Quality of Service Management in Service Oriented Architectures

Gaetano Francesco Anastasi

ReTiS Lab.

SCUOLA SUPERIORE SANT'ANNA

A thesis submitted for the degree of

Doctor of Philosophy

Tutor: Prof. Giuseppe Lipari

18^{th} of November, 2011

Acknowledgments

This thesis is the result of three years of research mainly conducted at the ReTis Lab. of the Scuola Superiore Sant'Anna of Pisa and I would like to thank all the people of the Lab with which I shared this professional and human experience. I avoid to name you all just to not forget anybody. However, particular thanks go to my tutor, Prof. Giuseppe Lipari, for helping and supporting me.

I also thank Prof. Marisol García-Valls for permitting me to spent a research period at the University Carlos III of Madrid. I would like to thank all the people I met there (the members of the Drequiem Lab. and the guys of the GAST group) for their hospitality. In particular, I thank Prof. Pablo Basanta-Val for offering me several *cañas* during my stay.

Finally, predictable acknowledgments go to my family and, last but not least, to my friends, for their friendship during all these university years.

Abstract

The concept of Service Oriented Architectures (SOAs) has gained momentum in Information and Communication Technology (ICT) application area in recent years, introducing an innovative approach to the analysis, design and development. Many applications provided as services are time-critical and demand soft real-time requirements that must be taken into account for providing a certain Quality of Service (QoS) to service consumers. Moreover, in the context of SOAs there is an increasing interest in enriching Service Level Agreements (SLAs) established between providers and consumers with attributes as quantifiable QoS parameters, that the provider must respect for avoiding to incur in penalties.

For providing strong guarantees, SOAs must be enhanced with an advanced execution management that takes into consideration the underlying resources used for service provisioning. Management in the SOA context is not only about managing the services, but also about managing the network, the computing units and various other resources, that could be also virtualized in the case of cost-effectively large-scale systems.

In this dissertation, this problem will be generically indicated with the term *QoS management* and will be addressed with a particular focus on service-oriented real-time applications. By using proper resource management techniques borrowed from the real-time system theory, it will be shown that the service provider can guarantee the QoS negotiated by consumers, in the context of QoS-enabled SOAs. In particular, a generic service-oriented QoS architecture has been designed and developed for negotiating and providing services with soft real-time guarantees. Also, many realistic experiments have been conducted in Linux testbeds for showing the effectiveness of the proposed approach in different ICT environments, like industrial automation platforms, virtualized infrastructures and Wireless Sensors Networks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The concept of Service Oriented Architectures (SOAs) has certainly gained momentum in Information and Communication Technology (ICT) application area in recent years [79, 33, 15]. The market for SOA can certainly be representative of such great interest, as markets at $450 million in 2005 are expected to reach $18.4 billion by 2012 [84].

SOA is a design methodology that relies on services, that constitute the smallest bricks of software necessary to build distributed applications: services are published, discovered and invoked over a net whilst applications are simply built by putting services together. This architectural model can be simplified by the well-known *publish/find/bind* pattern, that involves three main parts playing different roles: a service provider, publishing and providing services; a registry, containing list of services; a service consumer, seeking for services and requesting them. Services must obey to certain well-known interrelated principles [48, 32] that drive their design. In particular, a service must be highly-abstracted and loosely-coupled to favor interoperability, it must be flexible and reusable for adapting to varying requirements and scenarios, and finally it must be autonomous and composable, for realizing distributed computation and permitting to easily create added value from existing parts. As many of the techniques used for SOA components (e.g. databases, transactions, software design) are already well-established in isolation, it should be stressed that the main innovation of SOA relies on the architecture, that is capable of putting into cooperation autonomous and heterogeneous components for building systems that can easily scale.

Fitting in with this context, the motivation inspiring this work will be analyzed. Then, the given contribution, with respect to the current state of art, will be detailed. Finally, the organization of this dissertation will be described.

1.1 Motivation

The SOA revolution consists in the introduction of an innovative approach to the analysis, design and development in all types of ICT environments, from the in-

dustrial automation [49] to the interactive World Wide Web [89], from pervasive environments [43] to the large-scale "Clouds" [103]. A lot of research effort has been put for applying the service-oriented paradigm in these fields, in order to gain all the benefits provided by the SOA adoption. Focusing on the European Union, many research projects have arisen with such intent in the last years: among them we can cite SIRENA [1] (Services Infrastructure for Realtime Embedded Networked Applications), intended to create a service-oriented framework for specifying and developing distributed applications in industrial automation and automotive electronics; SOA4All [2], providing a framework and infrastructure that combines SOA with Web 2.0 technologies, including the semantic web and context management technologies; IRMOS [3] (Interactive Realtime Multimedia Applications on Service Oriented Infrastructures), aimed at design, develop and validate Cloud Solutions which will allow the adoption of interactive real-time applications, and especially multimedia applications.

Most of these projects explicitly address the real-time requirements that many time-critical application demand, like banking and finance or industrial automation applications. Systems characterized by time requirements (being them explicitly or inherently defined) are real-time systems and, according to a widely-accepted consensus [17], can be distinguished in *hard real-time* systems (e.g. mission-critical systems), in which violating a deadline will cause a catastrophic event and *soft real-time* systems (e.g. multimedia systems), in which violating a deadline will only cause a performance degradation. The applications considered in this dissertation belong to the soft real-time realm, where the timing requirements are often tied with the concept of Quality of Service (QoS). The QoS term assumes different meanings in different contexts but it is always related to the satisfaction of users consuming a service. If a (soft) real-time service misses its deadline very often this will cause significant performance degradations and thus user in-satisfaction and a low QoS. This is very undesirable, especially in the context of SOAs, where there is an increasing interest in enriching Service Level Agreements (SLAs) established between providers and consumers with attributes as quantifiable QoS parameters or penalties for the provider if such parameters are not respected.

For providing strong guarantees, SOAs must be enhanced with an advanced execution management that takes into consideration the underlying resources used for service provisioning. Management in the SOA context is not only about managing the services, but also about managing the network, the computing units and various other resources, that could be also virtualized in the case of cost-effectively large-scale systems. In this dissertation, this problem will be generically indicated with the term *QoS management* and will be addressed with a particular focus on service-oriented real-time applications. By using proper resource management techniques borrowed from the real-time system theory [16], it will be shown that the service

[1] More information is available at the URL `http://www.sirena-itea.org/`

[2] More information is available at the URL `http://www.soa4all.eu`

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

provider can guarantee the QoS negotiated by consumers, in the context of a QoS-enabled SOA.

1.2 Related Work

The motivation of this work started from the fact that the SOA design paradigm is being applied to many application contexts. Thus, this work investigated the QoS management problem for SOAs in three main contexts: the industrial automation environments, the Clouds and the pervasive environments. As these contexts have many differences, the organization of this dissertation reflects such division and each chapter has its own related work section, where differences related to each contexts are carefully addressed. Instead, this section aims to identify the common issues faced in this work, situating it in a broader picture.

The idea of managing and controlling QoS through the proper use of soft real-time techniques is not new and many architectures have arisen for addressing such issues. As an example, Hola-QoS [41] is an architecture of a QoS resource manager tied to the needs of Consumer Electronics Embedded Multimedia Systems (CEEMSs). Through a set of homogenous architecture layers, Hola-QoS gives support to QoS management at different abstraction levels, building an integral QoS management for CEEMSs with hierarchical control across layers. Higher-level management is performed less frequently than lower level; however, higher-level management operations and decisions have more influence on system operation than lower level ones. At the lower level, the QoS management is performed by assigning a budget to application tasks for executing on a system resource, where the budget is an amount of resource that is granted for use. Also, the architecture embeds a component [7] for collecting statistical information on applications resource usage and system resource availability, to let the resource manager improve system behavior.

Despite the different application context of Hola-QoS with respect to SOAs, the approach of assigning resource budget to application tasks and performing statistical monitoring resembles the one taken in this work. However one important difference exist in the fact that Hola-QoS relies on the assumption application experts know well the structure of media processing functions, and can model their resource needs in the average case. As such assumption is not true in the context of SOAs, where services can be of different nature, this work present a new QoS data management framework that is capable of predicting the initial resource budget assignment. Moreover, Hola-QoS relies on external Real-Time Operating Systems (RTOSs) for resource accounting and enforcement facilities, whilst this work relies on General Purpose Operating Systems (GPOSs) - enhanced with real-time facilities - that have more issues concerning resource management, due to the presence of many hardware and software features that limit predictability (cache, interrupts, etc.).

In the last years, one of the most important innovation for achieving predictability in GPOSs has been the introduction of the Resource Reservation (RR) [72] framework. 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 utilization, so that its ability to meet timing constraints is not influenced by the presence of other tasks in the system. Specifically considering CPU reservations, different algorithmic solutions [83] [69] [60] [1] have been implemented on a variety of systems. However, the traditional way for using Resource Based (RB) scheduling (reserving a fixed fraction of the CPU bandwidth to each task), is not very suitable for QoS management, as a static allocation of resources can only be based on the average requirements, leading to imprecision in respecting the timing constraints and inefficiency in terms of CPU utilization. This problem can be addressed by dynamically adapting the amount of resources reserved to each task throughout its execution, by using a feedback inside the scheduling mechanism.

For example, such approach is followed by Abeni, Cucinotta et al. [2] [22], that tackle requirements of multimedia tasks and present a modular architecture including three control algorithms for providing the so-called adaptive reservations. The point they advocate is that, in presence of large fluctuations on the computation requirements of the tasks, a feedback control must be applied for dynamically adapt the fraction of the CPU allocated to a task based on QoS measurements. The architecture they present has been later revised, extended and implemented into the AQuoSA (Adaptive Quality of Service Architecture) framework [77], a layered architecture that runs on top of the Linux kernel, providing also user-space APIs for accessing the RR scheduling facilities implemented in kernel-space. The feedback algorithms embedded in the AQuoSA architecture focus on resource-level adaptation, in which the resource shares granted to the applications are adapted to the dynamic workload requirements, and do not deal with application-level adaptation, where applications may switch among various modes of operations. In a work by the same authors [21], an integrated approach for QoS control is presented, using application-level, resource-level and power-level adaptation techniques in conjunction.

With respect to such approach, this work do not apply the traditional feedback-based scheduling that dynamically adapts the scheduling parameters for an application continuously running. In the SOA context, in fact, consumer requests are served by service instantiation that can occur at different distances in time and the QoS is managed by instantiating the service each time along with the creation of the associated resource reservation. Thus, this work takes into consideration changing requirements of the service by closing the "feedback control loop" in an off-line fashion, recurring to the persistent storage.

However, this work leverages the AQuoSA architecture at the lower level, for managing resource allocation in the Linux kernel through the RR APIs. Also the work of Sojka et al. [93] exploits AQuoSA at the low-level scheduler for building

a middleware architecture that allows soft real-time applications to reserve heterogeneous resources with real-time scheduling capabilities in a distributed environment. In their architecture, applications negotiate a service contract specifying the amount of resources that are needed for achieving the desired real-time performance. Contracts are negotiated with the framework, which is in charge of performing admission control in the distributed platform and, if the contract is accepted, of resource allocation. Even if such approach resembles the one of this work, some differences exist. First, in this work the QoS can be negotiated at different levels of abstraction, i.e. consumers can directly negotiate the share of resources to reserve or can negotiate high-level requirements like the desired response time. Second, this work does not use custom APIs for QoS negotiation but leverages open standards particularly suited to the SOA environment, where QoS parameters are represented in an XML-fashion, allowing providers and consumers to modifying parameters' semantic in a flexible way. Finally, the underlying mechanisms for resource reservation currently support only CPU and wired network, whilst the work of Sojka et al. integrates all the real-time scheduling techniques developed in the context of the FRESCOR (Framework for Real-time Embedded Systems based on COntracts) project (CPU, disk, wired and wireless network). However, we are confident that current state-of-art techniques could be integrated in this work for supporting a broader range of resources.

Also, the work conducted in the context of FRESCOR is based on CORBA (Common Object Request Broker Architecture), even if it is extended with the concept of contract-based scheduling. Other works follows the CORBA-based approach, and in particular the RT-CORBA specification [106] that brings real-time features to CORBA by specifying a priority-based scheme for handling object requests. For example, TAO [88] constitutes a C++ implementation of the Real-Time CORBA specification. Also, TAO has been integrated with QuO [54], a framework that exploits the capabilities of CORBA to reduce the impact of QoS management on the application code. The result [87] is a middleware for adaptive QoS control using real-time scheduling facilities at the computation and network levels.

However, the work presented in this dissertation is based on the SOA paradigm (not on CORBA), which is leveraged in order to achieve important properties such as automatic reconfiguration, location-independence and fault-tolerance. Moreover, the RT-CORBA approaches used to rely on the traditional priority-based scheduling, neglecting issues related to temporal enforcement, 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 [17]. Note that the Dynamic Scheduling extensions to real-time CORBA, also integrated within TAO [55], 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).

