A Time-Triggered Data Distribution Service for FTT-CORBA

Adrian Noguero
TECNALIA. ICT/ESI Division
Parque Tecnologico de Zamudio, #202
48170 Zamudio, Spain
adrian.noguero@tecnalia.com

Isidro Calvo
DISA (University of the Basque Country),
E.U.I. de Vitoria-Gasteiz, C/Nieves Cano, 12,
01006 Vitoria, Spain
isidro.calvo@ehu.es

Abstract

FTT-CORBA is a middleware architecture aimed at synchronizing the task activations of a distributed system according to a plan that may be changed at runtime. In this architecture tasks are wrapped within CORBA methods that are activated by a central node, the Orchestrator, over a LAN. Previous versions of FTT-CORBA focused on CPU-bound applications in which the communication time was neglectable. However, in some cases, this assumption is not valid. This work presents an add-on service, integrated within the FTT-CORBA middleware, aimed at minimizing the interference among task activation and data distribution messages. By using this new service the Orchestrator will be capable of controlling the transmission of data distribution packages from the distributed nodes by dynamically allocating them in specific time slots.

1. Introduction

Open technologies are increasingly demanded in distributed industrial applications since they ease the integration of devices from different vendors. These devices may include heterogeneous technologies, such as (1) different hardware platforms (e.g. based on Pentium or ARM architectures); (2) different operating systems that range from general purpose OS (Operating Systems) like Windows or Linux to RTOS (Real-Time Operating Systems) like QNX or VxWorks; (3) different programming languages like C/C++, Java or Ada; and, (4) most often, TCP/IP protocols over common LANs like Switched Ethernet [1] or even IEEE802.11.

Modern distributed industrial applications introduce also strict requirements that increment their complexity; such as, a precise management of the resources of the distributed system (CPU, network bandwidth, energy, etc), mixing several types of traffic with different quality of service (QoS) requirements and, in occasions, accurate task synchronization capabilities.

Unfortunately, programming this type of applications over TCP sockets or directly over the data link layer may become a clumsy task. That is why the introduction of middleware specifications that have been successfully used in other domains, such as CORBA [2, 3], ICE [4], OPC [5], Web Services [6] and DDS [7, 8] may simplify the creation of distributed industrial applications. Note that these middleware specifications are implemented as an additional layer over the transport layer of the TCP/IP stack to hide low-level programming details. They are based on modular approaches to solve scalability and heterogeneity issues, and they were originally designed to work in heterogeneous environments in terms of hardware platforms, operating systems and programming languages.

But most of these middleware solutions do not cope with certain issues such as the synchronization of the tasks or mixing several types of traffic in one system with different parameters of QoS.

This work is based on CORBA. The authors have chosen CORBA despite its maturity since in the DRTS domain it is still a valid standard technology that allows programmers to develop source code reusable across multiple platforms, OS and applications, maximizing technology investments [9, 10]. In particular, CORBA is frequently used in robotics applications [11]. Moreover, most of the concepts developed in this work can be easily adapted to other middleware technologies, such as ICE or DDS, since the approaches followed by these solutions are, in essence, similar to those used in CORBA. More specifically, this work presents a solution integrated within the FTT-CORBA service [12, 13, 14]. FTT-CORBA is a middleware architecture aimed at the synchronization of the task activations, wrapped as CORBA methods, in a distributed system. It also allows managing the resources (CPU, memory and battery) of the individual distributed system nodes. FTT-CORBA implements the Flexible Time-Triggered (FTT) paradigm [15] which is based on a centralized architecture for the management of the resources and task synchronization. According to the functional and non-functional requirements of the distributed applications loaded in the distributed system, the central node, the so-called Orchestrator, generates the task activation time-table using one of several scheduling

978-1-4673-4737-2/12/$31.00 ©2012 IEEE

algorithms. Then, following this table, the Orchestrator sends activation messages to the distributed tasks [13].

Previous versions of FTT-CORBA assumed CPU-bound applications; i.e. those in which the time spent communicating is much less than computing. In these applications, it is generally acceptable to disregard communications. This was a valid assumption in many applications that use high speed communication links, like switched Ethernet, to transfer small amounts of data generated at relatively low rates. However, some industrial applications require higher network bandwidth, for example, when video streams that may affect the performance of the whole system are involved. Similarly, in applications executed over shared networks, such as IEEE 802.11 or CAN, the use of the network cannot be disregarded.

In this paper the authors extend the FTT-CORBA architecture with a new service, the FTT Event Service. This service allocates specific time-windows for data transmission ensuring that data packets do not overlap with task activation messages. So, since task activation messages are 'protected', real-time performance of the application will not get affected in data-intensive applications. Furthermore, the article describes the API of the designed extension and some experimental results that prove the validity of the proposed solution.

