A Real-time Service-Oriented Architecture for Industrial Automation

Tommaso Cucinotta, Antonio Mancina, Gaetano F. Anastasi, Giuseppe Lipari
Real-Time System Laboratory, Scuola Superiore Sant'Anna, Pisa, Italy
{t.cucinotta, a.mancina, g.anastasi, g.lipari}@sssup.it
Leonardo Mangeruca, PARADES, Rome, Italy
leonardo@parades.rm.cnr.it
Roberto Checcozzo, Fulvio Rusinà, COMAU, Grugliasco (To), Italy
{roberto.checcozzo, fulvio.rusina}@comau.com

***Abstract*—Industrial automation platforms are experiencing a paradigm shift. New technologies are making their way in the area, including embedded real-time systems, standard local area networks like Ethernet, Wi-Fi and ZigBee, IP-based communication protocols, standard Service Oriented Architectures (SOA) and Web Services. An automation system will be composed of flexible autonomous components with Plug & Play functionality, self configuration and diagnostics, and autonomic local control that communicate through standard networking technologies.**

However, the introduction of these new technologies raises important problems that need to be properly solved, one of these being the need to support real-time and Quality of Service (QoS) for real-time applications. This paper describes a Service-Oriented Architecture enhanced with real-time capabilities for Industrial Automation. The proposed architecture allows for negotiation of the QoS requested by clients from web services, and provides temporal encapsulation of individual activities. This way, it is possible to perform an a-priori analysis of the temporal behaviour of each service, and to avoid unwanted interference among them. After describing the architecture, experimental results gathered on a real implementation of the framework (which leverages a soft real-time scheduler for the Linux kernel) are presented, showing the effectiveness of the proposed solution. The experiments were performed on simple case studies designed in the context of industrial automation applications.

***Index Terms*—Industrial Automation, Real-Time Embedded Systems, Service-Oriented Architectures**

I. Introduction

The factory automation industry is slowly but steadily experiencing a paradigm shift. The increasing demand for efficiency in machine retooling and commissioning to reduce time-to-market of new products requires a drastic improvement in efficiency throughout the design chain, from process engineering to field tests. A reasonable way to improve this efficiency is to leverage the deployment of new hardware and software technologies, such as embedded real-time systems, standard networking protocols and Information and Communication Technologies (ICTs). Furthermore, the possibility, in the factory automation context, to reuse open standards, protocols, network infrastructures and software components that are widely used in general- purpose ICT application areas, is becoming increasingly appealing, due to their support for Quality of Service and low costs of deployment.

Unfortunately, some technological barriers are preventing the deployment of such technologies in current industrial practise. A critical bottleneck in process efficiency and flexibility of current systems is represented by the networking infrastructure. Today, many communication networks adopted for process automation are still proprietary, designed for collecting I/O data from the field, even though open standardised protocols (Modbus, Profibus, Ethernet variants) are making inroads. The adoption of an open network infrastructure, with the ability to provide Plug & Play services and the capability to hide the devices complexity, provides a simpler and more natural work-flow from the mechanic engineer to the control engineer, allowing for the adoption of the same platform in the identification of the objects and their properties.

Also, reconfiguration of an industrial plant suffers of a set of inefficiencies that are related, among other factors, to the lack of a sufficient level of "intelligence" embedded directly within the components. In fact, these usually exhibit a passive behaviour and are controlled by a centralised Programmable Logic Controller (PLC). Such an approach requires a change in the PLC code at each reconfiguration, which limits modularisation and interoperability. On the other hand, increased efficiency, configurability and monitoring capabilities may be obtained by distributing such functionality as self-diagnostic, self-configurable, and local real-time control within the components, or within embedded micro-controllers close to them.

Clearly, the increased complexity both at the networking and at the component level, where the same set of physical resources (network links and micro-controllers) are shared for functionality related to the support of both remote high-level control, monitoring and reconfiguration, and local, low-level, control logics, needs appropriate design to ensure the appropriate temporal isolation degree between such activities.

a) Paper Contributions: This paper presents a software infrastructure for industrial automation, that leverages widely used open technologies in the domain of general-purpose systems, such as Service Oriented Architectures (SOAs), Ethernet-based communications and real-time technologies.

Specifically, the envisioned architecture is based on: Ethernet-based communications embodying TCP/IP networking capabilities with real-time traffic management; SOAs for easing the problem of identification, discovery and communications among networked components, where the WS-Agreement protocol has been extended in order to support attributes related to real-time and QoS of individual activities, to allow for the configuration of the system at run-time; a mod-

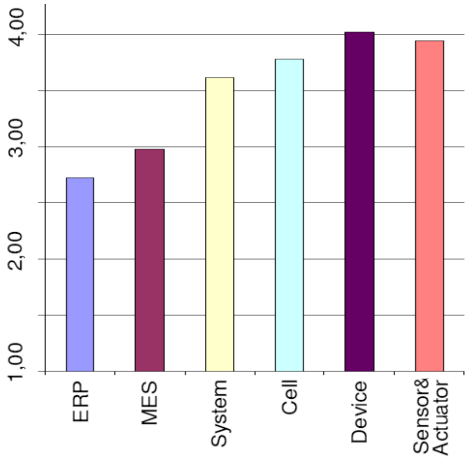


Figure 1. Importance of Plug & Play technology foreseen for year 2010 by interviewed experts, in the different control layers of a manufacturing plant.

ified Linux kernel supporting real-time scheduling strategies for the purpose of achieving temporal isolation between high-level software infrastructures and low-level control logics.

b) Paper Outline: The remainder of the paper is organised as follows. Section II introduces key problems in the area of systems for industrial automation. Section III surveys related work in the area. Section IV gives a brief outline of the system architecture, focusing on aspects related to the achievement of temporal isolation. Section V describes the architecture explaining design choices and providing implementation details. Section VI describes the final demonstrator built for the RI-MACS project making use of the proposed architecture, in order to show its practical relevance in the context of an industry-driven scenario. Section VII presents quantitative results gathered through proper experiments on the proposed platform, focusing on the achieved enhancements in the timing behaviour predictability. Finally, Section VIII describes the current status of the development, along with the planned future directions of work.

II. Requirements in Factory Automation and Problem Presentation

The issues briefly introduced in the previous section have been extensively studied during the first phase of the RI-MACS project [1], where a set of precise requirements on the software and hardware infrastructures for industrial automation have been derived also considering interviews that have been carried out on a total of 35 experts[1], chosen from both large enterprises and Small and Medium Enterprises (SMEs) operating mainly in the areas of automotive, machinery and equipment, and industrial control. Results are summarised in Figure 1, showing the importance, in a scale from 1 to 5, of Plug & Play technology foreseen for year 2010 by interviewed experts, in the different control layers of a manufacturing plant: Enterprise Resource Planning (ERP), Manufacturing Execution Systems (MES), single production line or building (System), single station (Cell), Device, Sensor–Actuator. This study [2] led to the following points related to this paper:

- production lines life-cycle is expected to become more dynamic in the next years, where rapid reconfiguration and re-installation, achievable through Plug & Play devices capability, is considered to be a key success factor;
- the percentage of standardised and Plug & Play mechatronic solutions is still below 25%, but it is increasing;
- investments on automation systems (software, hardware and communication infrastructures) constitutes a significant part of the total investments for new plants;
- the adoption of open architectures and embedded control solutions is expected to put the foundations for an easy, dynamic reconfiguration of plants, while there is a general consensus in the will to substitute current proprietary communication solutions with open ones.