This work can also be situated within the real-time service oriented architecture (RT-SOA) research area, that is new and challenging in seeking the confluence

of real-time and SOA, for benefiting from SOA advantages and still addressing the timing requirements that many applications demand while consumed as services. A typical SOA application is composed by many single services, that are bound together for accomplishing to an elaborate goal. Services are put together in workflows, that provide semantic information, usually in an abstract way, for composing services. For achieving predictability to an application, the problem should be considered at two different levels: the service level and the workflow level.

- Service-level predictability: It requires ensuring that a single service has sufficient resources to complete its execution by a given deadline and only involves local resource management.
- Workflow-level predictability: It requires ensuring that all services composing an application complete their execution by the end-to-end deadline and mainly involves orchestration among hosts, and thus a global management.

In the last years some work has emerged following such holistic approach. It is worth to mention the RT-Llama [78] architecture, conceived for supporting predictability in business processes. The RT-Llama architecture, upon receiving a user-requested process and deadline, can reserve resources in advance for each service in the process to ensure it meets its end-to-end deadline. Such architecture contains global resource management and business process composition components. They also create a real-time enterprise middleware that manages utilization of local resources by using efficient data structures and handles service requests via reserved CPU bandwidth. Only recently they have considered the opportunity to apply their framework to industrial systems [59], and in particular to Cyber-Physical Systems (CPS).

Instead, the architecture proposed in this work has been early conceived for industrial systems like factory automation and pervasive environments, and thus specifically addresses some peculiarities. First of all, it only focuses on service-level predictability, given that in a factory plant could be perfectly reasonable to consume only a single service at a time, e.g. for discovering new services or examining device logs. Moreover, a lot of work exists for addressing the problem of time-bounded service composition [35, 39, 38, 75, 98]. Such algorithms could be plugged in the proposed architecture for achieving time predictability on a workflow-level and can constitute the next step for future extensions of this work. Secondly, this work does not rely on advance reservations and thus no assumption is done on the service request arrival time, i.e. requests can arrive in any time and not only in the reserved time-frame. This implies on-line admission control systems for accepting request based on the actual resource allocations, efficient and multitasking request management systems, a fine-grained resource management for avoiding to over-allocate resources to services and, eventually, management of overloads. In particular, the resource management techniques constitute one of the main difference with respect to the Rt-Llama project. As an example, the Rt-Llama project allocates fixed “shares“ of CPU for service executions, whilst in this work

each services has allocated the minimum share necessary to complete its deadline and such share is dynamically computed request-by-request.

The IRMOS Project (which supported part of the research results shown in this dissertation) has also investigated on providing end-to-end QoS guarantees for interactive distributed real-time applications. The IRMOS architecture fulfills the general cloud computing stack that comprises three main layer of service provisioning: (a) The Infrastructure-as-a-Service (IaaS), which refers to the provision of 'raw' machines on which the service consumers deploy their own software (usually as virtual machine images); (b) the Platform as a Service (PaaS), which refers to the provision of a development platform and environment; (c) the Software as a Service (SaaS), which refers to the provision of an application as a service over the Internet or distributed environment. The IRMOS project has developed the necessary infrastructure to allow hosted services and applications to run with predictable levels of QoS while allowing providers to share physical resources among the multitude of applications that will be instantiated at run-time. While trends in resource management in distributed environments tend to focus on best-effort performance or service levels within the very limited scope of a single service or resource (e,g. compile farms or virtual machines), IRMOS advances the state-of-the-art in resource management by allowing users and providers to establish well-defined SLAs including performance terms which are guaranteed on an end-to-end basis. For this sake, computational nodes, network links and storage units are combined. They are all capable of providing temporal guarantees to individual activities, while the underlying physical resources are shared across multiple applications and users.

Some of the results presented in this work, regarding in particular the use of real-time scheduling techniques for respecting temporal constraints in virtualized service components, have been used in the context of IRMOS.

1.3 Contribution

This thesis is the result of three years of research and most of the results have been already published on the *IEEE Transaction on Industrial Informatics* [20] and on several conference proceedings [53, 24, 25, 8, 9]. In this section, the given contribution is summarized as follows.

- *design and development of a real-time SOA with QoS negotiation and management capabilities*. In particular, an effective way to guarantee QoS in service provisioning has been proposed by achieving temporal isolation between high-level software infrastructures and low-level control logic, exploiting a modified Linux kernel supporting real-time scheduling strategies. Moreover, to allow for the configuration of the system at run-time, SLAs have been extended in order to support QoS attributes related to individual activities. Also, the effectiveness of the proposed architecture has been shown by means of extensive experimental evaluations, both quantitative and

qualitative, highlighting that it provides significant and effective advantages over existing solutions.

- *design and development of QoS registry for supporting the QoS management of adaptive service-oriented real-time applications*. It permits to gather persistently QoS data related to different functional behaviors of the application (application modes) and to predict the future performance based on data already collected in the past. A modular architecture allows for defining various models for the prediction of the resource requirements under a set of conditions which has not been observed yet. The registry has been implemented on Linux and overhead measurements are provided for showing viability of the proposed approach. Also, the benefits of using such a framework in a real SOA scenario with QoS provisioning capabilities are described, showing that it allows for a nearly correct resource allocation (self-configuration) while providing QoS guarantees.

- *a methodology to support QoS management for virtualized services deployed in Service Oriented Infrastructures (SOIs)*. In particular, admission control policies are proposed for providing both deterministic and probabilistic guarantees for service activations within a predefined time frame. Moreover, a methodology for scheduling virtualized software components is presented for respecting temporal constraints of individual activities. Experimental results are performed for showing that our methodology, based on the Resource Reservation (RR) scheduling framework, can be effectively applied to the problem.

- *design and development of a service-oriented, flexible and adaptable middleware for QoS configuration and management of Wireless Sensor Networks (WSNs)*. Such architecture supports QoS specification and management by using a contract negotiation scheme based on SLAs; it allows applications to reconfigure and maintain the network during its lifetime and it is independent of the underlying WSN technology. Moreover, it is characterized by an accurate design that permits to both abstract WSNs for a seamless integration into enterprise information systems and address specific low-level features that must be taken into consideration for guaranteeing certain QoS levels. A case study has been also built and presented to show the effectiveness of the proposed solution.

1.4 Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents an architecture for the QoS Negotiation and Management that has been developed for the sake of satisfying timing requirements needed by soft real-time activities. An implementation of such architecture has also been built for the next generation industrial automation platforms and experimental results have been provided

for showing its effectiveness in providing QoS guarantees. Chapter 3 presents a QoS registry that has been also integrated in the proposed architecture for supporting the QoS management of adaptive real-time services. Some experiments have been performed for showing that, by leveraging the proposed registry, the system can self-configure for a better exploitation of internal resources while guaranteeing the QoS required by users. Chapter 4 deals with the problem of providing QoS guarantees (especially regarding the CPU) for virtualized services available in the Clouds. An approach is presented for scheduling virtualized services and it is validated through experimental results. Moreover, novel admission control tests are introduced for service workflows, considering the particular issues in this domain. In Chapter 5 the proposed framework for QoS Configuration and Management has been tied to pervasive environments for designing and building a service-oriented middleware for WSNs. A case-study has been also built for showing the effectiveness of such middleware in enriching WSNs with self-configuration and self-management capabilities. Conclusions and directions for future work are presented in Chapter 6.

Chapter 2

QoS Negotiation and Management Architecture

This chapter focuses on the QoS Negotiation and Management Architecture, developed for the sake of satisfying timing requirements needed by real-time services. In Section 2.2 the design of the proposed architecture is presented and its components are described in details. In Section 2.3 the QoS guarantees supported by the architecture are specified and an insight is given on the mechanisms used for enforcing such guarantees in Linux. Section 2.4 presents the experimental results gathered with an implementation of the proposed architecture in realistic scenarios tied to the industrial automation environment. Such results shows the effectiveness of the proposed architecture in providing services with soft real-time guarantees. The chapter is summarized in Section 2.5.

2.1 Related Work

The QoS architecture presented in this chapter has been firstly conceived for the industrial automation platforms, in the context of the RI-MACS (Radically Innovative Mechatronics and Advanced Control Systems) European research project. Thus, 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, 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 [94]) standard, that is being adopted in the context of existing industrial plants, as documented in the work by Jammes et al. [50].

In a work by Komoda [52], 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 [17] 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 this work.

Note that this work 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 by Lastra and Delamer [56].

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 [76]. 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 by Delamer and Lastra [28], 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.

Considering the actual interest for adopting standard networks in industrial automation plants, this work considers the provisioning of network guarantees in Switched Ethernet (SE). In the latter years, a lot of work has been done for providing predictability in the communication over SE. Some approaches focus on the behavior of available commercial switches, that, while offering helpful features in term of timeliness improvement (like traffic priority management), still have many limitations. As an example, the work of Pedreiras et al. [80] highlights how traffic scheduling on a priority-based fashion cannot guarantee predictability when switches are overloaded, as lower-priority traffic may lock the switch memory, causing high-priority traffic to be dropped. To overcome the limitations of commercial switches, many works propose to enhance switches with real-time

scheduling capabilities. Among them, we can cite the EtheReal switch [100], the work by Hoang et al. [46] and the work by Wang et al. [105]. Also Lo Bello et el. [62] suggest that the fuzzy traffic smoothing technique they developed for classic Ethernet could be effectively adopted inside switches for guaranteeing soft real-time constraints.

Some other approaches focus on developing network protocols for achieving timeliness on SE. One of the most promising protocol for real-time communication is the FTT-SE protocol (Flexible Time-Triggered communication over Switched Ethernet) [67], that leverages a master-slave approach for controlling the traffic load submitted to the network and offers support for arbitrary scheduling policies. This feature can be exploited to realize real-time scheduling supported by adequate utilization-based schedulability tests that can be performed on-line [68].

Another class of techniques for controlling the load submitted to the network consists of using traffic shapers in each node. The work of Loeser and Haertig [63] falls in this category, showing that traffic shaping can be used to achieve reliable packet transmission with bounded transmission delay. Our work follows the latter approach, as we believe it perfectly fits to our QoS architecture, comprised of many consumers and a single provider, that generates all the traffic. In fact, by using traffic shaping, we can control the amount of traffic going out to each consumer, enforcing guarantees that can be individually negotiated.

Our work leverages the traffic control mechanisms of Linux TC (Traffic Control) [47] for determining the way in which a node can send its network traffic. An evaluation of SE and Linux TC for real-time transmission has been also performed in a work by Vila-Carbó et al. [101], that highlights how traffic control mechanisms applied in network interfaces and in switches can provide good levels of predictability. An analogous Linux TC configuration is used by the same authors [102] for extending FRESCOR with the concept of Classes of Service (CoS). These two works are related to the joint transmission of highly variable bit rate streams and high-priority real-time periodic traffic.

Instead, our work does not focus on the transmission of network traffic of different priorities but on providing guarantees about the network bandwidth used in the communication between a service provider and service consumers. Each communication flow between provider and consumers has the same priority but each consumer can negotiate a different network bandwidth. Thus, the provider must enforce such guarantee avoiding that concurrent service requests can “steal” the reserved bandwidth for that client.

2.2 Design

In this work, a QoS negotiation and management architecture is proposed, which allows clients to negotiate QoS parameters that will be honored during service provisioning. Regarding this founding functionality, the proposed architecture can be divided in the following layers, as highlighted in Figure 2.1:

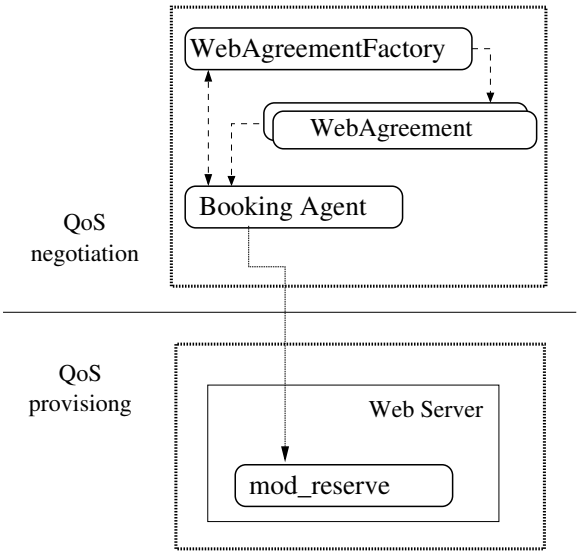

Figure 2.1: QoS Negotiation and Management Architecture

1. the QoS Negotiation Layer, in which the QoS parameters of provided services are negotiated,

2. the QoS Provisioning Layer, which takes care of providing services while respecting negotiated guarantees.