The rest of the article is structured as follows: section 2 covers some related work relevant for this research; section 3 provides an overview of the FTT-CORBA architecture; section 4 describes the proposed solution: the FTT-Event Service; section 5 outlines the experimental layout and describes the results of the tests carried out; and finally section 6 draws some conclusions and presents further research steps.

2. Related Work

Typical distributed industrial applications mix different types of traffic such as (1) acyclic distribution of alarms and events; (2) cyclic distribution of process variables; and (3) client/server operations. These traffic types adapt better to different communication paradigms. In addition, industrial applications require the configuration of different parameters of QoS.

Coexistence between the client-server paradigm and other paradigms such as publish-subscribe or supplier/consumer has been broadly discussed in the literature. There exist several solutions that mix these approaches. In the case of the CORBA architecture, the Event Service [16] implements a publish/subscribe service on top of CORBA. This service evolved into two different extensions: the CORBA Notification Service [17] and the non-standard Real-Time Event Service provided with the TAO ORB [18]. The Notification Service included several extensions to the Event Service, like strict typing, inheritance, type correlation, or distribution policies. The Real-Time Event Service, instead, focused more on QoS assurance.

Other middleware architectures similar to CORBA, such as ICE [4], also developed analogous add-on services that include the publish/subscribe paradigm. This is the case of Icestorm [19], equivalent to the CORBA Notification Service.

The Data Distribution Service (DDS) [7] defines another middleware specification that provides a platform-independent middleware for Data Centric Publish/Subscribe many-to-many communications. Since some of the DDS vendors are also providers of CORBA solutions they provide proprietary products to decouple different types of traffic using both client/server and publish/subscribe paradigms. Unfortunately, too often this integration is based on expensive products.

The integration of data distribution paradigms has also been analyzed in other middleware architectures like Web Services [20], or in Grid computing environments [21].

Some authors [23, 24, 25] have proposed other common middleware services based on the invocation of objects in time according to a pre-planned schedule to support Time-Division Multiple Access (TDMA) network centric approaches at middleware level. However, their approaches require global time synchronization across the distributed system to coordinate the triggering of the requests. This may be not suitable for some systems that require a higher degree of flexibility to adapt to changing requirements at run-time.

This article proposes a new service integrated within the FTT-CORBA architecture. Its main objective is to allow the smooth distribution of process data without affecting the activations of the distributed tasks according to the execution plan enforced by the Orchestrator. Thus, the Orchestrator becomes a central node that dispatches an execution plan that may be changed at run-time. This plan includes the activation of the distributed tasks as well as the coordination of the data distribution among the tasks. The proposed service is based on the FTT-Ethernet protocol [15] and the Kokyu scheduler of the TAO Real-Time Event Service [22].

3. Middleware Architecture Description

This section describes the fundamentals of the Flexible Time-Triggered CORBA (FTT-CORBA) middleware [12, 13, 14]. The service assumes an underlying network infrastructure capable of doing physical multicast like Ethernet or IEEE802.11.

3.1. Architecture Overview

The FTT-CORBA architecture is comprised of a set of centralized services, whose ultimate goal is to schedule the execution of the distributed applications over a period of time. More specifically, FTT-CORBA

plans the activation of the tasks in which distributed applications are broken down. These services are organized in layers [13, 14], as depicted by figure 1.

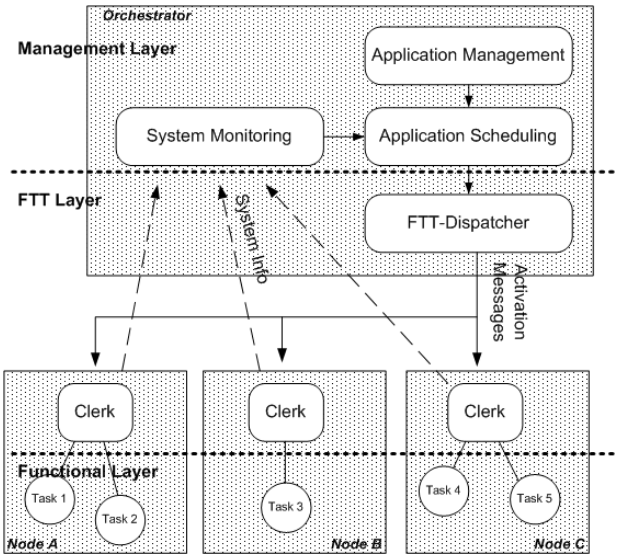

Figure 1. Layered architecture of FTT-CORBA

The top layer of the architecture contains a set of services that carry out management tasks, namely, distributed system monitoring, applications management (used as main user interface), and time and resource scheduling. All these services are implemented in the Orchestrator.

