# XXXIV JORNADAS DE AUTOMÁTICA

*Terrassa, 4-6 de septiembre 2013*

## Actas

# EVENT DRIVEN MIDDLEWARE FOR DISTRIBUTED SYSTEM CONTROL

Manuel Muñoz, Eduardo Munera, J. Francisco Blanes, José E. Simó, Ginés Benet
Institute of Control Systems and Industrial Computing
Polytechnic City of Innovation
Polytechnic University of Valencia, Spain
{mmunoz; emunera; pblanes; jsimo; gbenet}@ai2.upv.es

## Abstract

*This paper presents a control kernel based middleware description. Capabilities and developed functionality are enhanced. The establishment of a event-based middleware control system offers a more reliable and efficient way to perform control tasks by ensuring a proper distribution of the requirements, actions and services between the system devices according to their availability of resources. Middleware is able to perform its capabilities over a distributed or partitioned system allowing to the programmer the control application design from a topology-independent point of view of the whole system. Quality of service (QoS) concept are introduced in data exchange providing a real-time communications between nodes. data distribution system (DDS) is used as base to achieve this reliable communications. A user interface has been designed to help the programmer in the middleware configuration task.*

**Key words:** Real Time Systems, Embedded Systems, Distributed Control Systems, Control Kernel.

## 1 INTRODUCTION

distributed control systems (DCSs) are used extensively in industry to monitor and control distributed equipment with remote human intervention. Furthermore, civil engineering applications, or even usual consumer products, require or adopt a distributed platform as solution. In general, DCSs are real time embedded control systems and they are usually constrained by limited resources as computational power, memory, or network-bandwidth.

DCSs achieve a correct solution for most of the control scenarios. But technology changes drive its evolution by modifying its traditional morphology and design criteria. While computing power of embedded systems is increasing over time, moving the ad-hoc node implementation the to powerful embedded computers, networking technology trend is to move from control-specific networks to Ethernet based infrastructures and wireless communication. Even the spectacular computing power increment allow to partition a embedded system, obtaining different criticity subsystems with added benefits such as isolation and safety.

This paper is organized as follow: In Section 2 is introduced an approach to the control kernel (CK) and middleware (MW) architecture. Section 3 decribe the proposed topology for a distributed embedded control kernel while the middleware main services and features are described along Section 4, which manage to get an overview of the proposed system. In Section 5 is presented the human middleware interface and its functionalities for seting the services and manage middleware configurations. Finally, in section 6 is presented the conclusions which summarize the presented system and the benefits of proposed topology.

## 2 CONTROL KERNEL CONCEPT

The availability of a robust kernel or middleware can provide functionalities, services and interfaces to the hardware resources of the system. This paradigm leads to the specification of the CK concept as is presented on [1] which achieves an user transparent design that offers an execution management interface for the control application.

We consider a scenario where the control system components (sensors, controllers, operator terminals and actuators) are often spatially deallocated offering a distributed perspective, in which all of them are interconnected through a network. Even in the case of a mixed criticality system, composed by a set of partitions running on a virtualized environment, some kind of communication between components is a must.

CK interface is composed by a set of services that helps the designer to configure the control application and define the basic and prioritized control activities involved. Also can guarantee autonomous operation at the same time as guarantee fault tolerance and safety. The design of a flexi-

ble system can offer reconfigurability and hybrid control solutions.

The main objective of the CK development is to allow control application adapt to the growing complexity and functionality which impact control and real time embedded systems having to extends its services to the demand of new applications. That paradigm moves away from classical control sytems like PLC arrangements which are usually less evolved and offer poor programing capabilities. Instead of this it takes the advantage of the characteristics provided by microcomputers like DSPs which are optimized for high speed and real time data processing and also creates a more profitable way for software reuse.

The design of a CK improves the reliability of control activities execution by implementing a task model and execution containers. This will help to divide control operations into subtasks, which also can be isolated locally or distributed in several nodes. Task partitioning is performed by the use of code delegation techniques and adaptive configuration.