This set of high-level considerations may be translated into precise requirements that need to be satisfied in the automated factories of the (near) future:

- **Integration:** the HW/SW architecture of the plant control system must facilitate integration of different parts, to reduce costs incurred when assembling the system for commissioning and due to maintenance operations.
- **Heterogeneity:** given the wide range of commercial solutions and standards and the fragmentation of available technical solutions, it is not realistic to mandate specific HW/SW. The provided solution must integrate multi-vendor and multi-purpose software and hardware. Different subsystems may in principle use different hardware, programming languages and models, where legacy subsystems with proprietary protocols must be supported and cannot be ruled out from any realistic solution.
- **Interoperability:** despite heterogeneity of devices composing the global automation system, it should be possible to interconnect them through a standardised, clearly defined, possibly open set of interfaces.
- **Accessibility:** it should be possible for operators to have an easy and immediate access to the monitoring and reconfiguration interfaces of each interconnected device.

These requirements implicitly require that some more "intelligence" be put inside (or as close as possible to) interconnected components of the automated factory. Not only must they be capable of carrying on the main control operations they have been designed for, but they must also embed the software infrastructure needed for dealing with monitoring and reconfiguration capabilities. The additional layers of software needed to provide a uniform interface for accessing the multitude of heterogeneous devices introduces new problems, along with the many advantages they have been conceived for.

One important problem concerns the real-time and Quality of Service (QoS) aspects. For the system to operate properly, activities must be provided within pre-specified timeliness and/or QoS constraints. As an example, the operation of setting the value for a property of a component must be completed within a bounded time, otherwise the system may not work properly. Different types of service may have very different needs in terms of timeliness guarantees: for example, the discovery of a new component that has been just plugged into the network may take place in a large amount of time, as this is not considered to be a critical operation. On the other hand, notification of failures and error conditions must be delivered within very stringent time frames.

[1] It was decided to perform interviews personally to a relatively small number of experts, rather than sending out a questionnaire to many companies.

In the current industrial practise, timing constraints are guaranteed by using dedicated hardware and careful off-line benchmarking and analysis techniques. On the other hand, in the next-generation automation platforms, increased flexibility at lower costs is expected to be achieved by sharing, among different activities, the available computational and communication resources (thanks to the increasing trend in using Ethernet-based communications also in mission-critical areas).

However, the inter-mixing of activities with different criticality levels, in a shared set of nodes and communication links, makes it more difficult to provide the required QoS. Therefore, to respect in-place timeliness requirements, appropriate real-time scheduling strategies ensuring temporal isolation among tasks on the same physical node, as well as among concurrent communications on the same physical link, are needed.

For these reasons, it is very important to provide a flexible and robust infrastructure for supporting QoS at all levels in the system: flexibility is necessary because the system must be scalable and allow for dynamic configuration and reconfiguration of the QoS parameters; robustness, here, means that the proposed solution must be tolerant to faulty components, in the sense that if a software service starts to behave incorrectly, the guarantees of other services must not be affected.

In what follows, this paper describes the RI-MACS approach to address these issues: adopting a HW/SW architecture based on heterogeneous nodes connected through standard communication networks (like Ethernet) for soft-real-time communication, and custom real-time networks (e.g. CAN, Profibus, Interbus, etc.) for critical hard real-time traffic. On top of these, while adopting a Real-Time Operating Systems (RTOS) for hard real-time activities, it is also possible to use a General-Purpose Operating System (GPOS) like Linux, enriched with appropriate Real-Time extensions at the scheduling level, for the deployment of a middleware layer based on Service Oriented Architectures (SOAs) and Web Services. This constitutes the fundamental ground on which it is possible to build higher-level features like Plug & Play, dynamic reconfiguration, diagnostics, monitoring and logging, and Human Machine Interfaces (HMIs).

Indeed, such software technologies are open and interoperable. By making use of an infrastructure based on web services, the engineering process may exploit all the advantages of client-server based architectures, with publish-subscribe mechanisms supporting automated discovery of interconnected devices and of the set of services that are available on them, such as monitoring, control and re-configuration. Furthermore, any error condition that should arise at run-time may be detected and delivered to the operator through the network, so that appropriate recovery actions may be undertaken (either remotely exploiting the same networked infrastructure, or working directly on the device whenever necessary).

III. Related work

This section overviews related work in the general domains of the adoption of SOAs approaches, and the support for soft real-time and QoS guarantees through general-purpose infrastructures, in automation engineering.

The idea of adopting SOAs for manufacturing systems is not new. For example, in the context of the SIRENA European Project [3], a service-oriented communication framework is proposed in which an industrial plant is composed of intelligent devices. Such devices expose their own functionality as a set of services, hiding their complexity and allowing for transparent communication with other devices. This way, devices may be composed and aggregated into higher-level services, achieving a high grade of scalability. This approach is certainly fascinating, however it is not practical nor convenient today, because of the costs needed for the integration of the additional functionality inside the devices, and the problem of legacy sub-system integration. Moreover, real-time sensitive tasks cannot be handled satisfactorily using Service-Oriented Architectures, as none of the technologies used for the implementation of these architectures explicitly target real-time constraints. This is true even for the "Device Profile for Web Services" (DPWS [4]) standard, that is being adopted in the context of existing industrial plants, as documented in [5].

In [6], the need has been underlined for using SOAs not only in the well-established "high-level" domain of work-flow and information management, but also in the "low-level" one of plant monitoring, configuration and control. However, the same work pointed out that usually implementations of such infrastructures lack real-time capabilities, which are of fundamental importance due to the in-place timeliness constraints.

It is well-known from the real-time literature [7] that increasing the computation power on which software is running is not enough, in general, for meeting precise real-time requirements. Appropriate scheduling strategies and analysis techniques need to be put in action, and this is exactly what is done in the approach proposed in the present paper.

However, prior works exist that investigate on the integration of real-time scheduling strategies within middle-ware components for distributed real-time applications [8], [9], [10], [11]. For example, Hola QoS [12] is an architecture specifically tied to the needs of consumer electronics embedded multimedia systems, providing flexible resource management and adaptivity. CORBA-based approaches are also worth to mention. In fact, the CORBA specification has been extended to address reusability in the CORBA Component Model (CCM), which also considers QoS aspects. For example, this has been implemented in the Component-Integrated ACE ORB [13]. TAO [14] constitutes a C++ implementation of the Real-Time CORBA specification [15], which exposes fundamental functionality of distributed real-time applications via the CORBA paradigm. Also, TAO has been integrated with QuO [16], a framework that exploits the capabilities of CORBA to reduce the impact of QoS management on the application code. The result [17] is a middleware for adaptive QoS control using real-time scheduling facilities at the computation and network levels. However, the work presented in this paper is based on the SOA paradigm (not on CORBA), which is leveraged in order to achieve important properties such as automatic discovery and configuration, location-independence and fault-tolerance. Moreover, TAO used to rely on the traditional priority-based scheduling services foreseen by real-time CORBA, neglecting issues related to temporal enforcement (such as achieved by techniques like POSIX Sporadic Server [18]), while the present work relies on the more efficient EDF-based scheduling and temporal encapsulation provided by techniques existing in the domain of the real-time a-periodic servers [7]. Note that the Dynamic Scheduling extensions to real-time CORBA, also integrated within TAO [19], addressed the first

issue (adding deadline-based scheduling and adaptive changes of the scheduling parameters), but apparently not the second one (enforcement of temporal constraints).