The middle layer of the architecture activates timely the individual distributed tasks following the FTT paradigm. This layer is distributed over all the participant nodes and comprises two different elements: the FTT-Dispatcher, at the Orchestrator, and the Clerks, at the distributed nodes. The FTT-Dispatcher notifies to all nodes the exact moment in which distributed tasks should start their execution by means of multicast messages. Each Clerk rules a distributed node by processing the activation messages and invoking the CORBA methods in its node accordingly. Also, Clerks gather periodically status information of every node, in terms of CPU load, available memory and remaining battery, and send it to the System Monitoring Service.

Lastly, the lower level of the architecture contains the tasks where the functionality of the applications is located. It is important to remark that, in FTT-CORBA, tasks are encapsulated as methods of standard CORBA objects. Thus, it is not necessary to recompile them to be reused, since they are externally activated by the service.

FTT-CORBA is provided as two Commercial Off-The-Shelf (COTS) executables: the Orchestrator and the Clerks that must be configured properly.

3.2. Implementation Details

In the FTT paradigm [15] time is measured and expressed using a unique time unit: the Elementary Cycle (EC) which defines the time granularity of the distributed system. FTT-CORBA uses the EC to define the timing characteristics of the distributed applications (e.g. periods or deadline) [13, 14]. This parameter must be configured during the start up of the middleware and cannot be changed at run-time.

Applications in FTT-CORBA are described as an ordered set of task executions, in which the order is defined by a directed graph, as shown in figure 2. In addition to this functional description, applications are given several non-functional parameters that complete their specification, including period, offset, deadline and priority value [13, 14]. Application models can be created using the FTT-Modeler tool [27], which is an Eclipse based tool with specific diagramming characteristics that meet FTT-CORBA requirements.

Applications can be loaded, unloaded and their parameters can be modified using the Application Management Service, described above. It is important to note that these changes can be carried out at runtime providing certain level of flexibility and dynamism to distributed applications.

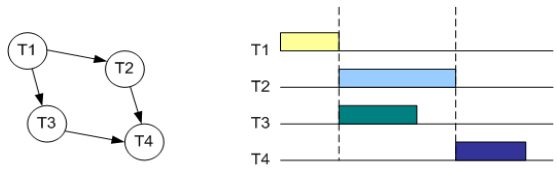

Figure 2. Application description and execution

4. FTT Event Service

Previous implementations of FTT-CORBA were aimed at CPU-bound applications where communication time was much shorter than computing time. This was a valid assumption when high-speed communication links, like switched Ethernet, were used to transfer small amounts of data generated at relatively low rates. However, some industrial applications are data intensive so data packets may cause interference in the distribution of the FTT-CORBA task activation messages. This may also be the case in shared LAN networks like IEEE802.11 or CAN.

In this work the authors propose extending the FTT-CORBA middleware architecture to solve this problem by means of a pluggable communications channel managed by the Orchestrator: the FTT Event Service. This service will be integrated within the FTT-CORBA middleware to allocate specific time-windows for data transmission ensuring that data packets do not overlap with task activations messages.

4.1. Architectural Design

The FTT Event Service has been designed to be integrated within the existing FTT-CORBA architecture non-intrusively. The new service, located at the middle layer of FTT-CORBA, is comprised of two different elements: the FTT Event Channel, deployed at the Orchestrator, and the Federated Event Channels, deployed at the Clerks.

The FTT Event Service is based on a logical communication channel, the so-called Event Channel, which follows the supplier-consumer paradigm providing one-to-many communication. The Event Channel notifies distributed nodes about when data must be sent, avoiding interferences between task activations and data exchanges. Even though in order to ease its use and comprehension it has been employed a terminology similar to the one adopted by the standard CORBA Event Service [16], it works slightly differently.

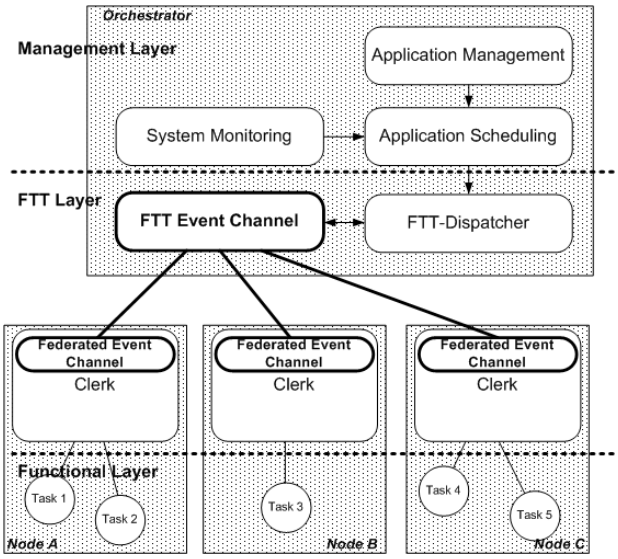

Figure 3. The Event Channel

As depicted in figure 3, data transactions are governed by the FTT-Event Channel deployed at the Orchestrator. Note that the Orchestrator is responsible for coordinating both the FTT-CORBA Dispatcher and the FTT-Event Channel. CORBA objects may access this channel through the Federated Event Channels, located at the Clerks, which negotiate with the centralized channel for their bandwidth share.

