A Distributed Real-Time Java-centric Architecture for Industrial Systems

P. Basanta-Val, M. García-Valls

Abstract.—There is a trend in industrial systems towards the use of common-off-the-shelf (COTS) components to develop applications that interact with open systems. This trend includes among others the use of high-level languages, such as Java, and Internet protocols (HTTP, and Web Services). Although many industrial systems use these technologies at their business layers, they are far from offering a homogeneous programming platform in their most internal infrastructures. This article extends the current practice by introducing a real-time Java-centric architecture for industrial systems. The architecture integrates existing and upcoming technology to define a Java-based approach. The empirical evidence, included in the article, illustrates the performance of the core of the industrial layer of this architecture.

Keywords—Real-time Java; Middleware; Computer Architecture; RTSJ; DRTSJ

I. INTRODUCTION

Embedded systems are suffering a radical transformation in many aspects. Ancient isolated real-time applications are being interconnected ([1]-[4]) by wireless (Wi-Fi or the new 802.15.4 standards) and traditional wired technologies, resulting in a new generation of distributed real-time applications that expand to a great variety of domains. The revolution does not seem constrained to any particular field ([3]-[8]): military, information, surveillance, and even conservative industrial systems may benefit from the technological revolution.

However, the new generation of embedded distributed systems challenges current infrastructures ([5]) too. In many cases, it must deliver end-to-end real-time performance and offer also an acceptable quality-of-service (QoS) in open systems which must interact at some point with humans. Other systems have batteries that must be preserved over the time, introducing new constraints in the algorithms and models that rule their behavior.

One global technology that may speed the adoption of these new principles in industrial systems is Java [8]. In an industrial environment, Java can be used as a “lingua franca” to fight against heterogeneity stemmed from having different kinds of operating systems and processing infrastructures.

Other advantages stem from the availability of additional programming interfaces for other domains. So far, Java addressed systems that range from the tiny 802.15.4 motes (like SPOT tech [9]) to large application servers (Java EE techs [10]).

Java also offers support for real-time and embedded systems, with commercial products that implement RTSJ (The Real-Time Specification for Java) [11] and promising specifications like DRTSJ (The Distributed Real-Time Specification for Java) [12] that target distributed environments. Together with other technologies such as JINI (i.e., a fault-tolerant discovery framework) [13] and DDS (Data-Distribution Service) [14], they form the base of a rather realistic Java-centric approach that eases the development, deployment, and maintenance of next generation industrial applications.

Unfortunately, real-time Java is a recent technology and its integration in other Java systems, especially those that expand current real-time Java horizons, requires additional exploration and integration decisions. Some of these technologies, like JINI or DDS, lack a clear characterization of the implications that the use of a real-time Java virtual machine is going to have on its interfaces and the changes they require on its current model to profit from the predictability of a real-time Java virtual machine. In other cases, like in DRTSJ, the interfaces were defined and implementations are under development.

In this article, the aim is to contribute a distributed real-time Java-centric architecture—mainly based on existing technology and also on upcoming technology—that helps practitioners reduce the efforts required to support cyber-physical infrastructures for industrial applications. Its main goal is to provide a holistic model that may be particularized or extended to different industrial systems, according to application requirements. The architecture may be adapted to requirements coming from different industrial applications, which may use only a subset of their features.

To introduce the Java-centric architecture, the rest of the paper is organized as follows. Section II introduces the architecture and how different Java technologies map to particular parts of the architecture. Section III evaluates a subset of the architecture. It refers to the performance of the core of the architecture, which helps to offer empirical evidence on the overhead introduced by core elements of the architecture. Section IV deals with other architectures proposed for industrial systems. Finally, Section V ends up
drawing conclusions and exposing ongoing work.

II. DISTRIBUTED JAVA-CENTRIC ARCHITECTURE

A. Requirements

The primary requirements that must be satisfied by the architecture are the following:
- Req1: As a general guiding principle, the architecture must offer connection to the Internet and to other general purpose services and applications.
- Req2: The architecture must offer a flexible industrial operational layer. This layer must be able to interconnect isolated low-level entities to offer added value applications. Besides, it must provide indirect access to the Internet.
- Req3: The architecture must offer access to industrial networks and other legacy factory floor elements.
- Req4: The architecture must provide real-time performance and general quality of service mechanisms.
- Req5: The architecture must deal with the heterogeneity of having different types of infrastructures (e.g. operating systems, and other execution environments).