Also, investigations on the adoption of real-time techniques in heterogeneous networks typical of automated factories have been carried on in the context of the Virtual Automation Network (VAN) project [20]. However, VAN focuses strongly on real-time and QoS support at the heterogeneous networking layer, whereas the architecture proposed in the present paper tackles the problem of real-time support both at the networking and at the computing/OS level. Similar comments apply to the work that can be found in [21], where the authors propose to extend the CAMX SOAP/XML-based communications framework with QoS support, where new XML messages are described for regulating the interactions among middleware components, whereas the actual QoS guarantees derive from the application of well-known Differentiated Services for IP networks to a set of aggregated data flows.

Finally, it is worth to mention the IRMOS European Project[2], started in February 2008, that is investigating on the use of SOAs and real-time technologies used at the networking, computing and storage levels. The project targets SOA-based high-performance computing services, to be provided through broadband Internet connections by service providers, under well-established Service-Level Agreements (SLAs) enriched with QoS specification, and in the context of well-defined business models with automated SLA negotiations.

To the best of the authors knowledge, this paper introduces for the first time an architecture that is comprehensive of the multi-faceted requirements typical of control units distributed in industrial plants: soft and hard (non-safety-critical) real-time computing and communications may share resources, and at the same time interact with safety-critical components that instead may coexist in the framework with their own dedicated (and usually proprietary) hardware elements; an SOA paradigm is used for the purpose of easing discovery of interconnected elements, control, configuration and monitoring of the plant, and web-service messages are extended for the purpose of supporting QoS-related attributes and their negotiation at the SOA level (see Section V).

Note that this paper mainly focuses on the intermixing of real-time techniques with SOAs, whilst other aspects typical of SOA-based approaches to software design, like semantics and ontology, are not considered. However, some works do exist that consider such aspects also in the application domain of industrial automation, for example the one in [22]. For aspects related to CPU scheduling, the work presented in this paper relies on the open-source AQuoSA [23] architecture for Linux. The presented framework has been built on top of AQuoSA, providing the SOA-level components needed to "plug" real-time scheduling in the wider context of a SOA platform for industrial automation. For further details on AQuoSA, the reader may refer to [23] and to the project web-site[3].

IV. System architecture

A. Real-time model of execution

Many real-time systems comprise activities with different levels of timing criticality. According to the classical definition [7], a *hard real-time* activity must always be completed before a certain *deadline*, otherwise some critical error may occur that invalidates the correctness of the entire system. An example of hard real-time activity is the low-level control loop of a robot arm. Another example is the identification of an hazard situation, the subsequent raising of an alarm and the execution of a procedure to put the system in a safe state.

Soft real-time activities have less critical requirements. They *should* complete before a certain deadline, however, if the deadline is missed, nothing catastrophic happens; rather, the *quality of service* delivered by the activity depends on the frequency of deadline misses. Examples of such activities are data streaming and logging and Human-Machine Interfaces (HMI). In an ideal world, it would be possible to always treat soft real-time activities like hard real-time ones; if no deadline is ever missed, then the QoS is certainly maximised. However, many practical issues affect the ability of a system to meet its timing requirements, like the unpredictability of the underlying hardware and operating system, the scarcity of resources, the sharing of physical resources between different activities, etc..

Finally, *non-real-time* activities do not present any real-time constraint and are performed in a *best effort* manner.

In a system in which all these types of activities coexist, the goal of the designer is: 1) to guarantee that hard real-time activities are always completed on-time; 2) to minimise the number of deadlines missed by soft real-time activities; and 3) to ensure that some residual bandwidth is available to non-real-time activities that are performed in background[4].

The timing criticality of an activity is not necessarily related to its time granularity. While a low-level control loop may need to be performed every few milliseconds, the deadline for a hazard identification may be in the order of hundreds of milliseconds. For soft real-time activities, a video stream that monitors an industrial process may be processed at various frequencies depending on the foreseen use of the video. What matters is not the timing granularity, but the possibility to a-priori *guarantee* that the activity will be completed on-time, or that it will be performed with a precise QoS.

B. Real-time requirements

The architecture envisioned in this paper aims to deal at the same time with devices and subsystems that are very different in nature and typical time-scale of operation. In the context of the RI-MACS project, the following real-time requirements have been identified:

- Enterprise Resource Planning (ERP) and Manufacturing Execution Systems (MES) have basically no QoS requirements and may be supported in a best-effort way.
- Communications at the entire "System" level (production line or building), which typically possess soft real-time requirements, and whose reaction times need to reside within hundreds of milliseconds (i.e., below $250ms$).
- Communications among devices localised at a single station (at the "Cell" level) where some work is done

[2] More information is available at the URL: http://www.irmosproject.eu.

[3] More information is available at the URL: http://aquosa.sourceforge.net.

[4] This may be done for example by considering an additional background scheduling entity for best-effort tasks when performing admission control for the system, like done by the Default Server of AQuoSA [24] or by the Best-Effort Bandwidth Server [25], and/or by a proper dynamic reclamation [26], [27] of the budgets unused by the real-time activities.

on the production line, which typically possess soft real-time constraints in the range of hundreds of ms.
- Interactions at the "Device" level, i.e., typical control loops within some mechatronic device (such as a soldering machine, or a robotic arm), which possess hard real-time requirements, with typical ranges below $10ms$.
- Interactions at the "Sensor&Actuator" level, i.e., typical of the *sense-compute-actuate* control loops needed for reacting to some environmental condition, which have also hard real-time requirements in a range below $10ms$.

While this classification may not be the most general one, this set of real-time requirements have been considered as a reference for designing the architecture presented in this paper.

The goal of the design is to let activities with different requirements and time granularity coexist on the same computing platform. To this end, the temporal behaviours of these activities need to be *isolated* as much as possible. In the proposed solution, this may be done by using dedicated hardware and software in some cases, like for the most critical activities and for legacy applications; in other cases, proper scheduling strategies may be used, like *resource reservations* (see Section V), in order to let different activities share the same physical resources without interfering with each other.

C. Architecture

Figure 2 depicts the automation platform that has been conceived in the RI-MACS project. In the envisioned architecture, standard (web-based) protocols and interfaces are combined with QoS support at the networking level, and real-time support at the computing level. The resulting facilities are exposed to application developers through two main Application Programming Interfaces (APIs). These have been designed to allow developers to build applications in a way that is as hardware-independent as possible.

The APIs can be divided into two categories: the Common API and the Custom API.

The Common API basically allows for the development of component-specific Web Services by exporting to the network the necessary sensor and actuator variables that are needed in order to monitor and operate on the associated device(s). This API is implemented through standard protocols and network stacks comprising SOAP, XML, HTTP and TCP/IP or UDP/IP. On the top of these, the Device Profile for Web Services (DPWS) standard is used in order to provide Plug & Play functionality of the devices within the architecture.