Each message sent through the Event Channel is identified by a Topic. Users may assign different priority levels to each topic so these priorities will be used by the Event Channel to order the distribution of the data according to their criticality. Federated Event Channels allow the tasks to create Suppliers and Consumers as required, each of them associated to a particular Topic. Suppliers and Consumers are in charge of producing and consuming respectively DataTokens from the Federated Event Channel. The Federated Event Channel, in turn, is responsible for holding the data tokens produced by the tasks until the FTT-Event Channel notifies its distribution. Finally, data in the FTT Event Service is not strictly typed to provide higher flexibility, so Topics are just binary elements identified by an ID.

The management of the Event Channel is carried out using standard CORBA interfaces. However, underlying multicast protocols can be used for data distribution in order to improve the service performance.

4.2. API and Service Operation

The API of the FTT Event Service defines close entities to those used by the standard CORBA Event Service [16]. This design decision has been made to ease the familiarization of the key elements of the API (see figure 4) by the potential users.

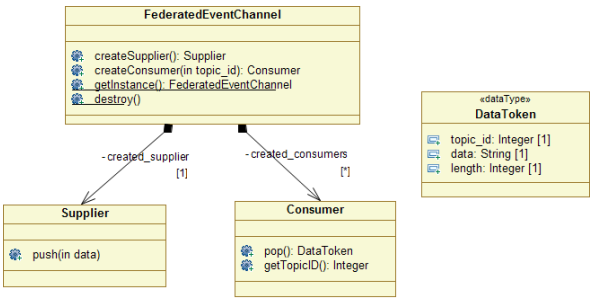

Figure 4. API of the FTT Event Service

To access the FTT Event Service tasks need to connect to an instance of a Federated Event Channel. This is done by means of the *getInstance()* method included in the FTT Event Service library. The *destroy()* method, on the contrary, cleans up all data in any queue and frees all the memory used by the service.

The Federated Event Channel is implemented as an extension to the Clerk. Therefore, tasks need to connect to the Clerk managing the distributed node in which they are deployed. Clerks on the other hand, set up the Federated Event Channels and connect them to the main Event Channel instanced by the Orchestrator. Once connected to a Federated Event Channel, task instances may create suppliers and consumers as required using the *createSupplier()* and *createConsumer()* methods.

Suppliers are used to send Data Tokens to the Event Channel using *push()*. However, they do not send the data straight away, but messages are queued in the Clerk until the Event Channel notifies the local Federated Event Channel to transmit them.

The Event Channel uses the CORBA interface specified in figure 5 to manage federated channels. The Event Channel needs to know which nodes want to send information, so it periodically polls the Federated channels for data tokens using the *getPendingTopics()* method. To avoid over and under-polling the polling period is configurable, and it is set during the booting phase of the Orchestrator. After polling all nodes, the

Event Channel orders the pending messages according to their priority, and starts the data sending process. It is important to note that the priority of each message is that of the Topic each message refers to.

The Event Channel uses the *sendTopic()* method to instruct a Federated Event Channel to trigger the distribution of a message. As the transmission time of a message can be very long in some applications (e.g. video streaming applications) the Event Channel needs to tell the Federated Event Channels which the maximum transmission time available for the current message is, in order to avoid collisions that could affect the synchronism of the system. To cope with this requirement, the *sendTopic()* method includes a timeout value that defines the maximum time available to send the message. This value is calculated by the Event Channel, and it is equal to the time remaining until the beginning of the next EC (i.e. the time until the next activation message).

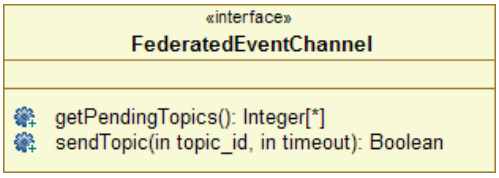

Figure 5. Control interface

Before transmitting the messages, the Federated Event Channel splits each data token at the queue into smaller fragments in such a way that each can be sent atomically through the communications stack; that is, in fragments that fit the size of the frame used at the underlying data link layer. Then, the Federated Event Channel starts sending every fragment over the network until either the whole data token has been completely sent or the timeout has expired. Messages can be sent using either unicast or multicast communications, depending on the configuration of the channels and the underlying technologies.