B. Global Overview of the Architecture

The proposed architecture defines three operational levels, namely: business, industrial and device (Figure 1).
- The business level is in charge of providing a general access to the underlying resources via standard access methods.
- The industry layer is in charge of providing a fully operational layer, which interconnects factory elements to offered high level abstractions. This layer is designed to be much simpler and more predictable than the business subsystem. This intermediate platform holds different libraries and services, which may be omitted to save resources when necessary.
- Lastly, the low-level device layer is in charge of providing factory level access to different networks. This part of the industrial application corresponds to those physical nodes that are closer than others to the physical (industrial) world. Typically these physical nodes consist of sensors or actuators in a factory.

Each level is related to a particular network which interconnects the different elements. The high-level network is the business network (BN), which interconnects general purpose elements (i.e. mail servers, data bases and file servers) to the Internet. The second network: general purpose real-time networks (GPRTN), interconnects special nodes in charge of providing interconnection to different factory floor elements and different industrial services like real-time databases. It is also in charge of controlling the data that are sent from the floor elements to the Internet (devices are not directly connected to the business system; all information should flow via the industrial layer before being sent to the Internet). Lastly, domain machine and sensor networks (DMSN), which are specific to a particular industrial environment (e.g. Profibus and other wired/wireless networks), provide access to control devices.

This physical organization enables the division of programming into three different types of modules: business modules (BM) which are related to the business applications; industrial modules (IMs) which are focused on providing acceptable real-time performance (necessary to interconnect different device elements); and device modules (DMs) which accommodate low-level access to factory elements. Implicitly, this division promotes modules that may be integrated in high-level deployable elements named industrial applications (IAs).

Fig. 1. Overview of the real-time Java-centric architecture for Industrial Systems: layers, networks and technology mappings

1) Use case application

To conclude the overview, Figure 2 shows a use case of a simple application designed for the proposed architecture. The goal of the application is to provide a simple alarm system. Basically, in the example, a temperature alarm flows from a wireless sensor (connected to an 802.15.4 network) an industrial node (DCP) that activates a remote fan (connected to Profibus). First, the data flow from the sensor to the industrial node, which processes the information sent by the sensor. Next, the DCP node decides to notify the business layer about the alarm. Next, the industrial node sends a message which is transferred by the middleware to the business layer and to other subscribed nodes. In this use case, the machine operator is also notified through the Internet and may connect its laptop to monitor the system.

The industrial application (IA) may be modeled as the composition of five modules. One business module (BM) in charge of sending the alarm to the operator; two industrial modules (IMs) in charge of triggering the alarm and reacting to the alarm; and another two device modules (DMs) that provide access to the sensors and fans.
C. DCP Nodes

In the industrial layer, the Java-centric architecture proposes reusable units called DCP nodes (Distributed Cyber Physical) nodes (called DCP-node in Figure 1). The advantage offered by encapsulating industrial applications in DCP nodes is that their code may be easily reused and accessed at the business level and in other developments. Their particular requirements are the following:

- DCP nodes must be able to access factory floor entities and business elements.
- DCP nodes must be able to support a software stack that allows standard communications.
- DCP nodes must be able to offer end-to-end real-time performance in communications.
- DCP nodes must support access to predictable data-bases.
- DCP nodes must offer basic hooks to (predictable) service discovery and life cycle management.
- DCP nodes must offer a general programming space able to host applications that provide added value (built by using different device elements and enterprise elements).

D. Java Mappings for the Proposed Industrial Architecture

This section subsection shows the different technologies chosen to be included in the core of the proposed architecture. For each layer (i.e. business, industrial, and device) the goal is to propose a set of candidate Java technologies.

1) Business layer

The business layer of an industrial application is based on Java EE technology. The use of this technology eases: (i) interoperability with other existing platforms and technologies (through a multiprotocol infrastructure); (ii) the definition of reusable industrial-components that model the business logic of the industrial application; and (iii) the definition of high-availability and load balancing strategies.

In the proposed industrial architecture, the server includes additional support to intercommunicate the operator with the factory elements. To accomplish this goal, there is an asynchronous real-time technology that monitors and shapes the traffic introduced in the GPRTN. This goal can be satisfied by either extending the support given by JMS (Java Message Service) [21] or, more directly, using DDS, which already offers the required real-time support built-in. In both cases, Java EE should not jeopardize the predictability of the industrial subsystem.