The Web Service communication stack is capable of providing response times in the order of tens of milliseconds [5] with a high variability in execution times, which is inappropriate for supporting hard real-time communications with stringent timeliness constraints, as needed within the automation plant. It is generally known that the performance bottleneck of current Web Service technology resides in the need for continuously parsing text-based XML chunks (from SOAP messages), and that such issue may be mitigated by the use of a binary-based encoding of XML hierarchies, as found for example in [28]. Recently, experiments [29] have shown that it is indeed possible to run complex web-services on top of real-time operating systems with bounded response times. However, powerful processors are still required making this approach not adequate to embedded systems with scarce resources.

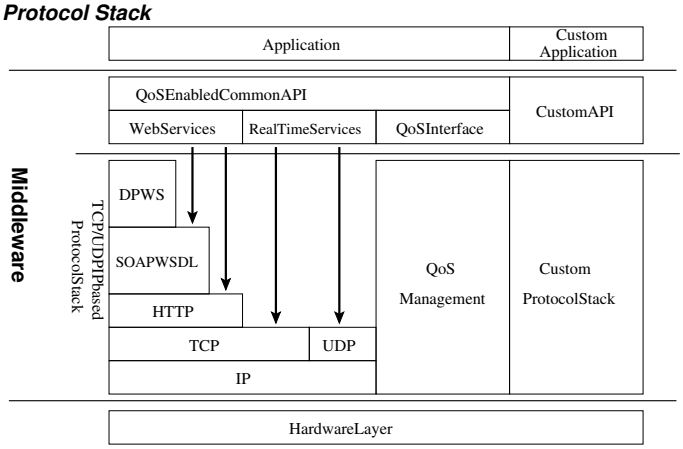


Figure 2. The RI-MACS Automation Platform

In the proposed architecture, the real-time service invocation channel has been separated from the Web Service communications stack. In other words, real-time operations are performed directly on the lower levels of the stack (e.g. UDP/IP on Ethernet), whereas less critical services (during discovery of new services or logging) are performed on top of the full web-service stack in a soft real-time fashion.

The Common API is rooted in standard OS and communication stacks so as to enhance portability, flexibility and composability. Proper scheduling techniques (see Section V) are used to isolate services from each other. This is achieved by associating each service with a fraction of the underlying resources, such that it appears to be executing on a slower dedicated virtual platform. This way, soft real-time support is enabled for commercial-off-the-shelf (COTS) components.

In the Common API, the RI-MACS platform also integrates the custom communication stack Modbus/TCP [30], so as to build the foundations for moving hard real-time automation services gradually towards the Common API paradigm, taking advantage of the benefits arising from such an approach. With the emerging standards for binary-encoded XML, and the ever increasing computation power of embedded micro-controllers, this is likely to constitute an innovative trend for development of next-generation hard real-time control units.

The **Custom API** provides hard real-time services, and it is instrumental for backward compatibility with legacy systems and for the provisioning of custom services. The Custom API is mostly executed on custom dedicated hardware. For example, it is possible to use a custom communication interface towards the CAN bus exclusively for legacy applications.

However, for the purpose of guaranteeing the correct operation of the RI-MACS Automation Platform, the management of the communication and computation resources is still done through the QoS management middleware. This is capable of allocating the available resources based on real-time requirements and load characteristics, and allows for negotiation of resources, monitoring of system operations, and reaction to transient overloads, as detailed in the next section.

D. Real-time Communications

Apart from custom dedicated networks, in the RI-MACS platform most of the traffic is directed through an IP stack on

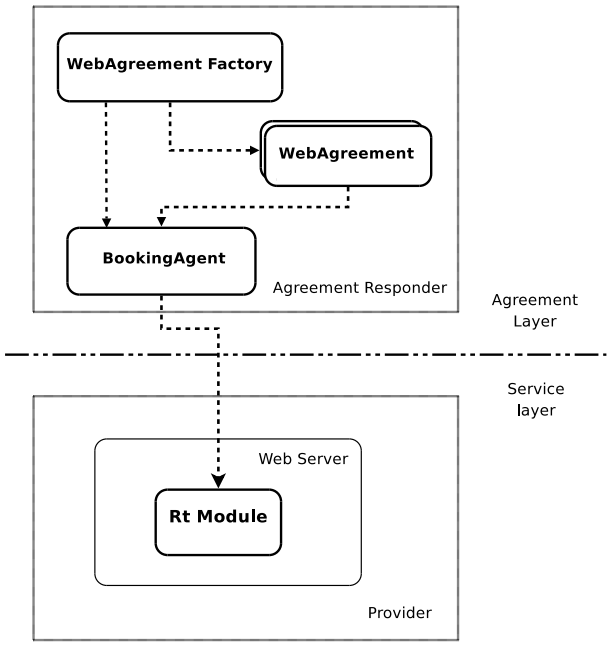


Figure 3. QoS Negotiation and Management Architecture

an Ethernet network. It is well-known that standard Ethernet, under high load that induces significant contention, cannot provide guaranteed response times. However, Ethernet and its variants are being used in industrial settings in place of more expensive and slower (but guaranteed) field buses.

Indeed, it is possible to provide at least soft real-time communication though IP on Ethernet. For the MAC layer, switched Ethernet is used within RI-MACS, which reduces the collision problems on the shared channels [31], [32]. It is then possible to use traffic smoothing techniques [33] to control the traffic load and allocate fractions of communication bandwidth to each application. For what concerns level 4 protocols, real-time communications rely on UDP, which is connectionless and does not support message re-transmission. Finally, non-real-time, or less critical real-time, communications, e.g., the DPWS service discovery protocol, rely on the TCP protocol.

V. QoS Architecture Description

This section focuses on the QoS part of the RI-MACS architecture, developed for the sake of satisfying timing requirements needed by soft real-time activities. In particular, a QoS negotiation and management architecture is proposed, which allows clients to negotiate QoS levels, with the guarantee of provisioning of the negotiated QoS. Regarding this basic functionality, the proposed architecture can be divided in two layers, as highlighted in Figure 3: the Agreement Layer, in which the negotiation happens; and the Service Layer, in which the service is provided with QoS support according to the negotiated parameters.

The QoS negotiation phase follows an agreement-based model, in which the two parties involved in the negotiation process establish a contract which specifies the QoS guarantees to be provided. The QoS architecture leverages the WS-Agreement framework [34], which uses open technologies (like Web Services and XML) to define: (a) a language for specifying QoS contracts; (b) a protocol to create contracts; (c)

```
<wsag:ServiceDescriptionTerm
         wsag:Name="server_parameters"
         wsag:ServiceName="use_of_web_server">
  <ret:ServerParams xmlns:ret="schemas.retis">
       <ret:CpuMinBudget unit="ms">9</ret:CpuMinBudget>
       <ret:CpuMaxBudget unit="ms">9</ret:CpuMaxBudget>
       <ret:CpuPeriod unit="ms">100</ret:CpuPeriod>
  </ret:ServerParams>
</wsag:ServiceDescriptionTerm>
```

Figure 4. XML fragment of an Agreement for negotiating CPU allocation parameters.

a protocol to verify the run-time compliance of contracts. WS-Agreement was chosen in this context not only for its flexibility in comparison with other QoS-enabled technologies (like WSLA [35]), but also for its standard nature (it is supported by the Open Grid Forum), which may ensure penetration of the platform within the industrial automation sector.