If the last data token was sent before the timeout granted by the Event Channel expires, the following topic in the queue is sent until the queue is empty. Before that, the Event Channel recalculates the timeout value. Otherwise, the transmission of the topic will be hold until a new invocation of the *sendTopic()* method of the Federated Event Channel is issued by the Event Channel as shown in Figure 6. If new data tokens with higher priorities enter the communication queue, the transmission of a data token can be delayed. However, the FTT-Event service ensures that the most critical messages are sent first.

Finally, transmitted data tokens are reconstructed by the Federated Event Channels of the Consumers and put in their output queues. Tasks may access these tokens using the *pop()* method.

As a final consideration, it is important to note that even if the selected service design reduces significantly the bandwidth available of the network, the main goal of the service is to avoid the interference between activation messages and data messages. Nevertheless, the service has been implemented as an optional plug-in.

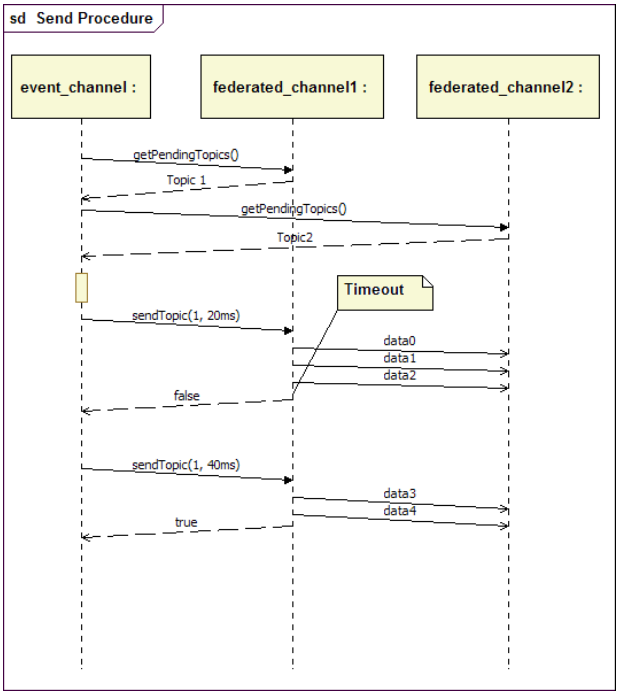

Figure 6. Message send procedure

5. Test Scenarios

In this section the authors will describe a set of laboratory tests designed to measure empirically the performance of the FTT Event Service in cooperation with FTT-CORBA.

5.1. Experimental Set Up

FTT-CORBA is intended to be used in distributed systems composed of embedded computing nodes. This kind of scenario is appropriate for many application domains, such as industrial applications, robotics or control applications. The hardware platform selected for the experiments was BeagleBoard-xM revision C. This board features an ARM Cortex A8 CPU running at 1GHz. To connect to the distributed environment, the BeagleBoard is equipped with a 10/100 Mb Ethernet connector. The central computer of the distributed system, used to execute the Orchestrator, was a desktop PC featuring an Intel i5 processor running at 2.80 GHz.

From the software point of view, all BeagleBoards run Angstrom Linux with kernel versions 2.6.34. The desktop computer runs Ubuntu 11.04, with kernel 2.6.39.

To connect the devices to the distributed system a 100 Mb Ethernet switch and a 100 Mb Ethernet hub have been used.

5.2. Synchronization Test

One of the goals of the FTT Event Service is to avoid interferences between activation messages and data exchange messages in the FTT-CORBA architecture, especially when either communication times are not small compared to the EC or the communication medium is shared by all the nodes (e.g. non-switched Ethernet or WIFI). Such interferences may lead to disturbances in the activation times of the individual tasks; thus, raising the jitter of periodic task activations. To demonstrate that FTT Event Service does not interfere with the synchronization messages of FTT-CORBA a test case has been designed using two BeagleBoards and the desktop computer.

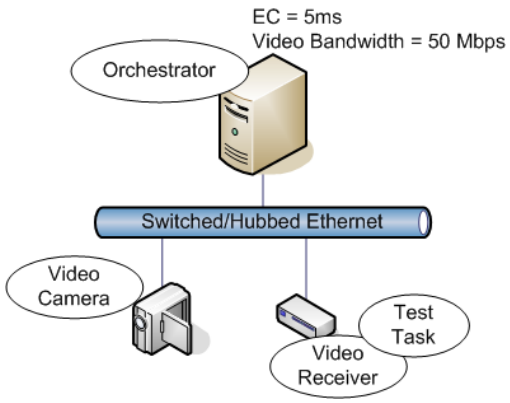

Figure 7. Synchronization test

As shown in figure 7, in one of the two BeagleBoards a simulated video camera was implemented whereas the other received the video streams and carried out the testing tasks. The first BeagleBoard simulated a constant video stream by sending a 250KB picture at a 25 fps rate. As a result, the bandwidth consumed by the application reached 50Mbps, that is, a 50% of the maximum network capacity. The second node contained just the TestTask, which stored the time value when activation messages arrive. Video messages were discarded by a simulated video receiver task. All the nodes were connected using the Ethernet switch.