The server should also offer some kind of quality-of-service support to the Java EE applications. One choice to provide this support is to use the RTSJ. Currently, RTSJ includes support able to deal with the priority inversion of the garbage collector, and real-time threads to control the amount of processor used by each component. Demanding this support from the platform opens the door to having real-time performance inside the business subsystem.

2) Industrial layer

DCP Java nodes (see Figure 1) may be equipped with software stacks that may include RTSJ, DRTSJ, RT-Jini, RT-O SGi, and DDS, real-time database support, and device driver libraries.

RTSJ and DRTSJ. The first technology supports predictable execution in the local virtual machine. The second extends the offer to networked applications. Both technologies (RTSJ and DRTSJ) collaborate to accomplish a remote invocation (See Figure 3). While RTSJ controls the processor in each Java-node, DRTSJ is in charge of controlling the network and coordinating the execution in remote nodes, according to a certain end-to-end performance (typically, a deadline). Both technologies are crucial to offer a proper end-to-end real-time performance.

![Fig. 2. Temperature alarm detection Industrial Application (IA)](image)

![Fig. 3. Relationship between RTSJ and DRTSJ in DCP Java nodes](image)
application, a service registry, and an execution environment. As RTSJ does with schedulable entities, this service introduces a real-time component characterization which describes each bundle with real-time parameters (like in [26] and [27]). This characterization is used to execute an application according to certain temporal constraints. As Figure 5 shows for a DCP-node, RT-OSGi may run on RTSJ, DRTSJ, and directly on physical resources.

**DDS.** This publisher-subscriber technology enables asynchronous communications between different nodes using queues. In the proposed architecture, DDS is used for two different purposes. On the one hand, it is used to keep in isolation the industrial level from the business layer. On the other hand, it may send messages to other DCP-nodes. Figure 6 shows a node sending a message (step 1 in the figure) to the subscribed events; this message is transferred automatically (steps 3 and 4) to the subscribed nodes by the service (step 2).

**Real-time persistence.** The last technology that takes part in the model is the real-time database (RTDB) technology. RTDBs offer efficient and predictable persistence in those applications that must manage a high amount of data. This database offers (Figure 7) not only persistence but also management of temporal data that is typically generated by sensors. The real-time flavor introduces the technology and models necessary to satisfy deadlines on different operations [28].

**Device drivers.** To access the different protocols and networks of the underlying infrastructure, the DCP node introduces libraries that enable low-level access to various networks and to other industrial systems. Each protocol could be integrated in two ways: (i) using JNI (Java Native Interface) to connect Java bytecodes to C/C++ libraries offered by the manufacturer and, more directly, (ii) using a specific Java API (provided by the manufacturer).

A DCP-node may control the downloading and deployment of libraries using RT-Jini and RT-OSGi. As a general rule, this action should be done using RT-Jini in dynamic and pervasive applications and using RT-OSGi in environments that are more static.

**Integration Levels for the Different Technologies**

<table>
<thead>
<tr>
<th>Table I: Technologies used in each level and its corresponding facilities in the enhanced Model (Summary Table)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level in the architecture</strong></td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>RTSJ</td>
</tr>
<tr>
<td>DRTSJ</td>
</tr>
<tr>
<td>DDS</td>
</tr>
<tr>
<td>OSGi</td>
</tr>
<tr>
<td>Jini</td>
</tr>
<tr>
<td>Database (DB)</td>
</tr>
<tr>
<td>Java EE</td>
</tr>
</tbody>
</table>

Lastly, Table I contains a summary table which describes the relationship among the different technologies used (RTSJ, DDS, OSGi, JINI, DB, and Java EE) and the three levels (D=device, I=industrial, and B=Business) previously described in this section. For each level-technology peer, the table shows whether the technology is included (Y) or not (-), or whether it is an optional element (O). It also describes two integration levels (L0, and L1-L2) for each technology.

The basic integration level (L0) does not require complex changes in existing technologies (basically, it requires being accessible from the application). Extra features are required at
higher integration levels (L1-L2) (explained previously in the article and summarized in Table I). The advantage stemmed from dividing the architecture into different integration levels is to allow different industrial applications to choose their corresponding integration level, according to their requirements.