In the WS-Agreement specification view, a contract (or *Agreement*), is represented by an XML document mainly containing meta-information about involved parties and QoS parameters to be negotiated. In this work, the parameters to be negotiated are represented by the scheduling parameters that regulate the CPU allocation on the side of the web server accepting service requests.

CPU allocations have been managed through the well-known Reservations Based (RB) scheduling framework [36]. Such an approach provides the fundamental property of *temporal protection* (a.k.a., temporal isolation) in allocating a shared resource to a set of tasks that need to concurrently use it: this means that each task is reserved a fraction of the resource utilisation, so that its ability to meet timing constraints is not influenced by the presence of other tasks in the system.

The set of parameters transmitted in an Agreement reflect the RB allocation model. In RB scheduling, a resource allocation is specified in terms of a *budget* Q and a *period* P, with the meaning that the resource is granted for a minimum of Q time units every time-frame of duration P. The ratio Q/P represents the "share" of the resource that has been reserved, whereas the period constitutes the basic time granularity with which the share is granted (and is representative of the maximum activation delay). The actual budget that is granted to each reserved activity in a time window of duration P may usually vary between a minimum budget Q_{min} that is always guaranteed independently of other concurrently running activities, and a maximum budget Q_{max} that is never exceeded. The additional budget with respect to the basic guaranteed value Q_{min} may be distributed among competing reservations according to various policies, whose description is out of the scope of the present paper (the reader may refer to [23] for further details).

Referring to the structure of an Agreement (see [34]), a `Service Description Term` is used to store the QoS parameters of the server, defined using the XML Schema language [37]. A representative XML fragment, in which the CPU allocation to negotiate is $9ms$ every $100ms$, is shown in Figure 4.

Secondly, the QoS architecture uses the WS-Agreement framework for defining the interactions between involved parties, usually a service client and a service provider. An *Agreement Template* is used to generate an agreement offer,

filled with the requested scheduling parameters. This is then inspected by the service provider, which decides, according to its internal resource management policy, whether to accept or reject it. In this case, acceptance test is based on the admission control policy embedded within the underlying resource-reservation scheduler. If the agreement offer is accepted, then an Agreement is created and sent back to the requester, so that it knows it may access the service with the requested QoS level. On the other hand, if the agreement offer is rejected, the client is notified so that it may adopt some error management policy, such as trying again after decreasing the requested QoS level, or trying at a later time. Such situation may occur in case of temporary overload of the server that has already accepted a number of agreements saturating available resources.

This kind of interaction is realised by the agreement layer components, which are described as follows:

- **WebAgreementFactory** This component, which is an implementation of the common AgreementFactory component defined in the WS-Agreement framework, mainly interacts with the client in the agreement creation process. So it provides agreement templates, receives agreement offers and communicates to client decisions about them.
- **WebAgreement** This component, which is an implementation of the common Agreement component defined in the WS-Agreement framework, represents a created Agreement, so it is instantiated after each offer acceptance.
- **BookingAgent** This component performs admission control in order to verify if the QoS level requested by the client can be guaranteed, and, in such case, it reserves the necessary resources to correctly execute the requested service. When the reserved resources are no more necessary, the BookingAgent deletes them. The resources are reserved and deleted through the communication with the lower level of the architecture.

This partition of the agreement layer assures that an Agreement will be created only if QoS guarantees can be maintained during service provisioning. The relationships between components during the creation of an Agreement can be seen in the sequence diagram of Figure 5, related to a successful agreement creation. It can be seen that the client interacts with the WebAgreementFactory to retrieve a template and make an offer. Then, the WebAgreementFactory receives the offer and invokes the BookingAgent for the admission test. The BookingAgent evaluates if the requested QoS can be guaranteed and reserves resources for the client. After the positive response of the BookingAgent, the WebAgreement-Factory invokes the WebAgreement component to create the Agreement. Finally, as a sign of acceptance, a reference of the created Agreement is returned to the client. Note that all interactions will follow the WS-Agreement interaction model.

After the creation of an Agreement, service requests of the client must be served assuring the negotiated QoS. In case of the WS-Agreement interaction model, this is translated to the need for serving client requests, within a web server, with the pre-specified scheduling parameters.

This has been implemented, in the architecture, by the Rt-Module component, that uses the functionality of the Apache2 web server for receiving and processing service requests, then it reserves the actual resources by using the available API

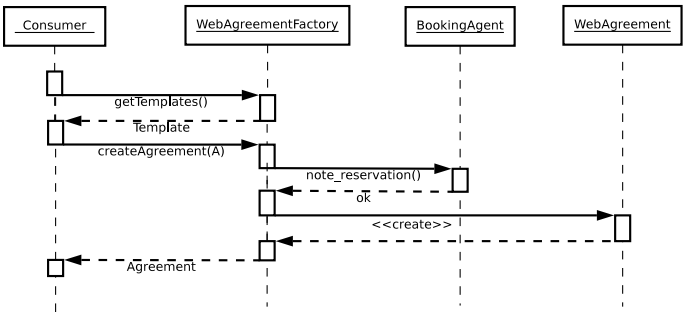


Figure 5. Successful Agreement creation

for accessing the RB facilities available in the underlying scheduler (see Figure 6).

Apache2 is a very popular web server and it is easily extensible thanks to its internal modular structure: this allowed for the realisation of RtModule as a web server module, making it more durable to server changes and easier to install.

In order to guarantee requested QoS in provisioning of services, the RtModule uses the user-space library made available through the AQuoSA framework [23], that enhances the Linux kernel with a real-time scheduling policy based on Earliest Deadline First (EDF), whose main concepts have been introduced in [38]. This way, the RtModule exploits real-time scheduling of the underlying modified OS kernel so as to provide temporal isolation to tasks that execute services on behalf of remote clients, resulting in guaranteed and predictable performance and response times of served requests. This approach perfectly suites the needs of soft real-time tasks in a Linux environment.

The RtModule has the following internal structure:

- The *WebServer Interface* uses web server functionality mainly to receive and process service requests.
- The *ReservationManager* uses the functionality of the underlying QoS support level to allow execution of services guaranteeing compliance with the negotiated QoS levels.

When a service request arrives to the web server, it is intercepted in order to determine if it has to be served with QoS guarantees. This is done by comparing the client identification with all the entries related to valid contracts. If a request must be served guaranteeing QoS, then the ReservationManager is invoked to create a reservation to manage client requests, if it has not been created yet. However, in case of multiple requests coming concurrently from the same client, only a single reservation is created, to which all service tasks are

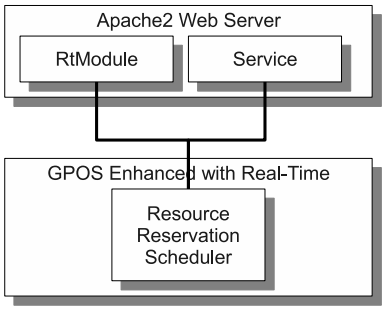


Figure 6. The RtModule in the RI-MACS QoS architecture for CPU reservations.