The elementary cycle (EC) used at the Orchestrator was set to 5ms. The test was executed twice, once with the video camera sending data without FTT Event Service and a second time with the FTT Event Service activated. Both tests were executed for 120000 EC, corresponding to 10 minutes. Then, the Ethernet switch was replaced by the Ethernet hub, and both tests were repeated.

The results, depicted in figure 9, show that the activation jitter measured in the TestTask without FTT Event Service and with FTT Event Service does not vary significantly using the Ethernet switch, being kept below 500µs. However, when using the Ethernet Hub layout, the usage of FTT Event Service greatly improves the performance of the middleware, as not using it increases the jitter up to 3ms, while activating it makes the jitter come back to normal. These results prove not only that FTT Event Service prevents interference with the main middleware, but also that the solution is applicable in soft real-time scenarios.

5.3. Traffic Prioritization Test

The second main design objective of the FTT Event Service is to allow traffic prioritization in applications managed by FTT-CORBA.

Traffic prioritization is especially important in real-time application domains, such as industrial applications, where different types of traffic share the network. The FTT Event Service handles this problem through topic priorities.

To validate the traffic prioritization mechanism the test case described in figure 8 was designed. For this test three BeagleBoards were used, each with a different role. One of the nodes simulated video camera that acquired and sent video frames at 25 fps rate as in the previous test. The second board run a sensor simulator that provided data periodically every 40ms. Along with the sensor simulator, the second node executed an alarm generator task that randomly sent alarm messages to the event channel. The time lapse between two alarm activations follows a [3,7] uniform distribution (in seconds). Lastly, the third board ran a test task that logged the time latency introduced by the FTT Event Service. It is important to note that, for this test, some extensions were developed to the FTT Event Service to synchronize the clocks of all the federated channels and to enable timestamps in the exchanged messages; however, these extensions are not part of the main FTT Event Service, but they are only intended for testing purposes.

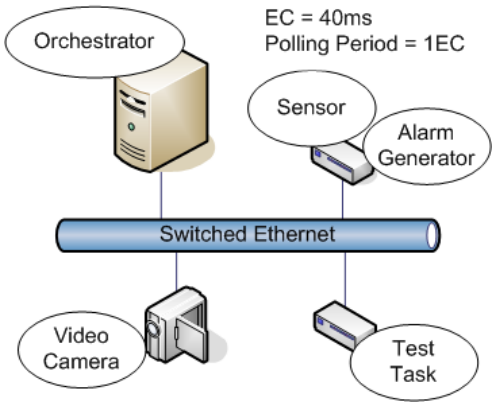

Figure 8. Traffic prioritization test

All the nodes were connected using the Ethernet switch. For the test the video traffic was given the lowest priority value, the sensor data traffic a medium priority value and the alarm traffic the highest priority value.