III. PERFORMANCE EVALUATION WITHIN THE INDUSTRIAL LAYER

This section is focused on the central part of the architecture: the industrial layer, which is the core of the Java-centric architecture. The experiments are focused on the cost of the remote communications among different real-time nodes by using distributed real-time Java. Other technologies like Java EE, RT-OSGi, real-time RT-Jini are set aside of the evaluation because they do not have tight real-time constraints such as those existing in end-to-end remote communications. The specifics of the different device level technology were also set aside in the evaluation to focus the evaluation on the DCP Java nodes.

The evaluation of the performance offered by DCP nodes is also relevant for applications that must use different DCP nodes to communicate with an application (such as the application described in Figure 2, which has two DCP Java nodes). This performance establishes operational limitations to the type of applications that may designed with the proposed architecture.

In this evaluation scope, the two main goals of the performance evaluation section are:

- To provide a metric of the overhead of the communications. The metric is useful to provide a mechanism that allows comparing different software implementations under similar conditions. To accomplish this goal, the evaluation considers the number of remote invocations for the worst-case. The standard cost of the logic in the server is also taken into account with low, medium and high overhead.

- To evaluate the overhead, in terms of % of the blocking time introduced by the benchmark applications within an application. This metric complements the previous one with overhead information. It is also valid to evaluate and compare the performance of different DCP Java nodes under similar application conditions.

The performance is measured by using the specification of an AUTOSAR application tested derived from an industrial application previously proposed in [29]. It contains test cases that have operational frequencies that range from 1 Hz to 250 Hz.

Two different stacks were evaluated on the industrial benchmark acting as DCP Java nodes. The first is based on DREQUIEMI ([30]-[36]): an academic framework for distributed real-time Java applications based on RMI and an optimized RTSJ implementation. DREQUIEMI is a good exponent of a RTSJ-DRTSJ (Level 1) technology support for the industrial layer. Each node runs on a 768 MHz machine equipped with DREQUIEMI. Currently, DREQUIEMI runs only on a 2.47.-TIMESYS-3.1.214 kernel and RTSJ-RI (the RTSJ-VM of TimeSys). The second type is based on an industrial stack for RTSJ-RMI Oracle JRTS 2.2 [37], running on 800 MHz processors, executing a real-time 3.0 Linux kernel. The Oracle JRTS stack is a good example of RTSJ-DRTSJ (Level 0) technology support for the industrial layer. Both types of nodes use the same network infrastructure (100 Mbps Switched-Ethernet).

A. Throughput Performance

The first experiment considered the use of DREQUIEMI and/or Oracle JRTS as a technology on which to deploy applications. The goal is to find the technological limits of both technologies when they support different application modules. The metric used in the experiment is the number of remote invocations carried out; the higher this number is, the better the performance delivered. The benchmark quantifies the maximum number of remote communications that can be carried out per activation (i.e. in each task period). The maximum number of invocations increases as the frequency decreases and when the data/work (i.e. low, medium and high) carried out at the server decreases. Taking DREQUIEMI _96g as reference, low remote invocations require 100μs for their computation at the server. Medium remote communications take 200μs and high remote communications take 400 μs.

| TABLE II: COMMUNICATIONS PER SECOND USING DREQUIEMI DCP-JAVA NODES, 786 MHz-100MBITS-786 MHZ. |
|------------------|-----------------|-----------------|-----------------|
| Freq./data       | Low             | Medium          | High            |
| 1Hz              | 833             | 625             | 555             |
| 10Hz             | 83              | 62              | 55              |
| 16.7Hz           | 50              | 37              | 33              |
| 20Hz             | 41              | 31              | 27              |
| 33.3Hz           | 25              | 18              | 16              |
| 83.3Hz           | 10              | 7               | 6               |
| 0.10kHz          | 8               | 6               | 5               |
| 0.20kHz          | 4               | 3               | 2               |
| 0.25kHz          | 3               | 2               | 2               |

| TABLE III: COMMUNICATIONS PER SECOND USING ORACLE JRTS DCP-JAVA NODES, 800 MHZ-100MBITS-800 MHZ. |
|------------------|-----------------|-----------------|-----------------|
| Freq./data       | Low             | Medium          | High            |
| 1Hz              | 1773            | 1054            | 876             |
| 10Hz             | 177             | 105             | 87              |
| 16.7Hz           | 106             | 63              | 52              |
| 20Hz             | 88              | 52              | 43              |
| 33.3Hz           | 53              | 31              | 26              |
| 83.3Hz           | 21              | 12              | 10              |
| 0.10kHz          | 17              | 10              | 8               |
| 0.20kHz          | 8               | 5               | 4               |
| 0.25kHz          | 7               | 4               | 3               |

Results (Table II and Table III) show that all nodes may communicate with a remote node within its period at least once. Low-frequency/overhead tasks (1Hz) may carry out hundreds of remote communications within a period, while those applications with the highest frequency (0.25 kHz) can carry out 2 or 3 invocations only, depending on the type of effort required from the server.