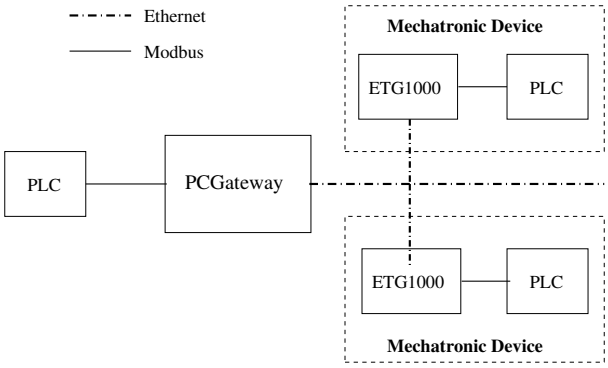


Figure 7. Final demonstrator architecture used in the RI-MACS project.

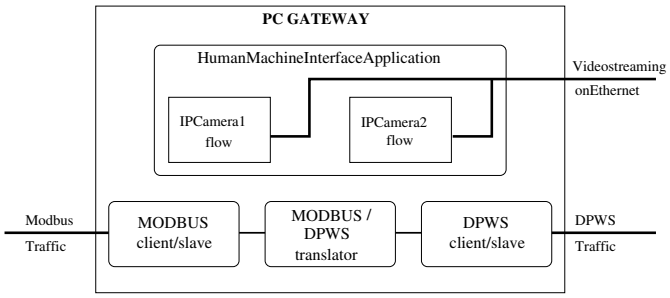


Figure 8. Software organisation within the Linux gateway.

attached. This way, all service requests coming from the same client are encapsulated in the same CPU reservation, guaranteeing temporal isolation across reserved services even in case of malfunctioning or misbehaviour of one or more clients (or services).

VI. RI-MACS Final Demonstrator

This section presents the demonstrator that has been set-up, in front of the EC reviewers during the last official project review meeting, in order to show effectiveness of the described architecture for QoS management in the domain of soft real-time industrial applications. The purpose of this section is to highlight qualitatively the advantages of the proposed architecture by means of an industry-driven application, whilst a quantitative evaluation is performed in Section VII.

In order to show the RI-MACS platform capabilities, the architecture shown in Figure 7 was conceived and set up.

In the picture, the following actors may be identified:

- a controlling PLC (the one on the left), with the purpose of running the controlling program within the global architecture;
- a PC, with the role of gateway between Modbus communications from the PLC and the DPWS-based ones throughout the rest of the network (over the Ethernet bus);
- one ETG-1000 server for each mechatronic device, which parses the DPWS messages and forwards them to the mechatronic devices on Modbus connections;
- several mechatronic devices, comprised of a PLC and a standard industrial tool (like clamps or conveyors). Each PLC transforms a passive tool in an active one, with the possibility to interact with the rest of the plant.

The Linux-based PC gateway has been equipped with the RI-MACS QoS management infrastructure described in the previous section, so that the underlying AQuoSA scheduler could be leveraged in order to guarantee the appropriate temporal isolation degree across concurrently running soft real-time tasks.

The gateway software consists of three components:

1) a background network-based application
2) two video streaming viewers;
3) a coordinating Human-Machine Interface application.

A visual representation is shown in Figure 8. The most important software component is the first one: its role is to take Modbus messages from the controller PLC and translate them into DPWS messages to be sent to the recipient of the original Modbus frame (bottom of Figure 8). Since the PLC is not able to speak an IP addressing compliant language, the software component takes care of translating the logic addresses of mechatronic devices into IP ones and replaces the source address with the IP address of the gateway itself. When the message reaches its destination, the corresponding DPWS answer is generated and reaches the gateway once again: it is then time to translate it back to the Modbus language, update all the source and destination fields, and send it to the PLC.

The application gives the user the possibility to start and stop the translating activity and to protect it by means of Resource Reservations, through the usual budget and period parameters specification. Furthermore, the experimental set-up comprises the creation of a reservation for two instances of the `mplayer` cross-platform multimedia player[5], whose streams come from two IP video cameras: their purpose is to give a video feedback to the user of what is going on inside the plant.

In the demo, the usual comparison of the plant (and video streaming) behaviour obtained with and without QoS management through the RI-MACS architecture, showed qualitatively the advantages of the proposed approach in provisioning of real-time performance guarantees to individual activities in the context of industrial automation.

Furthermore, note that the envisioned architecture allows also for the containment of the possible problems caused by misbehaving components in the plant, due to undesired temporal interference. In fact, should a software component exhibit a failure leading to a wasteful consume of resources (i.e., a bug leading to infinite CPU-intensive loops), the interference remains contained within the bounds that have been assigned at system design time for that software component, without disrupting the functionality of the rest of the system.

Finally, note that, whenever mechanisms of this type are engaged for all the resources involved in the networked plant, they may be regarded as a robustness/security feature: in case an attacker voluntarily manages to cause the misbehaviour of a software component (consuming excessive CPU or bandwidth) for the purpose of building a Denial of Service attack to the plant, its efforts are likely to be contained within the boundaries of the temporal allocations that were in place for that computing or communication element.

VII. Experimental evaluations

The experimental evaluations described in this section focus on the verification of the behaviour of the proposed architec-

[5] More information is available at the URL: http://www.mplayerhq.hu/.

ture, especially in guaranteeing a certain QoS level during service provisioning. In particular, it is shown that it is not possible to ensure predictable QoS levels, especially in heavy load conditions, without using appropriate real-time scheduling techniques. In the following experiments, scenarios that are well-suited to the industrial automation field have been set-up. The first two scenarios are built so as to "mimic" typical image-processing services that may be needed in complex vision-based control logics.

A. First scenario: centroid detection

The first scenario regards the object tracking problem and, in particular, centroid detection. A network camera was used as a device, capable of continuously acquiring images in jpg format with resolution of 640x480 pixels. A gateway PC was directly connected to the camera, exposing to clients a WS-service providing centroid position detection within the acquired image. The service, provided through a CGI interface, consisted of: image acquisition from the camera; image decompression; binarisation and centroid computation. These details were obviously hidden to clients, which only received the centroid coordinates in the acquired image. Then, two clients have been deployed that simultaneously requested the service, 50 times each. The service requests were generated by using the Apache Benchmark tool[6].

The network connecting the two clients with the server was a switched Ethernet LAN, and traffic smoothing techniques have been used as described in [33] to avoid interference between different traffic flows. In particular, each client was reserved a fixed fraction of the network bandwidth.

Note that the service needs to be provided respecting timing guarantees even if the PC gateway, which provides services through an Apache2 web server, is in heavy-load conditions: to simulate this aspect, all the experiments were made when the server executed in background a time-consuming task.

As the PC gateway is stressed by the Apache2 web server executing requests, its behaviour has been verified both using an unmodified Apache2 web server and an Apache2 enhanced with the RtModule. In particular, a reservation of $45ms$ every $100ms$ has been assigned to each incoming request, in order to exploit almost all the CPU computation power for service provisioning (remember that clients generated two concurrent requests each time). For each test case, 20 runs of the experiment have been repeated.

Among all the results collected by the benchmarking tool, the service response times have been collected, and in particular the minimum, average and maximum values, the standard deviation and 90% confidence intervals.

Results obtained for the unmodified web server are reported in the first column of Table I (90% confidence interval is 7.5%), whereas results obtained for the web server containing the RtModule are reported in the second column of the same table (90% confidence interval is 1.1%).

In order to allow the client application to track the centroid position with a sufficient precision, this service needed a soft real-time constraint of a response-time below $300ms$.