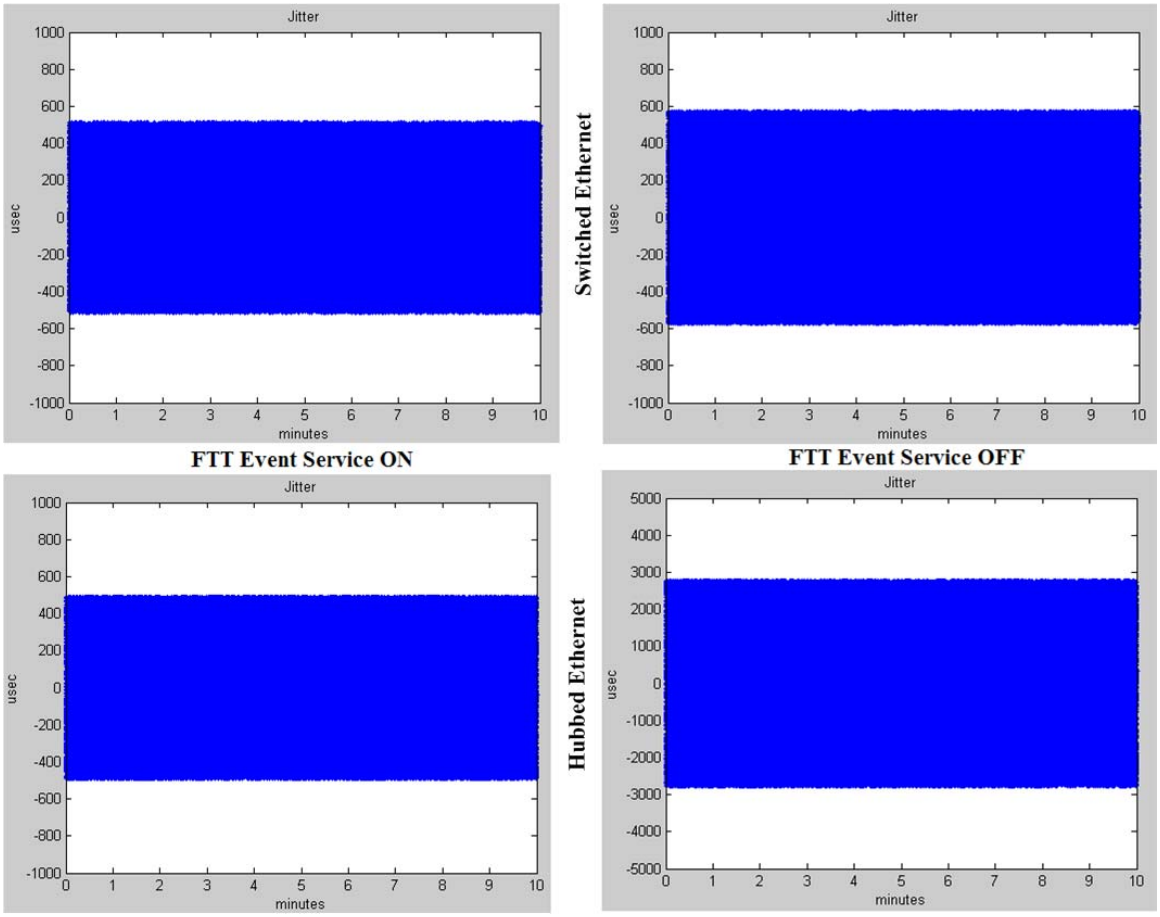

Figure 9. Synchronization test results

In this second test the EC was set to 40ms, and the Event Channel was configured to poll the federated channels every EC. The test lasted 20 minutes.

The results of this second test are shown in table 1. The results show that the FTT Event Service succeeds in handling the different types of traffic proposed in the example. Indeed, the mean latency introduced by FTT Event Service to the higher priority messages is lower and, what is more important; the standard deviation of these latency values is also much lower.

Table 1. Latencies (μs) per traffic type

	Video Data	Sensor Data	Alarms
Minimum	82830	4025	8175
Mean	163860	10971	9682
Maximum	171340	97971	11960
Std. Dev. (σ)	**4781**	**3578**	**1639**

6. Conclusions and Future Work

In this paper the authors have presented the FTT Event Service, an extension to the FTT-CORBA middleware architecture that enables the integration of data communications traffic in the distributed system without affecting the strict time synchronization provided by the middleware. The proposed solution provides a prioritized event channel integrated with the FTT-CORBA architecture. This channel generates time windows in which the distributed nodes may send their data messages without interfering with the activation messages that the Orchestrator sends to the distributed nodes to activate their tasks.

The presented service has been implemented using a simple API, similar to the one used in the CORBA Event Service standardized by the OMG, even though its functionality is slightly different.

To validate the proposed approach two experiments were developed. The first experiment proved that the presented service controls the jitter increase derived from collisions between data and FTT-CORBA activation messages when the communication medium is shared among the nodes. The second experiment demonstrated that the proposed service is capable of managing prioritized traffic under high network utilization scenarios; and therefore, that it is valid for soft real-time industrial distributed applications.

In the future the authors will explore other implementation alternatives that improve the

performance of the service with ECs below 1ms. Moreover, the authors plan to study the performance of the FTT-CORBA and FTT Event Service together in wireless communication media.

The work presented on this paper has been contributed to the FTT-CORBA project [26], and it is available as open source software.

Acknowledgements

This work has been financed partly by the ARTEMIS JU through the CRAFTERS project (grant no. 295371) and the iLand project (grant no. 10026), the MICINN through the DPI2009-0812 project and the Basque Government through the S-PE11UN061 SAIOTEK project.

References

[1] Skeie, T., Johannessen, S., Holmeide, O. "Timeliness of real-time IP communication in switched industrial Ethernet networks", IEEE Transactions on Industrial Informatics, Vol. 2, No. 1, pp. 25-39, Feb. 2006.

[2] OMG, Object Management Group, "Common Object Request Broker Architecture: Core Specification", Version, 3.0.3, March 2004.