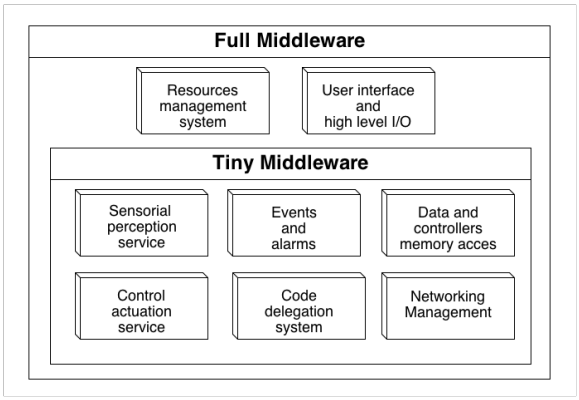



Figure 1: Control Kernel Blocks

An application running on top of control kernel middleware (CKM) is isolated from the operating system (OS) and the middleware is responsible of offering funcionalities and abstractions related to the software control components. It also have to provide real-time services (strict or not), management of the control tasks, access to its physical surrounding (sensors, actuators and plants), or communication management in both local and distributed way.

## 3 DISTRIBUTED EMBEDDED CONTROL KERNEL

Distributed Systems usually applies middleware technology in order to provide communication services and data exchange. There are available many middleware solutions: Java Message System (JMS) [5], Internet Communication Engine (ICE) [6], Common Object Request Broker Architecture (CORBA) [12], or data distribution system (DDS) [13]. Although there are different perspectives like a Message-Centric (JMS) or Data-Centric (DDS) system all of them aims to provide reliable and flexible services for asynchronous data exchange. JMS is based on the Java Platform and enables distributed communication by adding a common API for the development of message based applications in Java. ICE, CORBA, and DDS all of them make use of an Object Request Broker (ORB) which defines the interface for exchanged objects.The use of an ORB provide end-to-end Quality of service (QoS). Most ORB implementations are based on The ACE ORB (TAO) [17] which, as can be noticed in its name, is built on the Adaptive Communication Enviroment (ACE) framework [16].However the most evolved of them is DDS which is created with the prpose of standardize data distribution frameworks. DDS provides a high performance QoS-based service which offers a very adaptable communication.

A proper QoS setting is a crucial factor for any network connecting nodes with distinct available resources, so services must have the capability to adapt different system restrictions. For that reason there must be taken into account the differences among the management of QoS in each system. In this way [14] introduced a survey for QoS support on some communication systems, including the previously mentioned and some more, explaining its own characteristics. Also [7] presents a study of the adaptability and QoS needs of middleware according to the offered typology of event-dissemination in the system.

More extended and specific middleware implementations can be found as Open RObot COntrol Software (OROCOS) [3] which offers a hard real-time framework for control systems providing tools for data exchange and event-driven services. On the other hand some robotic-specific frameworks like Carmen [11] or Robot Operating System (ROS) [15] performs successfully but lack of the desired generality. In every case these frameworks do not provide real-time communication with fault tolerance system.

Proposed solutions aims to offer similar extended services but preventing the unavoidable restrictions found in other frameworks. Adaptability to physical embedded control systems is the key of the system, this goal is reached by exploding the capabilities of event-dissemination and QoS aware real-time communications.

### 3.1 Underlying Topology

From this background, the shape of a distributed control network could be depicted without loss of generality as shown in Figure 2.