The maximum values reported in the first column of Table I show that the original unmodified web server is not capable of satisfying this timeliness constraint. On the other hand, the maximum response times exhibited by the web server enhanced with RtModule successfully managed to always respect the design constraint: this behaviour is due to the CPU scheduling mechanism leveraged within the modified Apache server architecture, that allows for guaranteeing temporal isolation among client requests that need CPU-intensive services.

[6] More information is available at the URL: http://httpd.apache.org/docs/2.2/programs/ab.html.

Table I
SERVICE RESPONSE TIMES

Processing times	Original web server (ms)	RtModule server (ms)
min	119	115
avg	198	173
max	1175	273
std.dev	117	23

B. Second scenario: image rotation

The second scenario regards the problem of object flaw auto-detection, which can involve geometric transformations on images, like reported in [39]. In particular, a simple image rotation algorithm has been chosen for the experiment.

Also in this case the server behaviour has been verified both using an unmodified Apache2 web server and an Apache2 enhanced with the RtModule. For each test case, 20 repetitions of the experiment have been done, with a heavy-loaded server. In this case, requests were made by 10 clients simultaneously: each client made 10 requests, for a total of 100 requests per simulation. The Apache2 web server was configured to serve 10 requests concurrently with 10 different tasks and each task was assigned by the RtModule a CPU reservation with a share of 9%, and a period of $P = 100ms$.

The service response times have been measured for a large image of 2000x2000 pixels, in order to highlight how the best-effort model cannot provide sufficient performance guarantees even when the computation times required for service execution are large. This fact can be appreciated by a graphical comparison between the different behaviours of the two configurations, as depicted in Figure 9.

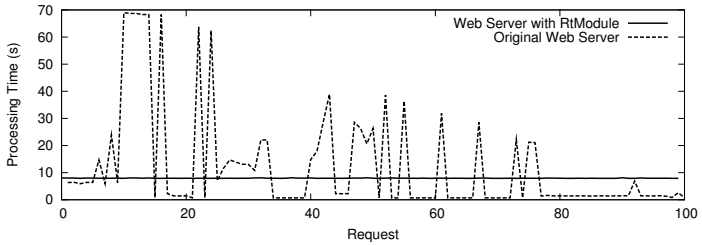


Figure 9. Response times obtained with and without RtModule.

The graphs report the request number on the x-axis, and the corresponding processing time (in seconds) on the y-axis. Their comparison shows that the response times obtained with real-time scheduling are far more predictable than the ones obtained without the RtModule, which exhibit an unpredictable behaviour. This can be explained by the fact that the resource reservation techniques implemented in the RtModule

provide a dedicated slower virtual processor for each service instance. Therefore, with the RtModule, each service instance has an almost constant response time (the continuous line in Figure 9), because it has been *reserved* a fraction of the real processor. On the contrary, without the RtModule, due to the lack of temporal isolation, the service response time can exhibit significant fluctuations (the dashed line in Figure 9), if the processor is subject to concurrent requests.

VIII. Conclusions and future work

This paper addressed some of the problems that arise in deploying a middleware layer for supporting Service Oriented Architectures in next generation industrial automation platforms. In particular, real-time and QoS aspects have been addressed, giving an effective way to guarantee QoS in service provisioning through SOA. The architecture of the proposed framework has been described, and its effectiveness has been shown by means of extensive experimental evaluations, both quantitative and qualitative, highlighting that the framework provides significant and effective advantages over existing solutions.

One possible direction of future work in this area is the integration, within the framework, of adaptive reservation techniques and feedback-based QoS control strategies [40], for the purpose of assessing their effectiveness for efficient resource management in the industrial automation domain.

IX. Acknowledgements

This work has been partially supported by the European Project "Radically Innovative Mechatronics and Advanced Control Systems" (RI-MACS), Project Number NMP2-CT-2005-016938, Thematic Priority Nanotechnologies and nanosciences, knowledge-based multifunctional materials and new production processes and devices (NMP).

It has been partially supported also by European Project "Interactive Real-time Multimedia Applications on Service Oriented Infrastructure" (IRMOS/ICT/2008/214777).

References

[1] "Radically Innovative Mechatronics and Advanced Control Systems (RI-MACS), Project Web Site," http://www.rimacs.eu.

[2] "Radically Innovative Mechatronics and Advanced Control Systems (RI-MACS) – Deliverable D1.2 – Report on Industrial Requirements Analysis for the Next Generation Automation Systems," http://www.rimacs.eu.

[3] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62–70, Feb. 2005.

[4] "Microsoft Devices Profile for Web Services specifications," http://msdn2.microsoft.com/en-us/library/ms951214.aspx, February 2006.

[5] F. Jammes, A. Mensch, and H. Smit, "Service-oriented device communications using the devices profile for web services," in *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, MPAC05*, ser. ACM Internation Conference Proceeding Series. Poznan, Poland: ACM Press, November 2005, pp. 1–8.

[6] N. Komoda, "Service oriented architecture (soa) in industrial systems," *Industrial Informatics, 2006 IEEE International Conference on*, pp. 1–5, Aug. 2006.

[7] G. C. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Santa Clara, CA, USA: Springer-Verlag TELOS, 2004.

[8] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic, "Controlware: A middleware architecture for feedback control of software performance," in *Proc. of International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.

[9] E. Eide, T. Stack, J. Regehr, and J. Lepreau, "Dynamic cpu management for real-time, middleware-based systems," in *Proc. of 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, May 2004.

[10] C. D. Gill, J. M. Gossett, D. Corman, J. P. Loyall, R. E. Schantz, M. Atighetchi, and D. C. Schmidt, "Integrated adaptive QoS management in middleware: A case study," *Real-Time Systems*, vol. 29, no. 2-3, pp. 101–130, march 2005.

[11] N. Shankaran, X. D. Koutsoukos, D. C. Schmidt, Y. Xue, and C. Lu, "Hierarchical control of multiple resources in distributed real-time and embedded systems," in *ECRTS'06: Proceedings of the 18th Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 151–160.

[12] M. García-Valls, A. Alonso, J. Ruiz, and A. M. Groba, "An architecture of a quality of service resource manager middleware for flexible embedded multimedia systems." in *SEM*, ser. Lecture Notes in Computer Science, A. Coen-Porisini and A. van der Hoek, Eds., vol. 2596. Springer, 2002, pp. 36–55. [Online]. Available: http://dblp.uni-trier.de/db/conf/edo/sem2002.html#VallsARG02

[13] N. Wang, D. C. Schmidt, A. Gokhale, C. Rodrigues, B. Natarajan, J. P. Loyall, R. E. Schantz, and C. D. Gill, *QoS-enabled Middleware*, qusay mahmoud, ed. ed. New York, Wiley and Sons, 2003, vol. Middleware for Communications.

[14] D. C. Schmidt, D. L. Levine, and S. Mungee, "The design of the tao real-time object request broker," *Computer Communications*, vol. 21, pp. 294–324, 1997.

[15] V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zykh, and R. Johnston, "Real-time corba," in *IEEE Real Time Technology and Applications Symposium*. IEEE Computer Society, 1997, pp. 148–.