Such decoupling permits to achieve many advantages:

- the negotiation can comprise high-level parameters that can be related to business aspects and not only to resource availability;
- negotiation and provisioning could be performed by different machines, enabling the possibility of using a cluster for providing services;
- each layer can be implemented with different technologies, providing compatibility only at the connection points between each layer.

2.2.1 QoS Negotiation Layer

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 [10], 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) a protocol to verify the run-time compliance of contracts. WS-Agreement was

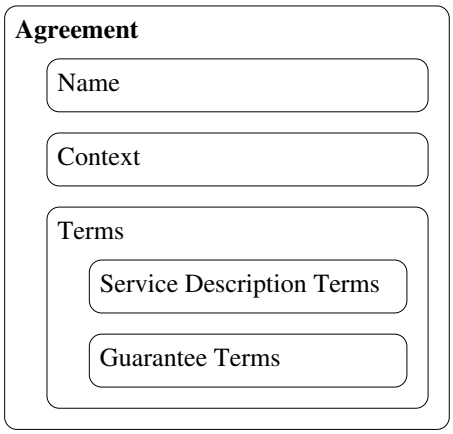

Figure 2.2: Agreement structure in WS-Agreement

chosen in this context not only for its flexibility in comparison with other QoS-enabled technologies (like WSLA [64]), but also for its standard nature (it is supported by the Open Grid Forum), which may ensure penetration of the platform within many sectors.

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. The key parts of an Agreement can be summarized as follows:

- `Name` - Field exploited to name an agreement in human-readable way.
- `Context` - Section exploited to store various meta-information related to the agreement
- `Terms/ServiceDescriptionTerms` - Section exploited to describe services related to the agreement. Each Service Description Term (SDT) can describe (fully or partially) one of the provided services.
- `Terms/GuaranteeTerms` - Section mainly used to specify guarantees to assure in providing services described in the corresponding SDTs.

This work support the specification of QoS parameters at different levels of abstraction. This chapter deals with low-level parameters regarding resource utilization, as specified in Section 2.3, whilst Chapter 3 deals with an example of high-level parameters, i.e. the service response time.

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 realized 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 2.3, 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 WebAgreementFactory 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.

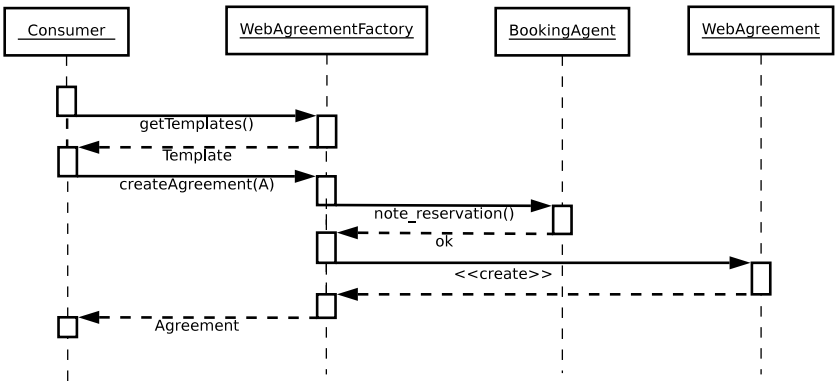

Figure 2.3: Successful Agreement creation

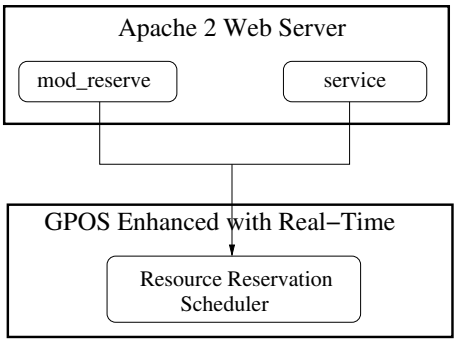

Figure 2.4: The *mod_reserve*

2.2.2 QoS Provisioning Layer

The QoS Provisioning Layer provides services with the QoS guarantees defined in the negotiation phase. A web server is responsible of receiving service requests and handling them through the common cycle of receiving-processing-responding. It executes in a GPOS enhanced with the RR framework, providing the real-time features that permit to enforce QoS guarantees. Actually, the architecture supports the allocation of CPU ”shares” and network bandwidth “shares”.

This has been implemented, in the architecture, by the *mod_reserve* component and the *NetReserve* component.

In particular, the *mod_reserve* [1] is a resource reservation module for the Apache 2 web server. It uses the web server functionalities for receiving and processing service requests, then it reserves the actual resources by using the available API for accessing the RB facilities available in the underlying scheduler (see Figure 2.4). Apache 2 is a very popular web server and it is easily extensible thanks to its internal modular structure: this allowed for the realization of *mod_reserve* as a

[1] The mod_reserve is available for downloading at the URL `http://freecode.com/projects/mod_reserve`

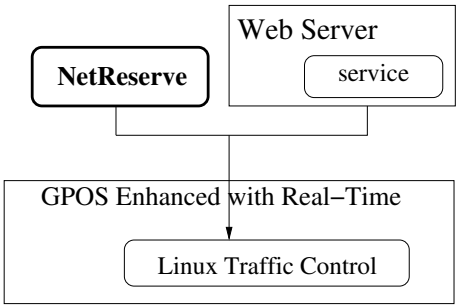

Figure 2.5: Network bandwidth reservation components

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 *mod_reserve* uses the user-space library made available through the AQuoSA framework, that enhances the Linux kernel with a real-time scheduling policy based on Earliest Deadline First (EDF). This way, the *mod_reserve* 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 *mod_reserve* has the following internal structure:

- The *WebServer Interface* uses the 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 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 misbehavior of one or more clients (or services).

The *NetReserve* component, depicted in Figure 2.5, has been instead designed for allocating network bandwidth reservations to each consumers. Such allocations have been managed through a consolidated Linux kernel module, generically called as Linux TC.

2.3 QoS Guarantees

In this section, the QoS guarantees provided for services by the proposed architecture are defined through the specification of proper parameters. On this basis, some admission control tests are also introduced. Finally, the mechanisms leveraged for enforcing QoS guarantees are briefly described.

2.3.1 Specification

The QoS negotiation framework allows applications to negotiate QoS parameters related to the resource allocation of CPU and network during service provisioning. Such parameters are defined through the XML Schema [104] language and are stored in the `ServiceDescriptionTerm` section (see Section 2.2.1) of an agreement proposal.

The QoS parameters related to the CPU allows applications to access to the underlying RB scheduling framework. Such framework provides the fundamental property of *temporal protection* (or 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 utilization, so that its ability to meet timing constraints is not influenced by the presence of other tasks in the system. 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 (interested readers may refer to Section 2.3.3 for further details on the low-level CPU enforcement mechanisms).

The set of parameters transmitted in an Agreement are specified as follows.

- `CpuMinBudget`. This parameter represents the minimum time units Q_{min} of CPU usage requested by a client. Such value is considered for every period defined by the `CpuPeriod` parameter. If a contract is accepted, the provider always guarantees this value, independently of other concurrently running activities.

- `CpuMaxBudget`. This parameter represents the maximum time units Q_{max} of CPU usage requested by a client every time-frame defined by `CpuPeriod`. Such value represents the maximum budget that is never exceeded.

- `CpuPeriod`. This parameter represents the duration of the period P for which the budget is specified.

```
<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 2.6: XML fragment of an Agreement for negotiating CPU allocations

```
<wsag:ServiceDescriptionTerm
         wsag:Name="server_parameters"
         wsag:ServiceName="use_of_web_server">
  <ret:ServerParams xmlns:ret="schemas.retis">
       <ret:NetMinBandwidth
         unit="Mbps"> 19 </ret:NetMinBandwidth>
       <ret:NetMaxBandwidth
         unit="Mbps"> 19 </ret:NetMaxBandwidth>
  </ret:ServerParams>
</wsag:ServiceDescriptionTerm>
```

Figure 2.7: XML fragment of an Agreement for negotiating network bandwidth allocations

A representative XML fragment, in which the CPU allocation to negotiate is $9ms$ every $100ms$, is shown in Figure 2.6.

Instead, the QoS parameters related to the network bandwidth allocated to consumers during service provisioning are described as follows:

- `NetMinBandwidth`. This parameter represents the minimum network bandwidth requested by a client. If a contract is accepted, the provider guarantees to allocate at least such bandwidth value in the communication with the consumer.

- `NetMaxBandwidth`. This parameter represents the maximum network bandwidth that can be allocated to a client. If the provider has some spare bandwidth capacity, it could be allocated to a client up to reach this limit in the total amount of the reservation.

A consumer can specify the required network QoS level by storing the relative parameters in a `Service Description Term` of an agreement proposal. An example can be found in the XML fragment of Figure 2.7, in which the client requests a guaranteed network bandwidth of $19Mbps$.

2.3.2 Admission Control

The service provider performs admission control for deciding if the requested network bandwidth could be allocated or not. Such admission control test can be formalized by introducing the following notation.

Let B^{eth} be the bandwidth of the provider network interface(s). The provider will be configured for dedicating a fraction of B^{eth} to the traffic going out to service consumers that use the QoS framework. We will use the term reserved network bandwidth utilization (denoted by U^{res}, with $U^{res} \leq 1$) to represent such fractional value. Instead, the unreserved bandwidth utilization is denoted by $U^{oth} = (1 - U^{res})$. Denoting $C = \{c_1, c_2, \ldots c_n\}$ as the set of service consumers that have successful negotiated a network bandwidth reservation, the provider keeps in memory the pairs $\{U_c^{min}, U_c^{max}\}$ with $1 \leq c \leq n$. Instead, the network bandwidth utilization currently allocated to the consumer c is denoted with U_c. Such value is computed according to the policy of the provider. As an example, a possible assignment that fairly distributes the spare bandwidth capacity could be the following:

$$U_c = min\left\{\left(U_c^{min} + \frac{U^{res} - \sum_{i=1}^{n} U_i^{min}}{n}\right), U_c^{max}\right\} \tag{2.1}$$

The minimum network bandwidth requested by a consumer through the parameter `NetMinBandwidth` is denoted by B_r and thus $U_r = B_r/B^{eth}$ denotes the minimum network bandwidth utilization requested by a consumer. The maximum network bandwidth requested through the `NetMaxBandwidth` parameter is instead denoted by B_r^{max} and $U_r^{max} = B_r^{max}/B^{eth}$ denotes the maximum network bandwidth utilization requested.

A consumer request can be accepted iff the admission test of Eq. 2.2 holds.

$$\sum_{i=1}^{n} U_i^{min} + U_r \leq U^{res} \tag{2.2}$$

Moreover, the preliminary constraints $U_r \leq U_r^{max} \leq U^{res}$ must hold too. In our QoS architecture, the admission control is performed by the *BookingAgent* component, as described in Section 2.2.1. In particular, a sub-component named `BookingNet` has been developed and integrated for the purpose of performing the test of Eq. 2.2. If such test is positive, the `BookingNet` communicates through an internal socket with its counterparts in the QoS Provisioning Layer (see Figure 2.5), which is responsible for allocating the negotiated amount of network bandwidth to the corresponding service consumer.

2.3.3 Enforcement in Linux

The proposed architecture focuses on QoS negotiation and management and does not introduce new mechanisms for the enforcement of QoS guarantees. Instead,

current state-of-art mechanisms are leveraged for providing CPU and network guarantees in a GPOS (i.e. Linux). They are described in the following.

AQuoSA

In this work, AQuoSA [77] has been leveraged for its capability of enhancing Linux with real-time features, especially concerning the CPU scheduling. AQuoSA is a complex layered architecture for the QoS control of time-sensitive applications and its main component can be described as follows.

- the Generic Scheduler Patch (GPS), that permits to extend the Linux scheduler by intercepting scheduling events and executing external code in a kernel module;
- the kernel abstraction layer (KAL), a set of C macros that abstract the additional functionality we require from the kernel, e.g. the ability to measure time and set timers, ability to associate data with the tasks, etc.;
- the QoS Reservation component, composed of a kernel module and an application library communicating through a Linux virtual device:
  - the RR module implements an EDF scheduler, the RR mechanism (running on top of the EDF scheduler) and the RR supervisor; a set of compile-time configuration options allows one to use different RR primitives and to customize their exact semantics
  - the RR library provides an application programming interface (API) allowing an application to use RR functions;
- the QoS manager component, composed of a kernel module, an application library and a set of predictor and feedback sub-components.

The proposed approach leverages the API provided by the QoS Reservation component for achieving temporal isolation. Thanks to this property, tasks can be thought of as running on a "virtual" CPU whose speed is a fraction Q/P of the CPU speed (notation has been introduced in Section 2.3).

In particular, such API is used in the *mod_reserve* component for creating "virtual" CPUs and assigning them to the web server tasks. The way this assignment is performed can vary depending on the provider's policy. As an example, the provider can differentiate the QoS on a per-client basis and thus a CPU virtual resource is assigned to each client and "survives" trough multiple connections (see also Section 2.2.2). Instead, the provider may want to offer a certain QoS only to some requests and thus the QoS management would be done on a per-request basis, by assigning a virtual resource for the management of each request.

Linux Traffic Control

The allocation of network bandwidth to each service consumer has been managed through a consolidated Linux kernel module, generically called as Linux TC. Such module provides some mechanisms for rearranging traffic flows and scheduling packet transmissions, by using the abstractions of *queuing disciplines*, *classes* and *filters*.

- A *queuing discipline* (qdisc) is a scheduler that rearranges packet queues. Qdiscs can be classful, in the sense that they can contain classes and provide a handle to which to attach filters. Otherwise they are classless and cannot contain classes, nor is it possible to attach filter to them. As an example, the default Linux qdisc (a prioritized FIFO scheduler called pfifo_fast) is a classless one.

- A *class* is an object that can only exists inside a classful qdisc. It can contain children classes and can have an arbitrary number of filters attached to it. If a class is a terminal one (also called a leaf class), it must contain a qdisc, responsible to send data from that class.

- A *filter* is an object that permits to classify packets in output queues. A filter must specify a classifier, which can be used to classify a packet basing on some information contained in the IP header.

For the purpose of our work, we will focus on the Hierarchical Token Bucket [30] (HTB) qdisc, that can be basically configured by using the following parameters:

- *rate*. Maximum rate this class and all its children are guaranteed.

- *ceil*. Maximum rate at which a class can send, if its parent has bandwidth to spare. The default value is equal to *rate*.

In the proposed approach, HTB is used both in the root qdisc and in the internal classes. In particular, a parent class is created with parameters $rate = ceil = B^{eth}$, allowing children classes to borrow spare bandwidth capacity from it. Then, a leaf class is created and associated to each service consumer that has successful negotiated a bandwidth reservation. Such class is configured with the negotiated parameters, by specifying $rate = B_r$ and $ceil = B_r^{max}$. The remaining traffic is managed by an additional leaf class configured with $rate = U^{oth} * B^{eth}$ and $ceil = rate$. In this way, unreserved traffic has a guaranteed bandwidth chosen by the system administrator and can be managed without compromising existing guarantees.

The *NetReserve* has been developed in the QoS Provisioning Layer for dynamically creating (or deleting) the HTB classes when network bandwidth reservations are created (or deleted). It leverages the functionalities of *tcng* [6] (Traffic Control - Next Generation), which permits to write configuration scripts in a flexible and