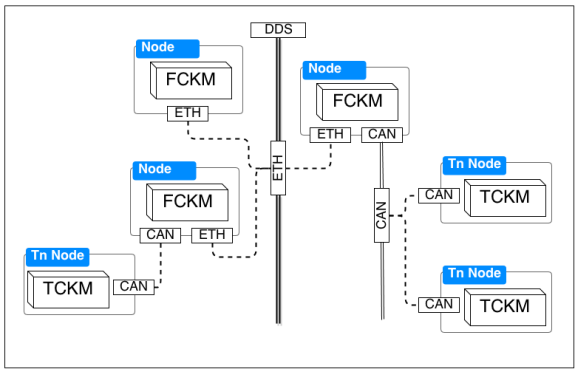



Figure 2: Real Time Control System Topology

Every component in the heterogeneous network could be characterized by its own performance and capabilities. From that, two type of nodes could be distinguished: the less powerful and cheaper (or light) nodes and the powerful service nodes.The difference between these nodes are the capabilities to offer services, according their resources limitation. In the same way, light nodes include a CAN bus connection (playing the role of a low-cost specific bus), while powerful nodes additionally have a ethernet network.

### 3.2 Light nodes and Tiny Middleware

The Tiny-Middleware Control Kernel (TCKM) is designed for working on less performance nodes and offers a cost-effective solution with low power consumption. Usually these nodes are physically closer to the controlled process and are responsible of low level operations as acting/sensing task. Relevant information is exchanged with the service node which, is the responsible of the high level tasks and code delegation. Light nodes also provide safety mechanism in order to mitigate and alert about defective operations through the rise events and alarms and its handlers. In order to detect this situations, execute handlers and raise events, the TCKM includes monitoring tasks supervising the state of the system. TCKM main services can be organized in the following categories:

- Sensorial perception service: This service interacts with sensors in order to provide data at the required sample period.
- Control actuation service: Is in charge of applying the calculated control action and interacts with the plant trough actuators.
- Code delegation: Offers a mechanism of adaptability by performing the delegation of functions from nodes with a high load to idle ones in order to obtain better use of the resources.
- Monitoring and events/alarms system: Provides fault tolerance by the monitoring the available resources and the controllers performance, raising alarms in case of a anomalous situation..
- Data and controllers: Deal to obtain transparency between data in local memory and the distributed objects. It also manage the access and switch between 'controllers pages' reserved in memory.
- Network Management: Manage all the network resources allowing the communication between the system nodes through both ethernet or can.

### 3.3 Service nodes and Full Middleware

The basic TCKM services are extended in the Full-Middleware Control kernel (FCKM) with the aim to expose a richer an interface to the control application. Powerful service nodes which hosts FCKM are the responsible of the execution of most complex control loops in order to provide a high quality control. At the same time the FCKM take profit of the high capabilities of Input/Output (IO) and networking of the Service Node. Resource management employs all the process information in order to adapt the system to the control requirements. Code delegation is the main mechanism to perform this adaptation as will be introduced. The desired control requirements are defined by the declaration of a QoS record which defines the necessary conditions to guaranty the performance and reliability of the designed control. Exclusive FCKM services can be synthesized as:

- Manage and guarantee specified QoS:Relying on DDS QoS mechanisms service nodes must guarantee the fulfillment of the requested policies, informing the system otherwise.
- Resources Management: This service performs an optimal use of the availables resources in the system by the application of code delegation mechanisms.
- Interface control application and high level I/O: As most control systems provides an HMI and SCADA interfaces, proposed system must also provide an interface for the control application and high level I/O for end-user HMI devices.

### 3.4 Virtualized and Partitioned Nodes

Virtualization [4, 9] is a well know technology in the field of powerful computers and servers. The increasing of processing power of embedded processors has opened new possibilities of virtualization application in the embedded systems market. One example of virtualization software, focused on embedded systems development, is the open source project XtratuM [10] used in our prototype.

Talking about embedded control systems it is mandatory to address the problem of putting together different subsystems having different criticality levels. Some application-critical elements are essential for the mission performance because safety reasons (p.e. airplane flaps servo-controllers) while others (p.e. graphical interfaces or comfort control) can be shutdown and restarted with minimal impact on the mission performance. The design trend in Distributed Control Systems is to spread the critical tasks on dedicated computing nodes and then, replicate them. Critical activities run on dedicated nodes in order to avoid side effects caused by faults in other (normally less critical) activities. On the other hand, preventing hardware failures these nodes are 3x replicated sending computing results to some kind of voting unit witch decides the correct result and detects possible failures. Following this design pattern "n" critical activities will need at least "3n" computing nodes in the distributed infrastructure.