The two tables are useful for a Java-centric practitioner to
evaluate which of two stacks is the most efficient or provides acceptable behavior (in conformance to design parameters). In this particular evaluation, the performance of Oracle JTRS is higher than the performance offered by DREQUIEMI. Notice that the Oracle JRTS stack runs on a real-time 3.0 Linux kernel a real-time Java Hotspot virtual machine, whereas DREQUIEMI uses a non-commercial solution (the TimeSys reference implementation). However, depending on the kind of the addressed industrial application, the practitioner may opt for the less efficient solution (e.g. when both are able to meet the required performance).

B. Blocking due to Communications

Previous section experiments offered an insight of the throughput offered by DCP Java nodes. This section extends previous results considering the interference, in terms of normalized blocking time, introduced by the remote communications. In the benchmark three cases were taken into account: (i) each task activation requires a single end-to-end communication; (ii) each activation requires two communications; and (iii) each activation requires four communications. The goal of the benchmark is to evaluate the amount of time (%) spent in communication processes including sending time, computation time at server (low, medium and high) and receiving time. The lower this time, the better the performance offered by an implementation.

The results obtained after the evaluation are shown in Figure 8 for the two infrastructures analyzed in this section (DREQUIEMI and Oracle JRTS). The results show some combinations that are not feasible (e.g. all that require 4 communications are not possible at 0.25kHz) using DREQUIEMI on 786 Mhz machines. As in previous experiments included in this section, Oracle JRTS outperforms DREQUIEMI playing the role of an industrial stack (i.e., the performance is higher).

From the point of view of the practitioner, the benchmark allows a Java-centric architect to choose a proper combination in terms of overhead. If the design rule is that the overhead due to communications should be at most 10%, the practitioner may evaluate which of the alternatives is valid from the point of view of performance.

IV. RELATED WORK

The related work is focused on different pieces of work that refer to open current industrial systems and the use of common-off-the-shelf technology in the development of these systems. For each main approach, the section compares the Java-centric approach (explained in Section II) against each approach.

Brathall et al. [19] stated the challenges of producing efficient industrial IT architectures. Their approach to deal with these challenges is the Aspect Object Model (AOM). Our approach offers a predictable Java infrastructure useful to host AOM applications, complementing the AOM approach with real-time performance.

JSR-7 (Java Specification Request number 7 [15]) was headed by Cyberonix. Together with other members of the automation industry, they proposed a Real-Time Cyber Technology (RTCT) which relies on the use of a standard protocol: Enterprise Common Protocol (ECP). The approach depicted by the community process is the construction of a cyber-world for industrial systems [16]. The DCP-Java node described in the Java-centric approach materializes their ideas on a COTS architecture which may be deployed in heterogeneous frameworks.

Cyberonix looked at the concepts of industrial systems from the Internet perspective. This solution has been applied to petroleum retail operations in the eGasStation architecture [16]. Our approach complements this cyber-vision with an architecture where each element offers predictability via real-time Java support.
Urdaneta et al. [20] proposed a reference software model in the petroleum industry context. Their architecture is Java-centric; it is based on a J2EE application server. Our approach is similar to the solution proposed by Urdaneta: both architectures share an enterprise layer and target industrial applications. Nevertheless, the architecture proposed enables predictability directly from Java code by means of RTSJ; a key feature not considered by Urdaneta et al. Therefore, their work may benefit from the Java-centric approach model to extend and generalize its industrial layer.

García-García et al. [22] introduced the Aspect Integrator Platform (AIP). Our approach complements the XML-based defined in [22], offering a real-time Java-centric platform in which applications are deployed. Our architecture may benefit from the xml transformations defined in [22] to provide high-level system operations.