```
#include "fields.tc"
#include "ports.tc"
#define INTERFACE eth0

dev INTERFACE {
 egress {
   class (<$rewww1>) if tcp_sport==8080;
   class (<$rewww2>) if tcp_sport==8081;
   class (<$rewww3>) if tcp_sport==8082;
   class (<$rewww4>) if tcp_sport==8083;
   class (<$rewww5>) if tcp_sport==8084;
   class (<$other>)  if 1;

   htb () {
     class (rate 100Mbps, ceil 100Mbps){
      $rewww1=class(rate 19Mbps) {sfq;};
      $rewww2=class(rate 19Mbps) {sfq;};
      $rewww3=class(rate 19Mbps) {sfq;};
      $rewww4=class(rate 19Mbps) {sfq;};
      $rewww5=class(rate 19Mbps) {sfq;};
       $other=class(rate 5Mbps, ceil 5Mbps){sfq;}
   }}
}}
```

Figure 2.8: tcng configuration for enforcing network bandwidth allocations

user-friendly way compared to the *tc* tool (contained in the iproute2 suite of utilities), that instead provides a command line syntax that is very complex and thus error-prone. Mainly, tcng adds another layer of abstraction above tc, being a compiler which takes configuration scripts written in the tcng language, translates them into an internal representation, and then generates commands in the tc language.

The typical tcng script dynamically generated by *NetReserve* is illustrated in Figure 2.8. It depicts a situation in which five network reservations are in place and all have been negotiated with the `NetMinBandwidth` and the `NetMaxBandwidth` equal to $19Mbps$. Each network reservation is characterized by a unique identifier, that also constitutes the HTB class name. At the packet level, the traffic directed to a specific service consumer is classified by means of filters. As an example, in the configuration of Figure 2.8 such filter is based on the provider port used by the client for requesting services. It could be worth to note, that in the depicted situation, the system is configured with $B^{eth} = 100Mbps$, $U^{res} = 0.95$.

2.4 Experimental Results

The experimental evaluations described in this section focus on the verification of the behavior of the proposed architecture, 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 us-

ing appropriate real-time scheduling techniques. The experiments have been conducted by submitting service requests to a server machine equipped with a GPOS enhanced with real-time capabilities. In particular, a GNU/Linux OS has been used, featured by a 2.6.28 kernel configured with high resolution timers ($1000Hz$) and patched with the GSP provided by AQuoSA (see Section 2.3.3). The server machine hosts a multiprocessing version of the Apache 2.2 web server, enhanced with the *mod_reserve* module.

The service requests were generated by different clients using the Apache Benchmark tool[2].

2.4.1 CPU Guarantees

In the following experiments, scenarios that are well-suited to the industrial automation field have been set-up. The described scenarios are built so as to "mimic" typical image-processing services that may be needed in complex vision-based control logic.

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; binarization 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.

Note that the service needs to be provided respecting timing guarantees even if the PC gateway, which provides services through an Apache 2 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 web server executing requests, its behavior has been verified both using an unmodified Apache 2 web server and an Apache 2 enhanced with the *mod_reserve*. 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

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

Processing times	Original web server	*mod_reserve* server
min	119	115
avg	198	173
max	1175	273
std.dev	117	23

Table 2.1: Service response times (ms) for the centroid detection scenario

values, the standard deviation and 90% confidence intervals.

Results obtained for the unmodified web server are reported in the first column of Table 2.1 (90% confidence interval is 7.5%), whereas results obtained for the web server containing the *mod_reserve* 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 2.1 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 *mod_reserve* successfully managed to always respect the design constraint: this behavior 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.

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 the work by Su [97]. In particular, a simple image rotation algorithm has been chosen for the experiment.

Also in this case the server behavior has been verified both using an unmodified Apache 2 web server and an Apache 2 enhanced with the *mod_reserve*. 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 web server was configured to serve 10 requests concurrently with 10 different tasks and each task was assigned by the *mod_reserve* 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 behaviors of the two configurations, as depicted in Figure 2.9.

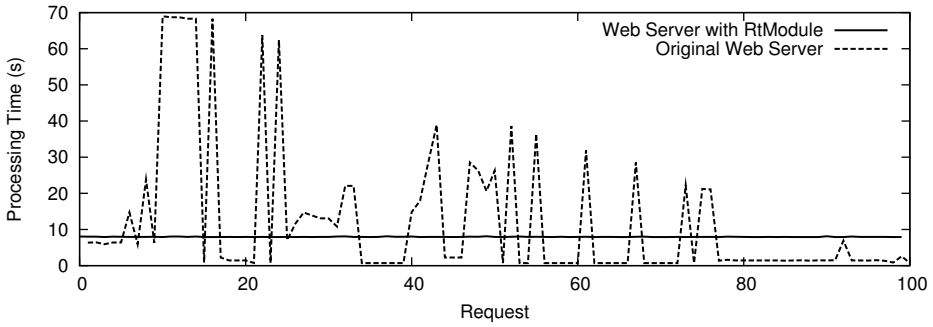

Figure 2.9: Response times obtained with and without mod_reserve

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 *mod_reserve*, which exhibit an unpredictable behavior. This can be explained by the fact that the resource reservation techniques implemented in the *mod_reserve* provide a dedicated slower virtual processor for each service instance. Therefore, with *mod_reserve*, each service instance has an almost constant response time (the continuous line in Figure 2.9), because it has been *reserved* a fraction of the real processor. On the contrary, without *mod_reserve*, due to the lack of temporal isolation, the service response time can exhibit significant fluctuations (the dashed line in Figure 2.9), if the processor is subject to concurrent requests.

2.4.2 Network Guarantees

Some experiments have been also performed for evaluating the effectiveness of the proposed approach in providing network QoS guarantees for concurrent service requests.

Two machines, a client and a server, connected by a switch through $100Mbps$ Ethernet links. The switch is a typical Commercial Off-The-Shelf (COTS) one (the AT-8024 by Allied Telesis), with a buffer memory of $6MB$ and store-and-forward mode. The server machine acts as service provider and it is featured by a $64bit$ Intel CPU running at $1.2GHz$.

The image rotation service described in the previous section has been modified for sending the rotated image through the network (given a certain resolution r and a rotation angle α). In these experiments, the image to rotate is a gray-scale one ($8bit$ per pixel), provided by means of Ram-disks with different resolutions. The client machine creates 5 tasks acting as service consumers and each one connects to the provider by using a different port. In these experiments, the consumers request the service with parameters $r = 512$x512 and $\alpha = 20$. The rotated image, sent as response for each request, has a size of $419KiB$.

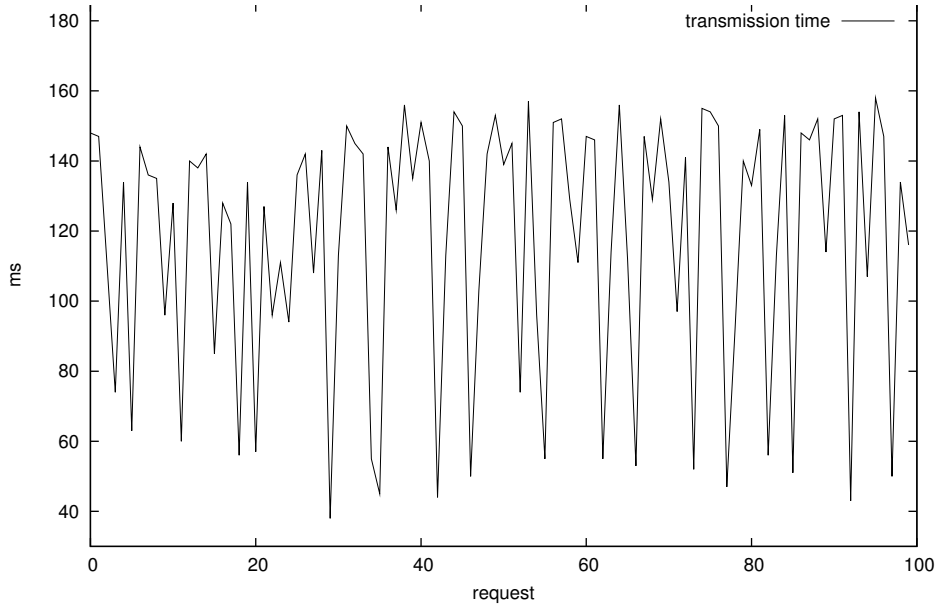

Figure 2.10: Transmission time of service response

We are interested in measuring the satisfaction perceived by consumers and thus all the results are collected by the client. In particular, we measured the transmission time of the service response by leveraging the report functionality of the *ab* tool. For each request, *ab* reports the so-called `wait` and `dtime` values, being respectively the time interval the consumer waits on the socket and the time interval that occurs between the request sending and the last byte received. The transmission time of the service response has been measured as the difference $dtime - wait$.

In Experiment 1, the five consumers concurrently request the service every 4 seconds for 20 times and the collected service transmission times have been plotted in Figure 2.10. It could be seen that there is an high variability in such values, mainly due to resource contentions in the network queue of the service provider. Thus, very different transmission times of the image are experienced by each consumer. Such variability can be also observed in the processing times of each request and, as highlighted in the previous section, it is due to the contention in accessing the CPU.

The behavior pointed out in Experiment 1 is emphasized in presence of interference in the network, as reported by the following Experiment 2. In this case a bandwidth eager task was created for sending UDP packet of size $12.5KB$ every $1ms$, trying to saturate the available bandwidth of $100Mbps$. The bandwidth eager task, illustrative of possible misbehaviors inside the provider, was running while the consumers requested the service as of the previous experiment. The collected transmission times are greater than those reported in Experiment 1, as can be seen by comparing the first two columns of Table 2.2, that reports minimum, average,

	Exp. 1	Exp. 2	Exp. 3	Exp. 4
min	38	124	186	222
avg	117.95	195.3	191.3	230.6
max	158	226	193	237
std.dev	36.63	26.28	1.51	3.03

Table 2.2: Service transmission times (ms) for Experiments 1-4

maximum and deviation standard values (the 95% confidence values are always below the 2.2%).

The third experiment consists in repeating the previous one when the service provider is configured by using the traffic control script of Figure 2.8, that depicts a situation in which a bandwidth allocation of $19Mbps$ has been assigned to each of the 5 consumers. It is worth to note that each consumer can negotiate a different bandwidth share with respect to others and we have only chosen such configuration for the sake of simplicity. The results show a substantial flattening of the transmission times in comparison with Experiment 2. It can be appreciated in Figure 2.11, that shows results collected without any tc configuration with label *no-tc* and the others with label *using-tc*.

The flattening of the service transmission time enhances the system with a certain degree of determinism and allows each consumer to experience the same QoS level across various requests over time. By repeating Experiment 3 without the interference in the network, we obtained a mean value of $182ms$ and a standard deviation of $0.66ms$, discovering that the transmission times are not influenced by concurrent requests of other consumers but they are minimally affected by the bandwidth eager task. This points out some limitations of the Linux TC packet scheduler in handling overloads and suggests the adoption of some precautions, e.g. redefining U^{oth} in a manner that $U^{res} + U^{oth} \leq 1$ (see Section 2.3.2). Some experiments, not reported due to space constraints, endorse the adoption of such trick, showing an improvement of up to $8ms$ in the mean transmission time by progressive lowering U^{oth} in a manner that $U^{res} + U^{oth}$ ranges from 1 to 0.95.

Another experiment (named Experiment 4) has been performed for pointing out a particular behavior of Linux TC. It consists of using the script of Figure 2.8 without specifying the `other` class, assigned to unreserved traffic. In this way, Linux TC does not find any matching rule for the UDP traffic, that is left out of control. The results, reported in the fourth column of Table 2.2, show that transmission times of reserved traffic are always flat in comparison of those of Experiment 2, but the mean value is greater than the value of Experiment 3, because the unreserved traffic is not shaped.

Another experiment has been performed for verifying the precision of the HTB algorithm in allocating network bandwidth for each class. Basically, the same configuration of Figure 2.8 has been applied, by varying the `rate` parameter assigned each time to all the five reserved classes corresponding to the consumers. In this

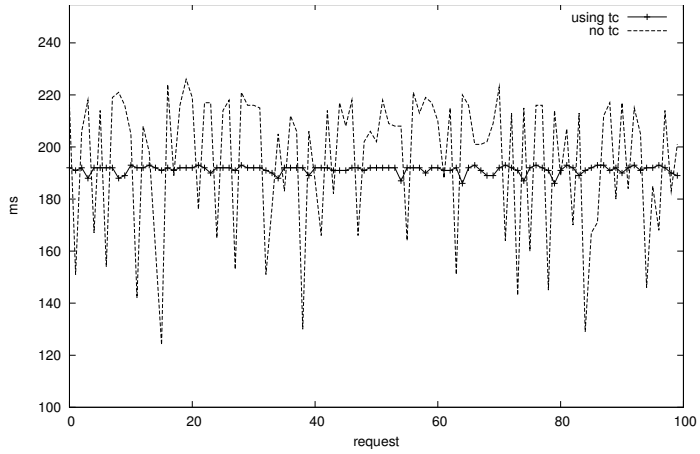

Figure 2.11: Comparison of service transmission times for overloaded network

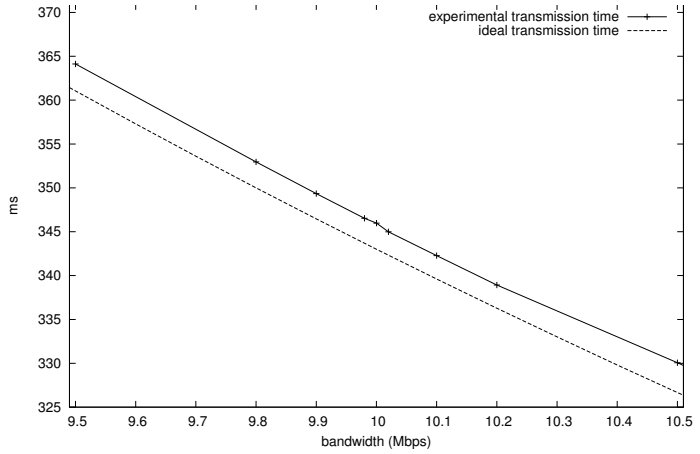

Figure 2.12: Variation of service transmission times for different bandwidth assignments

case, results have been collected by establishing a point-to-point connection between client and server, with the purpose of not adding the switch overhead to measurements. In particular, the mean values of the response transmission time have been plotted in Figure 2.12, together with the ideal curve $f(bw) = S/bw$ representing the transmission time as a function of the bandwidth bw (with $S = 419KiB$ be a constant equal to the rotated image size sent through the network). It could be seen that the experimental curve is very close to the ideal one and the bandwidth assignment, tested at different grain levels ($1Mbps$, $0.1Mbps$, $0.01Mbps$), has an acceptable level of precision.

2.5 Summary

This chapter addressed the design and development of a SOA for supporting real-time and QoS aspects, with particular reference to the next generation industrial automation platforms. The architecture of the proposed framework has been described, and its effectiveness in guaranteeing QoS in service provisioning 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.

In particular, an effective way to guarantee QoS in service provisioning has been proposed by achieving temporal isolation between high-level software infrastructures and low-level control logic, exploiting a modified Linux kernel supporting real-time scheduling strategies. Moreover, to allow for the configuration of the system at run-time, the WS-Agreement protocol has been extended in order to support QoS attributes related to individual activities.

Chapter 3

QoS Registry

Service-oriented real-time applications are commonly executed in open systems, where they can be activated and terminated in any moment, generating a time-varying workload. For these reasons, classical real-time systems design methodologies are rarely used in this context. Instead, the use of adaptive techniques is more suitable for managing the Quality of Service (QoS) of service-oriented real-time applications, given that such techniques have the advantage of not requiring an offline analysis of the application workload and provide the flexibility needed in these cases.

The most common adaptive techniques can be classified as follows: *application-level* adaptation, in which the application operating modes are adapted to the availability of resources; *resource-level* adaptation, in which the resource shares granted to the applications are adapted to the dynamic workload requirements; and *power-level* adaptation, in which the resource speed, and thus the corresponding power consumption, is adapted to the requirements of the system. Though adaptation techniques belonging to each group can be reasonably used by themselves, a better approach for QoS control counts to use these techniques in conjunction [21] and as part of an integrated QoS framework that contains the set of required mechanisms for QoS management [40]. Moreover, a particular care must be done when providing QoS guarantees for services, as concurrent activations can easily disrupt the response time of a service, as shown in Section 2.4.