On the other hand, virtualization technology provides temporal and spatial isolation properties. Spatial isolation is understood as the inviolable assignment of hardware memory areas and peripherals access to "partitions" running in the same computing node. Temporal isolation is the predictable and inviolable computing time assignment to "partitions" running in the same computing node. The use of virtualization technology lead us to a more simple and efficient engineering pattern running "n" redundant critical tasks on 3 computing nodes holding "n" partitions each. See Figure 3. Different partitions hosted in the same node, can run different Operating Systems and can be loaded and restarted independently. In this way, a partition can be seen as a separated distributed node, but in practice, the communications abstractions and discontinuous temporal profile available at partition level difficult the application of the partitioned design pattern.

Control Kernel Middleware has been extended to support partitioned nodes in a transparent way. Control applications run using Middleware services witch implementation and mechanisms differ with the underlying infrastructure. This independence is one of the key requirements from the earliest implementations of the Middleware.

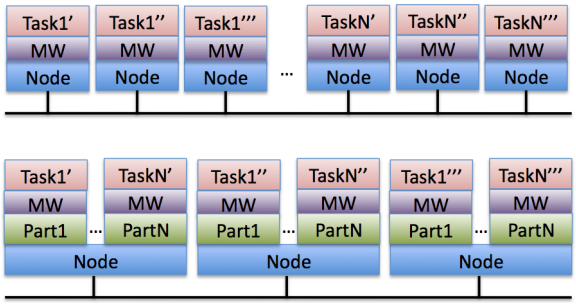



Figure 3: Virtualization

## 4 MIDDLEWARE GENERAL FEATURES

Below will be described some of the major features performed by the proposed CKM. These functions have been grouped around services:

### 4.1 Sensory Perception Service

Sensor measurements are acquired according to the time periods required by the application and suitable for each sensor. Data acquisition delays should not be reflected as control loop delays, so, the instant when data is captured must not affect to the controller computation start neither the instant when control actions are delivered. Its supposed that controllers are calculated for a specified period, the variations on it, known as jitter[2, 8], entails losses in control quality. CKM provide a extrapolated delay-free measure calculated for the instant when control task starts as well as actuation service.

Acquired measures must be adapted from raw sensor values to processed understandable information. This adaptation affects to the capacity of integration between nodes to the point that every element in the system can read and understand any data that has been provided by the data service. That way, CKM provide some primitive functions that allow this adaptation (e.g. raw analog digital converter (ADC) data form 10bits values to -5 to 5 range).

### 4.2 Control Actuation Service

The main provided functionality is to supply a valid control action at each moment to the corresponding actuator. To avoid possible system failures, has been implemented a safety mechanisms that execute a secure routine in case that any control action has arrived from the controller.

In that situations the programmer can choose between several options, ranging from acting with the last available value, to applying a safe-stop maneuver, allowing to force a concrete dynamic performance in the actuation signal.

In this line the service allow to use the algorithm described in [18] where the proposed control model allows to perform a set of basic activities to ensure the safe operation of the controlled system. In order to provide this functionality, a set of future control actions are extrapolated using a generalized predictive control (GPC) controller, and stored by the actuation service. If is not possible to close the loop, then this service will use these stored actions until the data losing problem is solved.

Other functionality offered by this service is the adequacy of the information provided from the data service to a type understandable by the actuator, just as the inverse of the adjustment made by the sensing service.

### 4.3 Monitoring and Alarms System

Monitor is in charge to obtain the information that represents current or previous system states as well as the resource utilization. Mentioned state information can be requested by the application to take appropriate decisions about the actions to perform, like code delegation, mode exchange, disconnection, or any other that has been required.

Additionally an alarm system has been developed offering the application to subscribe a set of states or situations where pretends to be warned by the middleware. This module takes information from the monitor and checks if the system can provide the quality previously agreed, otherwise an specific alarm will be raised.

