaders associated with SUB3_T2_N1 and SUB4_T2_N3. It is clear to see that all of the DataReaders, in each of the periods, read temporally consistent data.

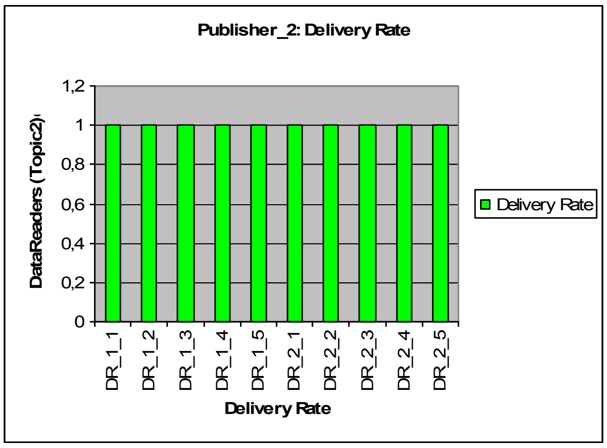


Fig.10 Topic 2 Consistent Distribution

Figures 11 and 12 display the deadline results for the test scenario. The horizontal line in each graph indicates the deadline for the specified Subscriber. The other points in the scatter graph represent the completion times of the DataReaders over the super-period cycle. As the figure indicates, except for a few statistical anomalies in the first few periods, all of the DataReaders complete before the specified deadline, as the theoretical results had predicted.

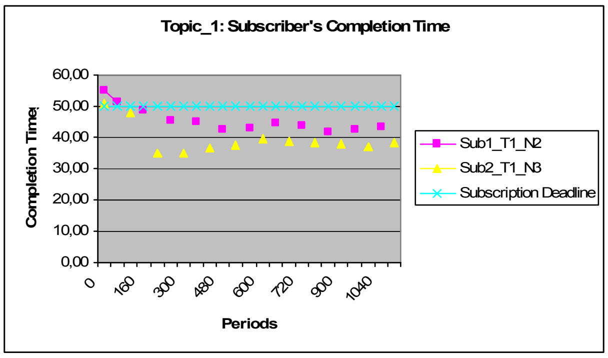


Fig. 11 Topic 1 Subscriber's Completion Time

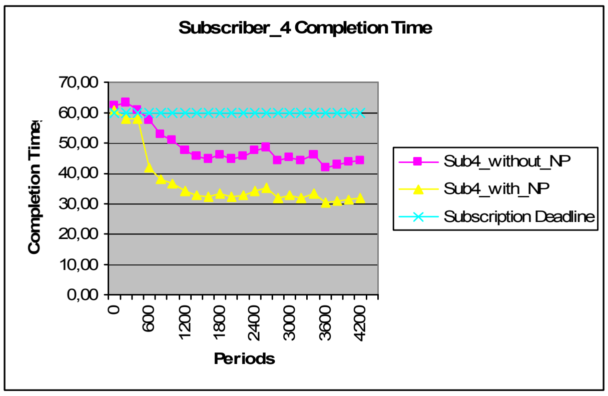


Fig. 12 Subscriber 4 Completion Time with/without Network Priority Support

In the first few periods, there was an overloading, on each node, caused by the local information collection and the system repository constitution that caused the tasks to complete after the deadline. Figure 12 shows that the proposed Network priority mapping algorithm significantly outperforms the global scheduling solution by up to 19.4% in the average task completion time. This improvement is achieved by reducing wasted network idle time (by introducing network priority that best reflects the task urgency without sacrificing fairness).

7. Conclusion

Data-Centric Publish-subscribe (DCPS) is a widespread communication paradigm for asynchronous messaging that naturally fits the decoupled nature of real-time distributed systems, allowing simple and effective development of distributed applications. With the work described herein, the CAN bus has been rendered more usable in the field of distributed DCPS systems. We tried to design a comprehensive scheduling system that interacts with the main actors described by the DDS specification. This interaction aims to integrate the DDS QoS parameters to improve the consistent data delivery and 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 resources. The priority based mechanism and the EDF scheduling strategy used within the context of this work is adapted with soft real-time communication system. One promising research direction is to combine priority-based mechanisms in conjunction with implicit coscheduling mechanisms and to extend this strategy to support the hybrid real-time bus scheduling mechanisms for CAN.

References

[1] G. Blair, G. Coulson, P. Robin, M. Papathomas, An Architecture for next generation middleware, Proc, 4th Annu, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, London, England, 1998.

[2] OMG, "Data Distribution Service for Real-Time Systems Specification", March 2004.

[3] ROBERT BOSCH GmbH, CAN Specification version 2.0 (1s edition, 1991).

[4] First International Workshop on Data Distribution for Real-Time Systems, In conjunction with International Conference on Distributed Computing Systems, May 2003.

[5] M. Karakaya, O. Ulusoy, Evaluation of a Broadcast Scheduling Algorithm, Lecture Notes in Computer Science, Springer-Verlag, v. 2151, 2001.

[6] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, K. Ramamritham, Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments, Proceedings of the Fourth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'97), 1997.

[7] A. Bestavros, Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems, Proceedings of the 1996 International Conference on Data Engineering, New Orleans, LA, March 1996.

[8] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, D. Estrin, Data-Centric Storage in Sensornets, First Workshop on Hot Topics in Networks (HotNets-I) 2002.

[9] F. Ye, H. Luo, J. Cheng, S. Lu, L. Zhang, A Two-Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks, MOBICOM'02, September 23-28, 2002, Atlanta, GA.