Adaptive techniques for QoS management achieve self-configuration capabilities by relying on previously collected information about the application configuration, its achieved performance and the corresponding resource requirements. For example, data from previous executions may be used in a control loop for adjusting the resource allocation for future executions. However, in SOA environments, the application might be instantiated on a request-by-request basis by a web server, and with potentially different parameters or operating modes, making it difficult to build such an on-line control loop. Also, in presence of a multitude of operation modes and environmental conditions, it may be cumbersome to build a historical data set which is comprehensive of all the possible cases for future instantiations

of the application. For these reasons, we believe that the use of a well-structured framework for handling QoS data is of paramount importance in such a context.

In this chapter we present QoSDB, a QoS registry for supporting QoS management in SOAs. It can be exploited for gathering persistently QoS data related to different functional behaviors of the application (application modes) and for predicting the future performance based on data already collected in the past. Furthermore, a modular architecture allows for defining various models for the prediction of the resource requirements under a set of conditions which has not been observed yet. This allows for achieving a nearly correct resource allocation (self-configuration) for the application with a great reduction of the needed observation/benchmarking points, especially in those contexts in which the space of possible configuration parameters is big (e.g., multimedia applications supporting arbitrary resolutions).

In order to show viability of the proposed approach, we provide overhead measurements gathered on an implementation of the QoSDB on Linux. Moreover, through some experiments we highlight the benefits of using such registry in a real SOA scenario with QoS provisioning capabilities. By leveraging the QoSDB, the system under study is capable of auto-tuning for a better exploitation of internal resources while guaranteeing the QoS required by users.

In the remainder of this chapter, related work is analyzed in Section 3.1, whilst Section 3.2 describes the architecture of the proposed QoS registry. Section 3.3 focuses on the interface exposed to the application, and the typical usage pattern. Section 3.4 shows experimental results gathered with an implementation of the proposed registry. Finally, Section 3.5 draws conclusions.

3.1 Related Work

In SOA applications the importance of historical data and statistics for supporting the QoS is commonly recognized and leveraged [90, 111, 107]. However existing approaches rarely deal with real-time services in dynamic environments, where respecting time requirements imposes a particular care. For example, the use of historical data for SOAs has been also exploited by Yu and Lin [108], that propose a QoS-capable Web service architecture (QCWS) by deploying a QoS broker between Web service clients and providers. This broker uses QoS information collected from each server for choosing the best provider that can satisfy client requests. However, this information is mainly static and it is not used to make application-level or resource-level adaptation. Instead, our experimental work focuses on adapting the resource shares for the enforcement of certain QoS guarantees.

In the QoS management of adaptive SOAs, the use of QoS prediction mechanisms is of paramount importance for understanding the trend of QoS data and reacting to changes in the application and/or in the execution environment. In our work such aspects has been considered in the design phase and analyzed in the

experimental section, however proposing novel prediction techniques is not in the scope of this paper. For the sake of simplicity, a linear regression model has been used in our experiments but, in principle, any QoS prediction technique can be embedded in the proposed work, like the WS-QoSP [58] approach based on forecast combination.

As a note, the QoS registry proposed in this work has been conceived for supporting service providers in the QoS management and thus its information is not produced with the intent to be published for discovery and/or integration processes. Some work in this direction has been done by Lee [57] for representing QoS information of services in UDDI [19] (Universal Description, Discovery and Integration) registries.

In the realm of real-time systems other well-structured framework for management of historical data exist. Among them, it is worth to mention the BACC [7] (Budget ACCountant) module inside the HOLA-QoS framework [41]. That module provides the basic means for enforcing and accounting for resource usage, by notifying task overruns and by keeping statistical information on the used task budgets. It stands at the Operating System (OS) level and the information provided by it is directed to monitoring tasks for checking how a task is behaving. Instead, our framework stands at a higher level than the OS, supposing the existence of a real-time enhanced OS for the QoS enforcement. In this way, it can be used for adapting the application performance by exploiting high-level information affecting the QoS, rather than for simply checking a possible misbehavior.

For the QoS management of soft real-time applications, the use of adaptive techniques is not new and some approaches have been recognized as effective, especially for adaptive scheduling. For example, the work by Abeni et al. [3] applies feedback control to a RR scheduler for dynamically adapting the CPU bandwidth each task should receive. Also, in a work by Eile et al. [45], feedback scheduling is used for the design and implementation of a CPU Broker, that adjusts allocations over time to ensure that high application-level QoS is maintained. Further, in the context of feedback-based real-time scheduling, Cucinotta et al. [2] introduced a clear separation between the prediction algorithm, responsible for estimating the workload of the subsequent task activation(s), and the control algorithm itself, leveraging the output of the predictor and the knowledge about the current task delay, for deciding the next allocation. Instead, our work does not propose any new feedback strategy but focuses on the collection and management of historical data for adaptive systems. Moreover, our proposal can be used for supporting feedback scheduling by clearly separating the feedback algorithm from the data management.

Finally, it is worth to note that, differing from some other work like that of Sojka and Hanzalek [92], in our experimental evaluations we do not take into consideration admission control and contract negotiation. Instead, the focus is on showing the effectiveness of the proposed framework in supporting adaptation techniques for the QoS Provisioning Layer introduced in Section 2.2.

3.2 Design

This section describes the design of the proposed QoS registry, named QoSDB. In the most simple case, QoSDB operates at the application level, shared between all the tasks of an application. In more complex architectures, a middleware acting as QoS manager could be interposed between the applications and the OS. In that case the QoSDB would stand in the middle layer and its functionalities would be better exploited by the QoS manager.

The main features of QoSDB can be summarized as follows:

- it supports adaptation techniques by counting that each application can be characterized by different modes and can run at different resource speeds;
- it allows applications to predict QoS parameters for application modes that have not yet been experienced by the application.
- it permits to store in memory, save in a database and recover statistics related to QoS parameters;

The QoSDB framework has been designed pursuing the following goals.

Modularity. QoSDB is characterized by different plugins, for permitting a rapid change in the functionalities of the framework. Moreover, even the core has been designed by keeping in mind modularity for a clean competence separation. This will eventually allow programmers to modify the internal data structures and algorithms with a minimum effort.

Flexibility. QoSDB has been designed for exploiting the three different levels of adaptation in conjunction. However, it is flexible enough for being used in many contexts, even if only a subset of the provided functionalities is required.

Efficiency. The overhead introduced by our framework should be negligible.

For pursuing this goal, the framework has been developed in the C language. This also widens the possibilities of usage in the context of embedded applications for industrial automation, without precluding the possibility to build gateways towards different programming languages, if needed.

3.2.1 Model

In this chapter we focus on a generic application, software component or service α, which is capable of switching among a set of operating modes M. The behavior of α is affected by a set of parameters $\{\mathsf{in}_j : j = 1, 2, \ldots\}$ (for example, for an image processing application, these can be the image resolution and color depth in bits, for an interactive application they can be represented by the current workload in terms of connected users, etc.). However, for the sake of simplicity, we assume that the set of parameters may be mapped to a scalar quantity $v = f(\{in_j\})$, so that the actual impact on the application behavior (in terms of resource requirements and

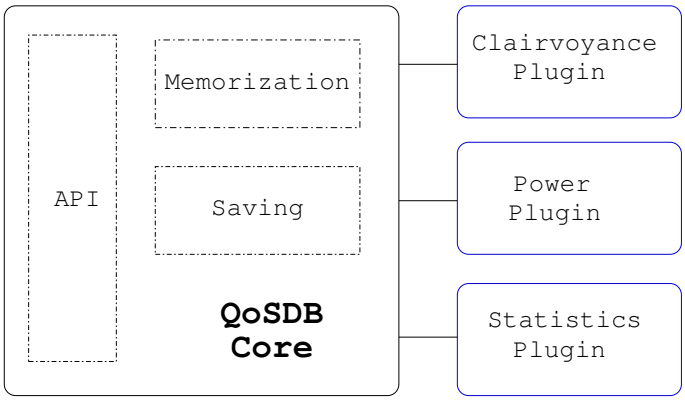

Figure 3.1: The QoSDB Architecture

performance) depends only on v. For example, for an application that operates on images, v could be computed as the image size ($\mathsf{width} * \mathsf{height} * \mathsf{depth}$). Thus, each operating mode $m \in M$ is characterized by a scalar quantity v_m.

Also, the historical behavior of α in each mode m is characterized in terms of a vector of observed samples sp_m with maximum dimension N_S, which constitutes a moving window over the past history of the system. Also, a vector of statistics $\mathsf{st}_m = (\mathsf{st}_{m,1}, \ldots, \mathsf{st}_{m,N_T})$ is associated with these observations, where each element of st_m is basically computed by performing certain operations on the elements of sp_m (e.g., moving average). The number of samples N_S and the number of statistic N_T are equal for each m and can be defined directly by α.

The resources used by α can be in a different power consumption mode pm. Thus, each resource has associated a set of power modes $PM = \{\mathsf{pm}_1, \mathsf{pm}_2, \ldots, \mathsf{pm}_L\}$. For example, considering the CPU, L could be equal to the different CPU speed levels of the system. For the sake of simplicity, we consider a relationship of orthogonality between m and pm, i.e. data collected for a particular m can be reused $\forall\, \mathsf{pm} \in PM$. Thus, each element of sp_m is stored after an operation of normalization with respect to the current power mode pm_{set} and is reused after an operation of unnormalization with respect to the future power mode pm_{get}.

3.2.2 Architecture

The QoSDB architecture has been designed with a particular emphasis on modularity. Its main components, depicted in Figure 3.1, are detailed below.

QoSDB Core

The core of QoSDB provides the main functionalities of the registry. The *API* constitutes the glue between internal modules and plugins, by coordinating operations and information flows. Moreover, this module allows applications to interact with the QoSDB as detailed in Section 3.3.1.

The *Memorization* module provides the functionalities for storing/recalling data to/from the QoSDB data structure, that contains sampling and statistic data for each application mode. The number of application modes could be potentially high, as applications characterized by various input parameters must collapse them in a unique value. For this reason, the used data structure is a Binary Search Tree (BST), with each node label corresponding to an application mode. From a theoretical point of view, the BST would guarantee a good performance in searching and inserting nodes in the tree, that are the operations mostly performed in the `QoSDB`.

The *Saving* module has the task of saving the statistics contained in the `QoSDB` data structure. These statistics are permanently saved in a database as a consequence of the corresponding API call. This module is also responsible of restoring such statistics in the `QoSDB` data structure when a new instance of the framework is started. The saved data are intended to be managed by the `QoSDB` methods only and thus the use of a relational database has been avoided: it would add overhead and dependencies without adding much benefits. Instead, an ad-hoc database file format has been used.

QoSDB Statistics Plugin

The `QoSDB` Statistics Plugin allows applications to specify the statistics they are interested into. Applications define the maximum number of statistics N_T to be stored and saved for each $m \in M$, and they define the proper computation function for each statistic. The index k used for the insertion in the vector st_m represents the statistic type.

Denoting $\hat{m}$ as the current operating mode, each computation function gets as input the vector of samples $\mathsf{sp}_{\hat{m}}$ and returns the computed statistic $\mathsf{st}_{\hat{m},k}$. Actually the `QoSDB` comes with the average and maximum computation functions already built-in.

QoSDB Clairvoyance Plugin

The `QoSDB` Clairvoyance Plugin permits to predict a QoS statistic related to a given operating mode $\tilde{m}$ for which no statistic has been observed so far. The plugin allows applications to define the chosen algorithm according to the preferred model. Actually, the `QoSDB` comes with a linear regression prediction algorithm based on the Ordinary Least Square (OLS) method. A new algorithm can be inserted by simply redefining the `qosdb_clair_stat_prediction()` function, that can predict the value of a particular statistic whose type is identified by the index k. Such function computes the value $\mathsf{st}_{\tilde{m},k}$ by receiving as input the mode $\tilde{m}$ for which we predict the statistic, the remaining set of modes in the `QoSDB` $X = M \setminus \{\tilde{m}\}$, and the statistics of interest for the given modes $Y = \{\mathsf{st}_{m,k} \, \forall m \in X, k = \mathsf{statistic\ of\ interest}\}$.