### 4.4 Resource Management

Resources must be managed and adapted in order to satisfy specified QoS. For that reason different performance QoS levels are established for control application execution. These criteria should be agreed before the execution starts. Also must be defined the conditions for identify high computational load situations in order to perform appropriate adaptation mechanisms. Early actuation on anomalous situations can avoid to reach malfunction situations which can be hard to solve. Moreover, it guarantees a high performance of the system because a node never come under overload, whenever another node in the system presents enough available resources.

#### 4.4.1 Code Delegation

In order to provide the system has been integrated in the middleware a code delegation mechanism. This service allow to exchange sections of code between system nodes. However, only some specific functions or modules are suitable to be exchanged. Delegation of code segments can be performed at runtime allowing its execution at anytime. Main delegated codes are related to a certain action just as a new controller execution or the addition/removal of a sensing process. That functionality allows the middleware to move code from one busy-node to another load-free. Among the criteria for perform these movements of code, not only computational load levels are taken into account, but also all those criteria which improve the use of resources.

#### 4.4.2 Controller Switching

Controllers can be delegated, added, or deleted, but this actuations could entail a unstable state. Middleware guarantees the proper switching between controllers. The suitable switch strategy should be chosen by the control designer. If the previously agreed conditions are not properly respected, actuation service would be responsible for delivering safe control action to the system.

#### 4.4.3 Scheduling and Flexible Tasking

Tasks will be optimally planned based on current control objectives, the quality levels and the amount of resources available. Likewise, we consider in some cases the use of optional tasks that will be performed only when is available an excess of resources. Following this concept control algorithms can be divided into a required and an optional part, being this last an optional task. That implies that high quality control will be performed when system have enough resources and otherwise executing a basic control. Optional tasks can also be delegated to any other node with available resources thanks to the code delegation paradigm.

One last adaptation mechanisms is the design of flexible tasks which can modify its parameter of execution at run time. It is well known that, in general, quality control decreases if the sampling period is increased, but it is also known that the lower the sampling period greater resources will be required. Therefore, if the sensory data availability changes, the task period also should change control for optimization, which imply changes in the parameters of the controller and the information stored. This assures the availability of a control action at anytime, and consequently a safe

operation, although it is forced to discard a high quality control.

### 4.5 Network Management

Being a distributed system communications between nodes are really important. The topology shown in the Figure 2 is taken as reference for this development. In it you can see two different buses that must be handled by the communications service.

The communication between service nodes will rely on a DDS-based implementation which in turn will employ TCP as transport protocol.

Tinny nodes use an ad-hoc protocol over CAN bus, maintaining some QoS properties. A service node is connected as gateway to DDS network, in that way the information in this bus is accessible for the rest of service nodes.

In a partitioned node, communications between partitions are implemented using queuing mechanisms offered by the hypervisor, so that increase the performance compared to the other buses.

In all cases, the middleware is responsible for managing the access to all the information with a common interface.

### 4.6 Data

The middleware provides transparent access to distributed (or partitioned) data in control applications, and between internal and external devices (sensors and actuators), so that distributed components can be accessed regardless of their location. Although this, data must also meet with several QoS. This is a significant step in order to reach the desired reliability of the system. This can be specially remarkable in the dissemination of sensor data in which QoS policies like "Deadline" or "Lifespan" can prevent to compute a mistaken control action using delayed values.

## 5 DEVELOPMENT MODEL

The CK proposed in this work has been implemented as a Middleware, that configures a software layer between the user applications and the OS or hardware abstraction layer (HAL). Middleware offers a set of services to the control application. Every node in the system runs a Middleware instance in the proper version (TCKM or FCKM) depending on their capabilities.

### 5.1 DEVELOPMENT INTERFACE

Control engineer develops the application from the whole system point of view.The aim is to ease as much as possible the application design process. For that reason an user interface has been developed, which allows the user to design the control application avoiding required real time (RT) programming, communications, etc. In Figure 4 can be observed a capture of the configuration and edition application.