[3] Sanz R., M. Alonso, "CORBA for Control Systems", Annual Reviews in Control, No 25, 2001, pp. 169-181

[4] Henning, M.,"A new approach to object oriented middleware", IEEE Internet Computing, Vol 8, Issue 1, pp. 66-75, 2004.

[5] OPC foundation, http://www.opcfoundation.org/.

[6] Jammes, F., Smith, H., "Service-oriented paradigms in industrial automation", IEEE Trans. Industrial Informatics, vol. 1, issue 1, pp. 62-70, Feb. 2005.

[7] OMG, Object Management Group, Data Distribution Service for real-time systems, version 1.2, 2007.

[8] Ryll, M., Ratchev, S., "Application of the Data Distribution Service for Flexible Manufacturing Automation", Proc. of World Academy of Science, Engineering and Technology, vol. 31, July 2008, pp. 178-185.

[9] Object Interface Systems, Inc "It's not your father's CORBA", Available at http://www.windriver.com/ announces/ad-rc-2011/documents/OIS_Not-Your-Fathers-CORBA_WhitePaper.pdf", OIS whitepaper, 2011.

[10] Román, I, Madinabeitia, G., Jimenez, L., Molina, G.A. Ternero, J.A. "Experiences applying RM-ODP principles and techniques to intelligent transportation system architectures" Computer Standards and Interfaces, Available online 2 January 2012

[11] Amoretti, M., Reggiani, M., "Architectural paradigms for robotics applications", Advanced Engineering Informatics, Volume 24, Issue 1, January 2010, Pages 4-13

[12] Calvo, I., Almeida, L. Noguero, A. "A Novel Synchronous Scheduling Service for CORBA-RT Applications", 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, ISORC07, pp 181-188, Santorini, Grecia, 2007.

[13] Noguero, A., Calvo, I., Almeida, L., "The design of an Orchestrator for a middleware architecture based on FTT-CORBA" In proceedings of the 6th Iberian Systems and Information Technologies Conference, CISTI, June 2011.

[14] Calvo, I., Almeida, L., Perez, F., Noguero, A., Marcos, M. "Supporting a reconfigurable real-time service-oriented middleware with FTTCORBA" In proceedings of Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference, IEEE Computer Society, 2010, 1 – 8

[15] Pedreiras, P., Gai, P., Almeida, L., Buttazzo, G. C. "FTT-Ethernet: A Flexible Real-Time Communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems", IEEE Transactions on Industrial Informatics, 2005, 1.

[16] Ramani, S., Trivedi, K.S., Dasarathy, B. "Performance Analysis of the CORBA Event Service Using Stochastic Reward Nets" Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems, IEEE Computer Society, 2000.

[17] Gore, P., Cytron, R., Schmidt, D., O'Ryan, C. "Designing and Optimizing a Scalable CORBA Notification Service", In Proceedings of the Workshop on Optimization of Middleware and Distributed Systems, 2001, 196-204.

[18] Harrison, T. H., Levine, D. L., Schmidt, D. C. "The design and performance of a real-time CORBA event service", SIGPLAN Not., ACM, 1997, 32, 184-200.

[19] ZeroC Inc. "Icestorm manual", version 3.4.2. Available online at: http://doc.zeroc.com/display/Ice/IceStorm

[20] Patcas, L.M., Murphy, J., Muntean, G.M. "Middleware support for data-flow distribution in web services composition" In the PhDOOS Workshop and Doctoral Symposium, 2005.

[21] Hoffert, J., Schmidt, D. C., Gokhale, A. S. "Adapting Distributed Real-Time and Embedded Pub/Sub Middleware for Cloud Computing Environments" Middleware, 2010, 21-41.

[22] Gill, C. D., Cytron, R. K., Schmidt, D. C. "Multi-Paradigm Scheduling for Distributed Real-Time Embedded Computing", IEEE Proceedings, Special Issue on Modeling and Design of Embedded Software, 2002.

[23] Losert, T. Extending CORBA for hard real-time systems, Ph.D. Thesis. Vienna University of Technology. 2005.

[24] Lankes, S.; Jabs, A.; Betriebssysteme, L. F. & Reke, M. A Time-Triggered Ethernet Protocol for Real-Time CORBA Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, IEEE Computer Society, 2002.

[25] Kim, K. H. (K.; Liu, J. Q.; Miyazaki, H. & Shokri, E. H. TMOES: A CORBA Service Middleware Enabling High-Level Real-Time Object Programming Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems, IEEE Computer Society, 2001.

[26] FTT-CORBA project webpage: http://sourceforge.net/projects/fttcorba/

[27] Noguero, A., Calvo, I. FTT-Modeler: A support tool for FTT-CORBA. In proceedings of the 7th Iberian Conference on Information Systems and Technologies, CISTI 2012.