```
qosdb_rv qosdb_init (qosdb_context **c, Database name);
qosdb_rv qosdb_cleanup (qosdb_context *c);
qosdb_rv qosdb_save (qosdb_context *c);
qosdb_rv qosdb_merge (qosdb_context *c,
                      const qosdb_app_mode_id app_mode_id);
qosdb_rv qosdb_lookup_app_mode (qosdb_context *c,
                                const qosdb_app_mode app_mode,
                                qosdb_app_mode_id *app_mode_id);
qosdb_rv qosdb_set_sample (qosdb_context *c,
                           const qosdb_app_mode_id app_mode_id,
                           const qosdb_mode_sample y,
                           const qosdb_rspeed s);
qosdb_rv qosdb_get (const qosdb_context *c,
                    const qosdb_app_mode_id app_mode_id,
                    qosdb_mode_stat **y,
                    const qosdb_rspeed s);
```

Listing 3.1: QoSDB API

QoSDB Power Plugin

This plugin allows for setting the preferred model for correlating statistics and samples to different resource speeds. It provides the `qosdb_power_normalize` method, used for store samples whose values are independent of the resource speed at which they have been taken (denoted by the pm_{set} value). It also provides the `qosdb_power_unnormalize` method, used for returning values related to the actual power mode pm_{get}, potentially different from pm_{set}. These two methods are internally called in the `QoSDB` API (see Section 3.3.1), respectively in the `qosdb_set_sample` and `qosdb_get` methods.

3.3 Interface

In this section a description of the available Application Program Interface (API) is given, in order to highlight the framework capabilities. Moreover, the typical structure of an application task using the `QoSDB` library is provided.

3.3.1 API

The `QoSDB` library allows applications to exploit its feature through a well-defined API, as described in Listing 3.1. The definition of these methods follows the common C programming practice of returning an exit status, represented by a `qosdb_rv` type.

The functionality of each method is detailed below.

- `qosdb_init` This method initializes the `QoSDB` library. It is responsible of creating the internal data structure and it will try to load historical statistics from the database, if present; otherwise it creates a new database.

- `qosdb_cleanup` This method cleans-up the internal data structure and resources (e.g. file streams, log handler) associated to the library.

- `qosdb_lookup_app_mode` This method looks-up an application mode identifier. Such identifier will be used for subsequent calling exploiting the `QoSDB` features.

- `qosdb_set_sample` This method sets a sample for a particular mode m. It operates on the `QoSDB` data structure and do not save data in the database. For each observed mode m, samples are saved in a circular buffer whose dimension N_S can be set by the application.

- `qosdb_save` This method saves statistical data contained in the `QoSDB` data structure to the application database file. In particular, it permanently stores st_m for each $m \in M$.

- `qosdb_get` This method gets the vector of statistics st_m for a particular mode m. It operates on the the `QoSDB` data structure and does not get data from the database. If a merge has been performed, the result will keep in count fresh values, otherwise it will return the historical statistics.

- `qosdb_merge` This method merges new statistics with historical ones, for a particular mode m. It operates on the `QoSDB` data structure and it does not get data from the database. The new statistics are computed on-the-fly from fresh samples and are merged with the historical ones (the importance of the fresh information with respect to the historical one can be weighted). The results are automatically set in the `QoSDB` data structure.

3.3.2 Task structure

The structure of a task using the `QoSDB` framework can vary according to the application purposes. As an example, the typical structure of a periodic task interested in collecting data for future statistical analysis is shown in Listing 3.2. It is shown a periodic task that performs a particular job, takes measurements and stores those samples in memory. At the end, the `qosdb_merge` method is called for computing statistics based on the fresh samples and store them. The `qosdb_save` is used for saving such statistics in the database, whilst the `qosdb_cleanup` will clean all the memory associated with the `QoSDB` library. For the sake of simplicity, the `QoSDB` error value is not checked.

It is worth to note that, as data are only collected, the proposed example does not make use of the `qosdb_get` method. A more complex usage of the `QoSDB` can be found in Algorithm 1 described in Section 3.4.2, where the proposed registry is leveraged for performing QoS management in SOAs.

```
void sample_task(){
   qosdb_context* s;
   qosdb_mode_sample sample;
   qosdb_rspeed speed;
   qosdb_app_mode mode1 = f(in1, ..., inJ);
   qosdb_app_mode_id mid1;