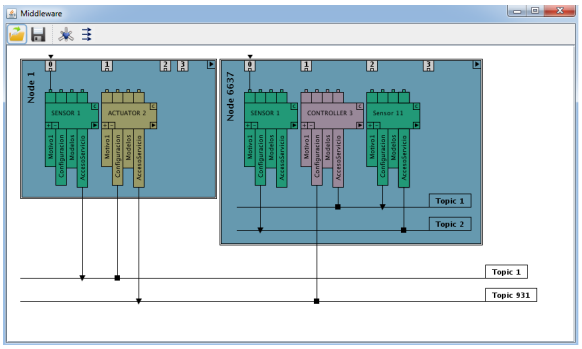



Figure 4: Configuration editor interface

Once the design is fulfilled using the interface, application generates xml files that will be used by the MW to load the application. This files description is shown in section 5.4. Using this process, manual data introduction is avoided, so a big amount of mistakes is prevented. Interface also allows to connect with the middleware at run time for handling the configuration dynamically.

### 5.2 MIDDLEWARE INITIALIZATION

When the Middleware starts up all the services are initialized, but no application is loaded. To load the desired application, the manager module is called to launch the proper services. TCKM or FCKM follow different strategies to achieve this goal:

- In the FCKM nodes, which are able to manage xml files, a initialization procedure is launch, where the configuration is read from the file and applied directly. If none configuration file were provided, the services remain in waiting mode, till some of them are called through the network.

- In every TCKM nodes, who can not manage any file, configuration process is not launch. Instead, a FCKM node reads the configuration and calls the TCKM node using the network interface to request its configuration.

### 5.3 CONFIGURATION MODEL

It has been defined a configuration model based in xsd files which specify all the needed parameters for starting the system offered services and load the desired application. Hierarchical model allow to extract partial configuration from distinct modules and/or components. This model contains precise information about the definition of the required parameters to perform a service request and its configuration.

### 5.4 CONFIGURATION FILES

The configuration is obtained through the use of xml files, who are generated from the model. In the same way as the model, xml files can describe the configuration both totally or partially. Every xml contains valid configurations in any case that is able to be validated itself against the model through standard xml/xsd validation mechanisms.

Once the services has been initialized, at the system start up, the middleware is able to load this xml configuration files. After loading and validating these files are performed the needed middleware calls for setting the described configurations.

In addition, in those cases that the user perform configuration changes during the execution, these changes can be stored within the rest of configuration files for future executions.

Configuration files can be edited in manual mode using a text editor, and following the described criteria in the model. However this edition mode is very unfriendly and can easily lead to errors. For that reason it has been developed an user interface which allows the user to generate and modify configurations in a practical way, without the need of a deep known about the model

## 6 Conclusions

A middleware based on the Control Kernel concept has been developed. This software allows the development of complex control applications. The establishment of a event-based middleware control system offers a more reliable and efficient way to perform control tasks by ensuring a proper distribution of the requirements, actions and services between the system devices according to their availability of resources. That way, the enhancement of the interaction between service and light nodes helps to make use of its own characteristics in a more effective way. Also, DDS-based communication system allows to exploit QoS benefits improving the interaction between nodes by providing a reliable real-time communication, as well as promotes the generation of events. The design of a new user interface facilitates the system design and helps to specify and manage configurations through the use of xml files, as a result, workflow is speed up considerably. An early version of proposed system has been implemented in order to test the viability and performance. Future works will be focused on performance improvements related with the management, generation, and handling of events, and extend the QoS unit parameters.

**Acknowledgment**

This work has been supported by the Spanish Science and Innovation Ministry MICINN under the CICYT project COBAMI: DPI2011-28507-C02-01/02. The responsibility for the content remains with the authors.

## References

[1] P Albertos, A Crespo, and J Simó. Control kernel: A key concept in embedded control systems. In *4th IFAC Symposium on Mechatronic Systems*, 2006.

[2] KE Arzén, A Cervin, J Eker, and L Sha. An introduction to control and scheduling co-design. In *39th IEEE Conference in Decision and Control*, Sydney, 2000.