[16] Y. Krishnamurthy, V. Kachroo, D. A. Karr, C. Rodrigues, J. P. Loyall, R. E. Schantz, and D. C. Schmidt, "Integration of QoS-enabled distributed object computing middleware for developing next-generation distributed application," in *LCTES/OM*, 2001, pp. 230–237. [Online]. Available: citeseer.ist.psu.edu/krishnamurthy01integration.html

[17] R. E. Schantz, J. P. Loyall, C. Rodrigues, D. C. Schmidt, Y. Krishnamurthy, and I. Pyarali, "Flexible and adaptive qos control for distributed real-time and embedded middleware," in *Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 374–393.

[18] IEEE, *Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) Amendment: Additional Realtime Extensions.*, 2004.

[19] Y. Krishnamurthy, I. Pyarali, C. D. Gill, L. Mgeta, Y. Zhang, S. Torri, and D. C. Schmidt, "The design and implementation of real-time corba 2.0: Dynamic scheduling in tao," in *IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society, 2004, pp. 121–129.

[20] P. Neumann, A. Poeschmann, and R. Messerschmidt, "Architectural concept of virtual automation networks," in 17^{th} *IFAC World Congress*, Seoul, Korea, July 2008.

[21] I. M. Delamer and J. L. Martinez Lastra, "Quality of service for CAMX middleware," *International Journal of Computer Integrated Manufacturing*, vol. 19, no. 8, pp. 784–804, December 2006.

[22] J. Lastra and M. Delamer, "Semantic web services in factory automation: fundamental insights and research roadmap," *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 1, pp. 1–11, Feb. 2006.

[23] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari, "AQuoSA — Adaptive Quality of Service Architecture," *Software – Practice and Experience*, vol. 39, no. 1, pp. 1–31, 2009.

[24] T. Cucinotta, "Access control for adaptive reservations on multi-user systems," in *in Proceedings of the* 14^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008)*. St. Louis, MO, United States: IEEE, April 2008.

[25] S. Banachowski, T. Bisson, and S. A. Brandt, "Integrating best-effort scheduling into a real-time system," *Real-Time Systems Symposium, IEEE International*, vol. 0, pp. 139–150, 2004.

[26] M. Caccamo, G. C. Buttazzo, and D. C. Thomas, "Efficient reclaiming in reservation-based real-time systems with variable execution times," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 198–213, Feb. 2005.

[27] L. Palopoli, L. Abeni, T. Cucinotta, G. Lipari, and S. K. Baruah, "Weighted feedback reclaiming for multimedia applications," in *Proceedings of the* 6^{th} *IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia 2008)*, Atlanta, Georgia, United States, October 2008, pp. 121–126.

[28] O. Goldman and D. Lenkov, *XML Binary Characterization*, W3C Working Group, March 2005.

[29] G. Moritz, S. Prüter, D. Timmermann, and F. Golatowski, "Real-time service-oriented communication protocols on resource constrained devices," in *Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT)*, Oct 2008, pp. 695–701.

[30] A. Swales, *Open Modbus/TCP Specification*, Schneider Electric, March 1999.

[31] J. Loeser and H. Haertig, "Low-latency hard real-time communication over switched ethernet," in *Proceedings. 16th Euromicro Conference on Real-Time Systems (ECRTS)*, 2004, pp. 13–22.

[32] H. Hoang, "Real-time communication for industrial embedded systems using switched ethernet," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, Apr 2004.

[33] L. Lo Bello, L. Kacinsky, and O. Mirabella, "Improving the real-time behavior of ethernet networks using traffic smoothing," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 3, pp. 151–161, Aug 2005.

[34] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web Service Agreement specification (WS-Agreement)," March 2007.

[35] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, "Web Service Level Agreement (WSLA) language specification," http://www.research.ibm.com/wsla, 2003.

[36] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves for multimedia operating systems," Carnegie Mellon University, Pittsburgh, Tech. Rep. CMU-CS-93-157, 1993.

[37] P. Walmsley and D. C. Fallside, "XML schema part 0: Primer second edition," W3C," W3C Recommendation, Oct. 2004, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[38] L. Abeni and G. Lipari, "Implementing Resource Reservations in Linux," in *Real-Time Linux Workshop*, 2002.

[39] C.-L. Su, "Robotic intelligence for industrial automation: Object flaw auto detection and pattern recognition by object location searching, object alignment, and geometry comparison," *J. Intell. Robotics Syst.*, vol. 33, no. 4, pp. 437–451, 2002.

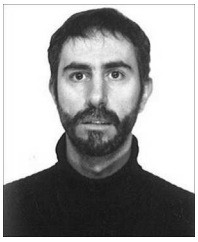
[40] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli, "QoS management through adaptive reservations," *Real-Time Systems Journal*, vol. 29, no. 2-3, March 2005.

Tommaso Cucinotta graduated in Computer Engineering at the University of Pisa in 2000, and received the PhD degree in Computer Engineering from the Scuola Superiore Sant'Anna of Pisa in 2004. He is Assistant Professor of Computer Engineering at the Real-Time Systems Laboratory (ReTiS) of Scuola Superiore Sant'Anna. His main research activities are in the areas of real-time and embedded systems, with a particular focus on real-time support for general-purpose Operating Systems, and security, with a particular focus on smart-card based authentication.

Antonio Mancina took his Ph.D. at the ReTiS Lab of Scuola Superiore Sant'Anna in April, 2009, with a work entitled "Operating Systems and Resource Reservations". His main research interests comprise real-time scheduling algorithms and micro-kernel operating systems.

Gaetano Anastasi was born in Marsala, Italy, in 1982. He is a PhD student at the ReTiS Lab of the Scuola Superiore Sant'Anna (Pisa, Italy). He received the Laurea degree in Computer Engineering from the University of Pisa in February 2008. Currently, his main research interests include real-time scheduling, QoS management issues in web applications, QoS negotiation methods and real-time virtualisation.

Giuseppe Lipari graduated in Computer Engineering at the University of Pisa in 1996, and received the PhD degree in Computer Engineering from Scuola Superiore Sant'Anna in 2000. He is Associate Professor of Operating Systems with Scuola Superiore Sant'Anna. His main research activities are in real-time scheduling theory and its application to real-time operating systems, soft real-time systems for multimedia applications and component-based real-time systems. He has been member of the program committes of many conferences in the field. He is currently Associate Editor of IEEE Transactions on Computers.



Leonardo Mangeruca is Senior Research Scientist at PARADES S.c.a.r.l., Rome, Italy. His research interests include design methodologies, safety-critical hardware/software architectures and formal methods for embedded systems design. He received the PhD degree in Electrical Engineering from the University of Genova, Italy in 1995.

Roberto Checcozzo received Degree in Informatics Engineering from the Politecnico di Torino, Italy, in 2003. Since 1993 he works in the field of software development for automotive production plants. In 1995, he was employed by Comau Body Welding and Assembly, where he leaded software control projects for Fiat, Ford, Daimler, BMW and Audi in Brazil, Spain, Germany and Italy. From 2006 he has been participating in the management of research projects in the Automation Control Engineering field.

Fulvio Rusinà began his career with CSELT, the Telecom Italia Labs. In 1986, he joined SESAM, a joint venture of COMAU and DEC (Digital Equipment Corporation), as Unit Manager, and he continued his assignment as Managing Consultant in the area of factory automation. Since 1993, Fulvio has served COMAU in a variety of management roles, including Research and Development Planning as well as European Business Development and Marketing of the Service Business. Since 2004, Fulvio has been responsible for Advanced Engineering at the COMAU Headquarters.