SIRENA (Service Infrastructure for Real-Time Embedded Networked Applications) extends the power of service oriented paradigms (SOA) to industrial automation [23]. Although the definition of a SOA access is out of the scope of this paper, our Java-centric architecture contains many functional elements that may support SOAs. For instance, SOAs may benefit from the predictability of a real-time Java virtual machine, and a holistic model on which SOAs may interact with other SOAs within the industrial system. Likewise, our proposed architecture may expose SOAs at its business level (EJBs with SOA access).

More recently, in the BlueWonder project, Oracle implemented a prototype in an industrial PC (equipped with its real-time Java virtual machine, Solaris™ and Profibus) to control an industrial system ([18]). Its demo, called Sydney, is made up of around two hundred different elements, which are managed by a single industrial PC. Our architecture extends BlueWonder with the possibility of having several Java-nodes that carry out coordinated actions by using industrial networks and the Internet.

Other architectures, such as the one proposed by Thramboulidis et al. [24], go a step beyond proposing the adoption of techniques that belong to the semantic web in the industrial automation context. Essentially, their approach uses a web service-based engine as development platform for industrial automation applications. Although using semantic web techniques is out of the scope of the paper, our architecture may host web-services at its business level.

These efforts, with the notable exception of BlueWonder, have used SOA (SOAP, Web Services and ontologies) or enterprise layers (Java EE) in their developments. The use of real-time Java (RTSJ) is constrained to specific developments (e.g. in the BlueWonder project) and has not been extended to a networked industrial framework yet. This is exactly the lack that our holistic Java-centric architecture mitigates shaping a real-time Java centric architecture for industrial systems based on Java technology.

V. CONCLUSIONS AND FUTURE WORK
A greater rapprochement between current Information Technologies (ITs) and current Industrial Systems (ISs) is still necessary to enhance current and future industrial infrastructures. Fortunately, current infrastructures may benefit from solutions previously proposed in the Internet. The proposed approach, which advocates an intensive use of Java technologies, helps by bridging the gap between the old isolated and specific industrial networks and the fully open model of the Internet. From the point of view of an industrial system, this intermediate level maintains interesting properties; it is able to deliver real-time performance, and it is designed to cooperate with the Internet.

This article has contributed a real-time Java-centric architecture for industrial applications which may be particularized depending on application needs or extended via standard Java technology. The empirical evaluation showed the performance of the core of this technology via the evaluation of two different Java stacks. They have illustrated certain performance parameters that industrial practitioners may expect for their applications when they opt for a real-time Java-centric approach.

Our ongoing work focuses on the evaluation of several 802.15.4 motes (SPOTs [9]) from the point of view of the Java-centric architecture described in this article. In addition, the authors explore service composition and multimedia application (see [38]-[42]).

ACKNOWLEDGMENT

The authors want to thank the anonymous reviewers of the article for improving the quality of the manuscript via constructive contributions. This work has been partly funded by the ARTEMIS Call1 project iLAND (ARTEMIS-JU 100026) funded by the ARTEMIS JU and the Spanish Ministry of Industry, Commerce, and Tourism. Also, this work has been partly funded by national REM4VSS project (TIN 2011-28339) and by regional e-Madril (S2009/TIC-1650) projects.

REFERENCES


Pablo Basanta-Val was born in O Valadouro, Lugo, Spain. He received the Telecommunication Engineering degree from the Universidad de Vigo, Spain, in 2001 and the Ph.D. degree from the Universidad Carlos III de Madrid, Madrid, Spain, in 2007. Currently, he is an Assistant Professor at the Telematics Engineering Department, Universidad Carlos III de Madrid, and he is member of the Distributed Real-Time Systems Lab. His research interests are in Real-Time Java technology and general-purpose middleware used to support next generation industrial applications.

Marisol García-Valls was born in Castellón de la Plana, Spain. She received the Computer Science Engineering degree from the Universidad Jaume I de Castellón, Spain, in 1996 and the Ph.D. degree from the Technical University of Madrid, Madrid, Spain, in 2001. Currently, she is a Professor with the Department of Telematics Engineering, Universidad Carlos III de Madrid. Since 2002, she has been the Head of the Distributed Real-Time Systems Lab of the GAST Group. Her research is focused on resource management for distributed embedded real-time system. She has been enrolled in a number of European, national, and regional projects: ARTIST, ARTIST2, and ARTISTDesign NoE EU projects.