[3] H Bruyninckx. The real-time motion control core of the Orocos project. *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference*, 2:2766–2771, 2003.

[4] A Crespo, I Ripoll, and M Masmano. Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach. In *2010 European Dependable Computing Conference*, pages 67–72. IEEE, 2010.

[5] M Hapner and R Burridge. Java message service. Technical report, Sun Microsystems Inc, Santa Clara, CA., 2002.

[6] M Henning and M Spruiell. Distributed programming with ice. Technical report, ZeroC Inc., 2003.

[7] SP Mahambre, M Kumar, and U Bellur. A taxonomy of qos-aware, adaptive event-dissemination middleware. *Internet Computing, IEEE*, 11(4):35–44, 2007.

[8] Pau; Marti, Josep M; Fuertes, Gerhard; Fohler, and Krithi; Ramamritham. Jitter compensation for real-time control systems. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 39—-48, 2001.

[9] M Masmano, I Ripoll, and A Crespo. XtratuM : a Hypervisor for Safety Critical Embedded Systems Virtualising technologies overview. In Tecom Project, editor, *11th RealTime Linux Workshop*, 2009.

[10] M Masmano, I Ripoll, A Crespo, and JJ Metge. Xtratum: a hypervisor for safety critical embedded systems. In *Proc. of Real-Time Linux Workshop.*, 2009.

[11] M Montemerlo, N Roy, and S Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference*, 3:2436–2441, 2003.

[12] Object Management Group (OMG). The Common Object Request Broker (CORBA): Architecture and Specification. Technical report, Object Management Group, 1995.

[13] Object Management Group (OMG). *Data Distribution Service for Real-time Systems.* Version 1 edition, 2007.

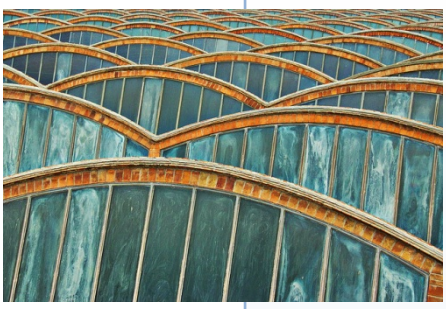
[14] JL Poza-Luján. A Survey on Quality of Service Support on Middleware-Based Distributed Messaging Systems Used in Multi Agent Systems. *International Symposium on Distributed Computing and Artificial Intelligence*, pages 77–84, 2011.

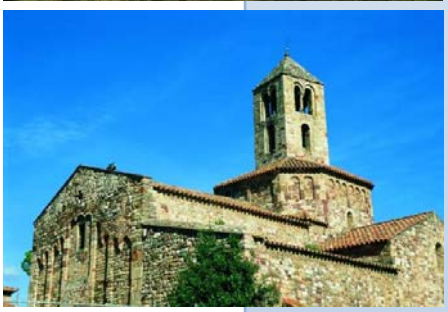
[15] M Quigley and K Conley. ROS: an open-source Robot Operating System. *ICRA workshop on open source software*, 3(3.2), 2009.

[16] DC Schmidt. The adaptive communication environment. Technical report, DOC group, Washington University, 1994.

[17] DC Schmidt. TAO, The Ace Orb. Technical report, DOC group, Washington University, 2007.

[18] R Simarro, J Coronel, J Simó, and JF Blanes. Hierarchical and distributed embedded control kernel. In *17th IFAC World Congress*, 2008.





Patrocinadores:

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

CEA comité español de automática

Ajuntament de Terrassa

Universitat i Societat del Coneixement www.terrassa.cat/coneixement

FUNDACIÓN CEA

GOBIERNO DE ESPAÑA MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

Parrot

MathWorks

GRUPO INFAIMON

PRODEL S.A. EQUIPAMIENTO DIDACTICO

OMRON

Robotnik

TECHNAID LEADING MOTION

ISBN: 978-84-616-5063-7