[10] Y. Yao, J. Gehrke, Query Processing for Sensor Networks, Proceedings of the 2003 Conference on Innovative Data Systems Research, Jan. 2003.

[11] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In Proceedings of the Second International Conference on Mobile Data Management, 2001.

[12] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, Energy-Efficient Communication Protocol for Wireless Microsensor Networks, In HICSS '00, January 2000.

[13] B. C. Lu, B. M. Blum, T. Abdelzaher, J. A. Stankovic, T. He, RAP: A Real-Time Communication Architecture for Large-Scale Wireless Networks, Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02), 2002.

[14] T. Abdelzaher, J. Stankovic, S. Son, B. Blum, T. He, A.Wood, C. Lu, A Communication Architecture and Programming Abstractions for Real-Time Embedded Sensor Networks, Proceedings of the First International Workshop on Data Distribution for Real-Time Systems, Providence, RI, May 2003.

[15] S. Kim, S. H. Son, J. A. Stankovic, S. Li, Y. Choi, SAFE: A Data Dissemination Protocol for Periodic Updates in Sensor Networks, Proceedings of the First International Workshop on Data Distribution for Real-Time Systems, Providence, RI, May 2003.

[16] S. Bhattacharya, H. Kim, S. Prabh, T. Abdelzaher, Energy-Conserving Data Placement and Asynchronous Multicast in Wireless Sensor Networks, Proceedings of the First International Conference on Mobile Systems, Applications and Services, San Francisco, CA, May 2003.

[17] PrismTech, http://www.prismtech.com.

[18] H. V. Hag, OpenSlice Overview, white paper, 2006.

[19] S. Lankes, A. Jabs, and T. Bemmerl, Integration of a CAN-based connection-oriented communication model into Real-Time CORBA, Proc, IEEE International Parallel and Distributed Processing Symposium, Proc, 11th Annu, Workshop on Parallel and Distributed Real-Time Systems, Nice, France, April 2003.

[20] Real-Time Innovations, http://www.rti.com

[21] Thales Netherland, http://www.thales-nederland.nl/

[22] J. H. van 't Hag Data-Centric to the Max - The SPLICE Architecture Experience, Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03), May 19 - 22, 2003.

[23] T. Guesmi, S. Hasnaoui and H. Rezig, "Network Priority Mapping Using Dynamic RT-CORBA Scheduling Service", International Revue On Computers Software,

September Issue, ISSN 1828-6003.

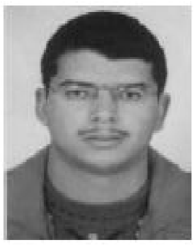
[24] T. Guesmi, S. Hasnaoui and H. Rezig, "Using RT-CORBA Scheduling Service and Prioritized Network Traffic to Achieve End-to-End Predictability", the 2006 International Conference on Communications In Computing (CIC'06), 26-29 June 2006, USA.

[25] T. Guesmi, S. Hasnaoui and H. Rezig, "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", the 2005 International Conference on Parallel and Distributed Process Techniques and Applications (PDPTA'05), 26-29 June 2005, USA.

[26] M. A. Mastouri, S. Hasnaoui, "Design of Switch Architecture According to the MSF Framework Using the VSI Interface and the CAN-IOP Protocol", ACIDCA-ICMI'2005 conference proceedings, p 13.

[27] B. Kao, H. Garcia-Molina, Deadline Assignment in a Distributed Soft Real-Time System, Proc, 13th Annu, International Conference on Distributed computing Systems. 1993.

[28] Sun Microsystems Inc., Java TM 2 SDK Standard Edition Documentation, http://java.sun.com/j2se/1.3/docs.



Tarek Guesmi is an assistant professor in the Department of Computer and Communication Engineering at High Institute of Computer Sciences and Communication Technologies (Sousse University, Tunisia). He received the Master of Science in Communication Systems from National School of Engineering of Tunis in 2002. He is actually preparing a PhD in Telecommunications. Tarek published many papers in International Conferences and Workshops. His research interests include Real-Time and distributed Systems, Computer Networks, Fault-Tolerant systems and Communication protocols.
Mr. Guesmi is a member of ACM SIGAPP Group.

Salem Hasnaoui is a professor in the Department of Computer and Communication Technologies at the National School of Engineering of Tunis. He received the Engineer diploma degree in electrical and computer engineering from National School of Engineering of Tunis. He obtained a M.Sc. and third cycle doctorate in electrical engineering, in 1988 and 1993 respectively. The later is extended to a PhD. degree in telecommunications with a specialization in networks and real-time systems, in 2000.
Salem is author and co-author of more than 40 refereed publications, a patent and a book. His current research interests include real-time systems, sensor networks, QoS control & networking, adaptive distributed real-time middleware and protocols that provide performance-assured services in unpredictable environments.
Prof. Hasnaoui is the responsible of the research group "Networking and Distributed computing" within the Communications Systems Laboratory at the National School of Engineering of Tunis. He served on many conference committees and journals reviewing processes and he is the designated inventor of the Patent "CAN Inter-Orb protocol-CIOP and a Transport Protocol for Data Distribution Service to be used over CAN, TTP and FlexRay protocols".

Houria Rezig is a professor in the Department of Computer and Communication Technologies at the National School of Engineering of Tunis. She received a PhD in Telecommunication from National School of Engineering of Tunis.
Houria is author and co-author of many refereed publications. Her research interests span distributed networks, QoS networking and optical communications.
Prof. Rezig is the responsible of the research group "Optical Networking" within the Communications Systems Laboratory at the National School of Engineering of Tunis. She served on many conference program and organizing committees.