   qosdb_init(&s, "myapp.db");
   qosdb_lookup_app_mode(s, mode1, &mid1);
   while (condition) {
      speed = get_current_speed();
      do_job();
      sample = obtain_sample();
      qosdb_set_sample(s, md1, sample, speed);
      wait_for_next_job();
   }
   qosdb_merge(s, mid1);
   qosdb_save(s);
   qosdb_cleanup(s);
}
```

Listing 3.2: Structure of a task using the QoSDB API

3.4 Experimental Results

This section describes some experiments that have been performed with an implementation of the proposed registry on Linux. First, we show the overheads associated with the use of the QoSDB, highlighting their sustainability in a large class of target applications. Subsequently, the functionalities of the QoSDB are shown by reporting its performances in service provisioning and the effectiveness in adaptive predictions of resource requirements.

3.4.1 Overhead Measurements

The QoSDB library has been developed pursuing efficiency, as the overhead introduced should be negligible for a proper integration in QoS architectures.

For this reason, the execution times of the QoSDB API methods has been measured by using a test program that reflects the typical usage. The test has been performed on a $64bit$ GNU/Linux system featured by an Intel CPU running at 1.2 Ghz. The average execution times, as perceived by the application, are reported in Table 3.1 as a function of the application mode number (the t_{10} row is related to a QoSDB featured by 10 application modes, whilst the t_{100} is related to 100 application modes). Each test case has been repeated 50 times and reported results in the first row have the 90% confidence interval always below 6.3%, whilst 90% confidence interval in the second row is always below 5.9%.

It can be seen that the overhead is almost always negligible and in the order of microseconds. Only the qosdb_save method has a significant value, as data have

	init	lookup	set_sample	get	merge	save
t_{10} (ms)	0.259	0.001	0.001	0.001	0.003	2.289
t_{100} (ms)	0.264	0.001	0.001	0.001	0.003	2.336

Table 3.1: Execution overhead of `QoSDB` API

	Code	Data	
		avg	max
m_{10} (KiB)	20	159.4	172
m_{100} (KiB)	20	340.6	448

Table 3.2: Memory overhead of `QoSDB`

to be saved on the hard disk. Other experiments show the same behavior when the `qosdb_init` method loads data from the database (in the reported experiments the database is deleted before each repetition).

As a further overhead measurement, the memory usage of a program using the `QoSDB` library has been measured. We did not use the information that can be gathered by the common *ps* tool, as it reports only a coarse grain information (the total amount of memory allocated for that process). Instead, we made use of the *smaps* interface present in the `procfs` of Linux since the 2.6.14 version. This interface permits to gather information about the actual memory reserved to the process, being also able to distinguish between the private memory and the shared memory, as due to dynamically linked libraries.

Our experiment consists in monitoring for 20 seconds (with a grain of 1 second) the output of the *smaps* interface while our program was running. The only shared libraries used by our program were `libc-2.7.so` and `ld-2.7.so`. The memory reserved for such libraries was always between 572 and 600 KiB.

Table 3.2 reports instead the amount of private memory used by the program as a function of the application modes number (the m_{10} row is related to a `QoSDB` featured by 10 application modes, whilst the m_{100} is related to 100 application modes). In the first column of such table is reported the memory usage for the code section, equal to 20 KiB in the two cases. The second and the third columns correspondingly report the average and the maximum usage related to the data section. This value is variable because it also counts the heap usage, that could vary during the program execution as a consequence, for example, of `malloc` calls. The 90% confidence intervals are 1.4% in the case of 10 application modes and 5.8% in the other case.

In our opinion, such values are acceptable and can suggest the suitability of the proposed registry for a wide range of applications working on Linux systems.

3.4.2 Service Provisioning Scenario

Some experiments have been performed in order to show the effectiveness of `QoSDB` in tuning the resource allocation for providing QoS guarantees in SOAs. The `QoSDB` has been plugged into the real-time QoS architecture described in Chapter 2 and, in particular, it has been tied to the `mod_reserve`, that is capable of providing QoS guarantees for CPU-intensive services (see Section 2.2.2 and 2.4.1) by allocating CPU "shares" to serving tasks. By following the notation introduced in Section 2.3, a resource allocation is specified in terms of a budget Q and a period P, whilst the ratio $B = Q/P$ represents the share of the resource that has been reserved. Focusing on the CPU resource, each task can be thought of as running on a virtual CPU, whose speed is a fraction B of the real CPU speed.

The experiments have been performed on a system with a 1.2 Ghz Intel CPU, $3GiB$ RAM, running a $64bit$ GNU/Linux OS with a 2.6.28 kernel patched with AQuoSA. The Apache 2 web server was configured to provide an image rotation service, like the one described in Section 2.4.1. The client requesting the service is allowed to specify the image width w, the image height h and the desired response time D. For satisfying such QoS requirement, the `mod_reserve` challenge consists in "guessing" the proper pair B and P for scheduling the serving task. In this case, a good choice for the time granularity of the reserve is $P = 100ms$ [1]. The CPU bandwidth B is instead computed by leveraging the capabilities provided by the `QoSDB` library. In this way, it is possible to allocate only the resource share needed for guaranteeing the required QoS, permitting to use resources for completing other tasks or eventually use strategies for energy saving (e.g. switching off a core processor or lowering the resource speed).

The operations performed by `mod_reserve` in providing the real-time adaptive service considered in this scenario can be described by Algorithm 1, which follows the notation introduced in Section 3.2.1 and reports the `QoSDB` functions without any C artifact. In particular, such operations can be detailed as follows:

1. Retrieving the application mode id v_m with the `qosdb_lookup` (lines 1-2). In the service considered in this scenario, each application mode m is represented by the product $w * h$ of the image to rotate.

2. Getting the average statistic for the corresponding mode with the `qosdb_get` (lines 3-5).

3. Computing the value B as the ratio between the average statistic and the service deadline d (lines 6-10). In case the statistic is null, an arbitrary initial bandwidth value B^0 is assigned. The deadline d could be set equal to the desired response time D, however a safer practice consists in setting d equal to a bit lower target value $\hat{D}$, for considering the various source of indeter-

[1] As the period is also representative of the maximum activation delay, lower values could be more appropriate in a different context (for example, see Chapter 4).

Algorithm 1 QOS_PROVISIONING(w, h, d)

1: $m \leftarrow w * h$
2: $v_m \leftarrow \mathsf{qosdb\_lookup\_app\_mode}(m)$
3: $\mathsf{pm}_{get} \leftarrow get\_current\_CPU\_speed()$
4: $\mathsf{st}_m \leftarrow \mathsf{qosdb\_get}(v_m, \mathsf{pm}_{get})$
5: $avg \leftarrow \mathsf{st}_m[AVG]$
6: **if** $avg = NULL$ **then**
7: $\quad B \leftarrow B^0$
8: **else**
9: $\quad B \leftarrow avg/d$
10: **end if**
11: $allocate\_CPU\_share(B, getpid())$
12: $execute\_service(w, h)$
13: $t \leftarrow get\_service\_execution\_time()$
14: $\mathsf{pm}_{set} \leftarrow \mathsf{pm}_{get} * B$
15: $\mathsf{qosdb\_set\_sample}(v_m, t, \mathsf{pm}_{set})$;
16: $\mathsf{qosdb\_merge}(v_m)$;
17: **return**

minism e.g., cache and software interrupts, that can still affect the real-time behavior of AQuoSA in implementing the RR mechanism on top of Linux.

4. Serving the request by assigning a fraction B of CPU to the serving task (lines 11-13). The metafunction *allocate_CPU_share* wraps the AQuoSA calls necessary for performing the resource allocation[2], whilst the *execute_service* wraps operations performed by the web server. We also assume the availability of a system call for getting the service execution time.

5. Storing in memory the service execution time with the `qosdb_set_sample` and updating the statistics with the `qosdb_merge` (lines 14-16).

Please note that such algorithm could be easily applied for the provisioning of a different service (with respect to the one considered in this scenario) by modifying only line 1, that refer to the mapping between service parameters and application modes.

A first experiment, called Experiment I, has been conducted for showing the auto-tuning capability the system acquires by leveraging the proposed QoS registry. Service consumers perform 50 subsequent requests with parameters $w = 1000pixel, h = 1000pixel, D = 660ms$. The desired response time D is not considered as the target deadline of the system, instead we consider, as discussed, a lower internal target $\hat{D} = 640ms$. Denoting $\bar{r}_{\mathsf{full}}$ as the average service response time when requests are processed by using the CPU at full speed, an off-line analysis on such service reveals that $\bar{r}_{\mathsf{full}}$ is equal to $50.18ms$ (the 90% confidence

[2]More information can be found at the URL: `http://aquosa.sourceforge.net/`

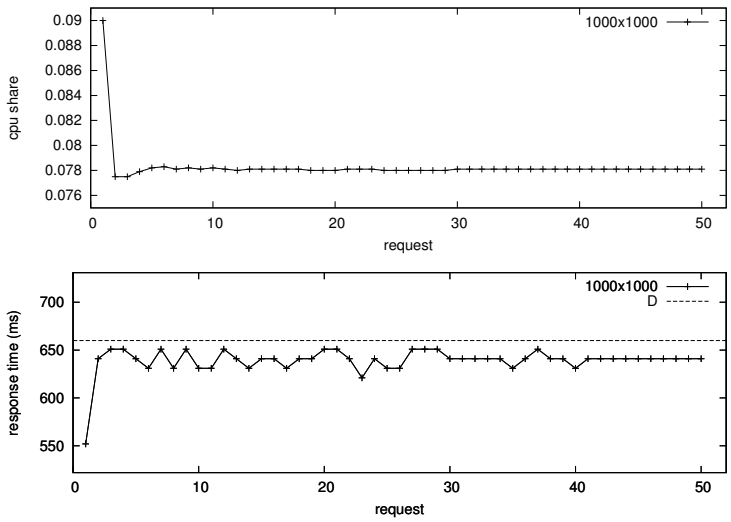

Figure 3.2: Adaptive resource allocation in service provisioning

interval is about 0.1% of the average value). Thus, an optimal assignment B^* in the sense of minimizing the resource allocation for sustaining the desired deadline would be $B^* = \bar{r}_{\mathsf{full}}/\hat{D} \simeq 0.0784$. This measure is used for benchmarking the performance of the system. Instead, for highlighting the auto-tuning capabilities of our architecture, an arbitrary value $B^0 = 0.09$ has been chosen for the initial CPU bandwidth assignment.

At the beginning, requests are performed by 1 service consumer. For each request the assigned bandwidth B and the response time r have been measured and results are plotted in Figure 3.2. In Figure 3.2(a), it can be seen that the first computations of B are clearly overestimated but the system rapidly evolves thanks to the use of `QoSDB`, assigning for most of the requests a bandwidth equal to B^*. Figure 3.2(b) reports the actual response times and shows how the serving task scheduled with the computed bandwidth reservation is capable of guaranteeing the required QoS (values are always the dotted line denoting the desired response time D).

Then, the experiment has been repeated when requests are performed by a different number of concurrent service consumers c and the same behavior of Figure 3.2 has been observed. For doing a comparison of the collected results we introduce two different metrics. By denoting $r_{c,i}$ and $B_{c,i}$ respectively the response time and the bandwidth allocated for the i-th requests performed by client c, such metrics can be defined as follows:

- the Deadline Miss Ratio (DMR) defined as

$$\mathsf{DMR} = \frac{\mathsf{dmn}}{N}$$

where the number of deadline misses dmn is the cardinality of the

	MAPE (%)	DMR (%)
1 client	0.71	0
5 clients	1.76	2.0
10 clients	1.85	5.8

Table 3.3: Performances of resource allocations based on the average QoS statistic

set $\{r_{c,i} | r_{c,i} > D\}$ and N is the total number of requests received by the provider;

- the Mean Absolute Percentage Error (MAPE) defined as

$$\mathsf{MAPE} = \frac{1}{N} \sum_{i,c} \frac{|B^* - B_{c,i}|}{B^*}$$

Table 3.3 reports the value obtained for the introduced metrics when requests are performed by 1, 5 and 10 concurrent clients. The results reported for the MAPE metric are very low and show that the system performs very well in allocating the right resource share. Instead, values for the DMR metric grows proportionally with the number of concurrent clients, reaching a quite significant value in the case of 10 clients. However, the average service response times, calculated throughout the whole experiment, are quite similar in all the three cases (respectively equal to $639.02ms$, $639.22ms$ and $641.79ms$ with the 90% confidence intervals lower than 0.2%) and do not present the same proportional difference with respect to the DMR values. This can suggest that when strong guarantees must be provided in terms of respecting deadlines for real-time service-oriented applications, QoS management techniques based on the analysis of the maximum values could be more appropriate, rather than referring to the average QoS statistics, as done in this experiment.

Following this reasoning, the Experiment II has been performed with the same setup of Experiment I, except for considering the max statistic instead of the average in line 5 of Algorithm 1. In the results, reported in Table 3.4, it could be seen that DMR values has been drastically reduced with respect to values of Table 3.3, meaning that a minor number of deadline misses occur (both in the case of 5 and 10 concurrent clients, only 2 deadline are missed). Of course, this is achieved at the cost of overestimating the resource allocation for service execution, as reflected by MAPE values that are increased with respect to those of Table 3.3.

Finally, an experiment is performed in which the server receives 50 subsequent requests by 1 client but the image resolution changes every 10 requests, forcing a change of the application mode. This experiment has been conceived for highlighting the advantage of using the Prediction Plugin for guessing the behavior of the application when changing from one mode to another. Thus, for each application mode the error in computing the bandwidth B has been measured and plotted

	MAPE (%)	DMR (%)
1 client	1.17	0
5 clients	2.89	0.8
10 clients	8.22	0.4

Table 3.4: Performances of resource allocations based on the max QoS statistic

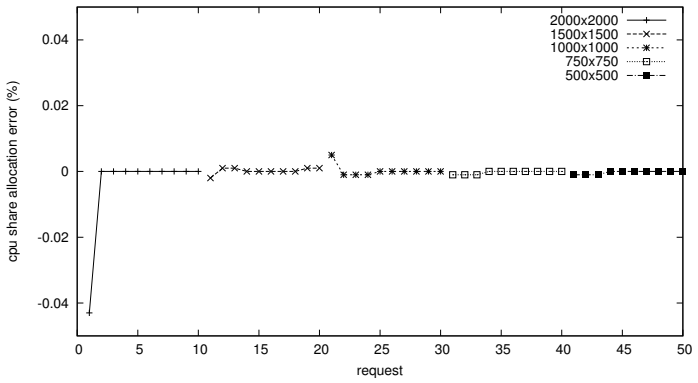

Figure 3.3: QoS Prediction on Application Mode Changes

in Figure 3.3. It can be seen that the allocation error for the mode 2000x2000 is significant, as the database is not populated yet and a $B^0 = 0.15$ allocation value is given. Subsequently, the system evolves towards a plateau, similarly to the first experiment reported in Figure 3.2(a). At request number 11, when the application mode changes to 1500x1500, the database is not populated for that application mode but the `QoSDB` exploits the statistics already collected for performing a guess that is very close to the plateau value. The same behavior can be observed for the other mode changes, that happen at requests number 21, 31 and 41. The prediction algorithm used in this experiment is based on the OLS method and is already built in the `QoSDB`, as described in Section 3.2.2.

In traditional feedback-based scheduling, an application continuously running adapts dynamically the scheduler parameters based on recently observed resource requirements. Instead, in the presented experiment, the individual requests are served by independent activations of the `cgi-bin` service, which can occur concurrently and/or at great or small distances in time. The service is thus instantiated each time along with the creation of the associated resource reservation into the scheduler (as handled by `mod_reserve`). Thus, in the proposed work, the "feedback control loop" is closed in an off-line fashion, by recurring to the registry.

3.5 Summary

In this chapter a QoS registry for SOAs (QoSDB) has been presented. It has been conceived for supporting QoS management in open environments by permitting to predict, store and save QoS statistics related to different functional behaviors of an application. It is characterized by a modular architecture that permits the insertion of new algorithms for an easy customization. The proposed registry has also been developed and we provide overhead measurements gathered on Linux, in order to show viability of the proposed approach.

Moreover, the QoSDB has been integrated into the service-oriented real-time QoS architecture described in Chapter 2 and some experiments have been performed for showing the effectiveness of the proposed solution in supporting adaptive techniques for providing services with QoS guarantees. In fact, by leveraging the proposed registry, the system can be able to self-configuring for a better exploitation of internal resources while guaranteeing the QoS required by users.

Chapter 4

QoS Guarantees for Virtualized Services

Service Oriented Infrastructures (SOIs) are taking advantage of the recent rediscover of resource virtualization [31], whose early works date back to 1967. Virtualization [11] basically refers to the technology that allows a system to host one or more emulated systems, called Virtual Machines (VMs), which may also be seamlessly migrated across physical hosts. In the last few years virtualization technology has been undergoing a steep evolution: the growing industrial interest in enhancing the performance and security of virtualized systems led hardware manufacturers and OS developers to provide more and more support for virtualization, allowing a virtualized machine to exhibit nearly the same performance as the physical one hosting it. Thanks to these advancements in performance, an increasingly appealing opportunity, from a resource provider perspective, is represented by the possibility to provide on-line access to dedicated resources, such as storage, computation and communication resources, in the form of entire VMs, so that there is complete freedom on what OS or additional software to install in order to manage them. Software may run seamlessly inside a VM, completely unaware of the actual hardware on which the VM is running, as well as of what other VMs are being multiplexed by the provider on the same physical resource(s), or even of the fact that is being migrated to a different location. Deployment of VMs instead of physical ones leads to a set of advantages: lower equipment costs, lower deployment and maintenance costs, increased security and fault-tolerance levels.

However, whenever the final distributed virtualized applications are characterized by some kind of timeliness requirements (or exhibit an interactive behavior which requires a high responsiveness), it is of fundamental importance to have control over the temporal behavior of each VM, and to limit the interferences among independent VMs. Even though these applications are likely to possess real-time requirements that are soft in nature (e.g., multimedia streaming or distributed editing of multimedia contents), their violation and the consequent degradation in the QoS experienced by the users may lead to undesirable effects like money losses,

because of the violation of SLAs that may be in place by the interacting parties. Therefore, it is of paramount importance to provide some kind of QoS (or soft real-time) guarantees to applications running in a VM, not only from the perspective of interactive and soft real-time applications, but also from the one of computation-intensive batch activities provided with predetermined SLAs.

Considering such issues, this dissertation gives in this field the following contributions:

- a novel admission control policy for virtualized services is introduced, considering the particular issues (e.g. cost-effectiveness) in resource management of infrastructures with hundreds of services spread across various domains;
- a methodology is presented for assigning system resources to the various VMs according to a proper scheduling algorithm, so that it is possible to evaluate the impact of virtualization on the temporal behavior (thus predictability) of the hosted applications and services;
- experimental results are provided for validating the soundness of the proposed methodology in real-world scenarios.

The remainder of this chapter is organized as follows. Section 4.1 deals with the related work in scheduling VMs and virtualized services hosted within. Section 4.2 describes a novel admission control policy for providing guarantees to virtualized services on a probabilistic basis. Section 4.3 introduces our approach for a soft real-time scheduling of VMs in a GPOS, whilst Section 4.4 presents experimental results for validating such approach. Section 4.5 summarizes the chapter.

4.1 Related Work

The need for real-time support within SOAs is witnessed by the RT-SOA paradigm recently appeared [98, 71], and by the increasing interest in real-time service provisioning within the Grid community [27], just to mention a few. Unfortunately, most of the works in these directions do not consider time-shared nor virtualized nodes. Dinda et al. [31] proposed the use of time-shared systems, but their work did not address the issues concerned with low-level real-time scheduling algorithms. Steps in this direction have been moved by Estévez-Ayres et al. [34], who applied real-time scheduling theory to the problem of providing temporal guarantees to distributed applications built as a network of composable services. However their work addressed the distribution issue, while this work focuses on node-level mechanisms that guarantee correct scheduling of concurrent RT services within the same physical host.

The latter problem has been attacked in some previous work, but the level of determinism needed to run real-time applications inside a VM has not been reached

yet. For example, Xen [11] uses an EDF-based reservation mechanism (called S-EDF) to enforce temporal isolation between the different VMs. However, the S-EDF scheduler lacks a solid theoretical foundation, and is not guaranteed to work correctly in presence of dynamic activations and deactivations. As a result, it seems to have problems in controlling the amount of CPU allocated to the various domains: in a work by Freeman et al. [37], it is shown that the Xen scheduler is not able to properly control CPU allocations for I/O intensive operations. In this work, the Constant Bandwidth Server (CBS) [1] is used as low-level scheduler for VMs. The CBS is an EDF-based scheduler which provides a strong theoretical foundation that has been proved to be able to cope with aperiodic arrivals. The CBS permits to implement the RR framework, in which the resource allocation for each application is specified not only in terms of a share, but also of the desired time granularity. Other approaches can be identified in the Proportional Share [95] and Pfair [12] techniques, aiming to approximate the Generalised Processor Sharing theoretical concept of a fluid allocation, in which each application using the resource marks a progress proportional to a given weight.

Other problems related to VM scheduling have been investigated in PlanetLab [81], a distributed testbed using VMs to increase scalability. PlanetLab tries to address this problem by combining a proportional share scheduler with a mechanism that limits the maximum amount of time available for each VM [13]. However, additional experiments [14] show that the scheduler used in PlanetLab is not able to fully isolate the temporal behaviors of the various VMs, and the authors propose to implement hard reservations.

If virtual machines are scheduled using proper real-time algorithms, the system can be modeled as a hierarchy of schedulers, and its real-time performance can be evaluated by using hierarchical scheduling analysis techniques. For example, Saewong and Rajkumar extended the RR framework to support hierarchical reservations [85]. Shin and Lee proposed a different approach based on a compositional real-time scheduling framework [91], where the timing requirements of complex real-time components are analyzed in isolation and subsumed into an abstract specification called *interface*, then combined to check schedulability of the overall system.

Mok and others [74, 36] presented a general methodology for hierarchical partitioning of a computational resource, where schedulers may be composed at arbitrary nesting levels. Specifically, they associate to each resource partition a *characteristic function* that identifies, for each time window of a given duration, the minimum time that the processor is allocated to the partition. On the other hand, Lipari and Bini [61] addressed the problem of how to optimally tune the scheduling parameters for a partition, in order to fulfill the demand of contained real-time task sets. However, in this dissertation the focus is on providing temporal isolation among VMs for enhancing predictability, rather than on analyzing the schedulability of real-time tasks running in a VM. Interested reader can refer to a former publication [25], in which such problem has been also addressed.

4.2 Probabilistic Guarantee Test

This section introduces a probabilistic admission control test to be run by service providers to decide about the admission of new workflows into the system. The context of this problem is largely inspired by the IRMOS project, where providers have to deal with interactive real-time workflows that terminate very shortly after each activation and rarely saturate underlying resources. The proposed admission test, by leveraging statistical knowledge of actual usage by the users, allows providers to trade resources saturation/overbooking levels for possible penalties that it may have to pay back to customers in case of SLA violations. Also, traditional deterministic admission test is presented, in order to provide strong real-time guarantees.

These admission tests enhance the so-called "advance reservation" mechanisms, that reserve-"in advance"- available resources for a given time span so that the hosted applications may be run with acceptable Quality of Service (QoS) levels. The advance reservation concept, introduced in the context of Grid computing (not to be confused with the "resource reservation" scheduling framework), could be useful leveraged in case of large-scale SOIs.

4.2.1 Model

The considered infrastructure consists of a number of autonomous sites in each of which a set of computational hosts is participating and can be approximated by a star topology architecture. The center of this SOA is occupied by a resource management service, which controls the sharing of the hosts' resources, and is directly connected with them through a wide area network (WAN). As hosts we consider time-shared machines with various processing speeds and soft real-time scheduling capabilities at the OS level. Various implementation of such a capability on the Linux OS exists, even in a way that is transparent to the applications [77, 18].

We consider the following problem whereby the users want to perform executions of a workflow application on the described SOI during a time interval in the future. The users want to have full control of the start time of the executions, i.e. the user is the one that initiates each time the execution of the workflow. Each workflow application consists of one or more services s from a given set of services S, and instances of the same service may reside in multiple hosts and machines within the same host. In order to execute a workflow application, an advanced reservation request, denoted as r, must be made prior to its execution. An advance reservation r has to be specified in terms of start-time and finishing-time for availability of resources. At the time a new request (r_{new}) arrives to the resource management service, each of the machines in the underlying infrastructure may already host a set of advance reservations $R = \{r_1, r_2, \ldots r_n\}$ that were accepted in the past. When r_{new} arrives, the resource management service makes an evaluation of the hosts and presents the user with different ways of running the workflow application on probabilistic guarantees. The cost of the offers fluctuates according to the cal-

culated by the resource management service probabilistic guarantee and the host's business policies. If the user chooses to accept one of the offers, the resource management service assigns the sub-services of the workflow application to the chosen machines within the hosts for future processing. For simplicity and without loss of generality, we assume that the completion time of a service is dominated by the execution time of the service itself, thus a possible delay that may be encountered when communicating with the hosts is considered negligible.

More formally, the problem may be defined as follows:

- Relatively to advance reservations, time is specified in time slices $\{T_k\}_{k\in\mathbb{N}}$ of X time units, following the paradigm of time slice based processors, with $T_{k+1} = T_k + X$. Each time slice represents the smallest temporal unit.

- Each service $s \in S$, activated in the context of r, is associated with a worst-case execution time (WCET) C_{rs}^m that the service may need on a given machine m (the machine index m will be omitted from the notation whenever implicitly identified). In the following, we assume that the WCETs of the services have been estimated by assuming entire execution on a single CPU, among the available ones on a given machine. Also, for the sake of simplicity, we do not address the issue of how and whether to exploit parallelism on the underlying CPU, when dealing with a single service that needs to be run on a physical node (note that parallelism in a workflow is not completely ruled out).

- As already stated, users want to perform multiple runs of the workflow application without fixed execution start times. To accommodate this requirement imposed by the unknown number of executions and arbitrary start time, the resource management service evaluates and reserves computational resources for each service in the workflow across the same time interval, which is equal to the reserved time for the entire workflow TR_r. However, it is reasonable to assume that, within the time period reserved by the user for all services in the workflow, each service is active only for a short time within it.

- Each execution of a workflow application associated to r corresponds to one at most activation of each service involved in the corresponding workflow (having a WCET of C_{rs}). In our model, we are using general service times (as opposed to exponential service times), and each r is associated with a minimum inter- arrival period T_r, corresponding to the minimum time that may elapse between two consecutive requests of activation of the same workflow and consequently between two consecutive activations of the same service in the workflow. The reservation on underlying physical resources should be tuned to sustain at most one execution of the entire workflow every time T_r.

- Each service s needed within each advance reservation is associated with a deadline d_{rs} constituting a timing constraint: the execution of the service instance should only be allowed if the response time (i.e. the time between the activation and the completion of the service execution) is less than or equal to the deadline. More formally, if we denote the activation time of the service by a_{rs}^k (that may not necessarily be equal to the start time due to the scheduling of other activities) and the finishing time by f_{rs}^k, then this constraint may be formalized as:

$$\forall k,\ f_{rs}^k - a_{rs}^k \leq d_{rs} \tag{4.1}$$

  The response-time for the k^{th} activation of a service s, can be defined as $\rho_{rs}^k = f_{rs}^k - a_{rs}^k$. The minimum allowed value for ρ_{rs}^k is C_{rs}, but such a strict value, which is the best achievable one, would imply no possibility for time-sharing the same physical node with other activities.

4.2.2 Methodology

Grouping of overlapping reservations

An advance reservation r will host an application workflow that consists of services and is characterized by a QoS constraint expressed in terms of an end-to-end deadline D_r on the overall response-time ρ_r. We may assume that such a constraint is split into individual relative execution deadlines $\{d_{rs}\}$ for the services composing the workflow. For example, the technique by Yuan et al. [110] may be used for such purpose. From now on, let focus on a single machine m. Let R denote the set of advance reservations already allocated on m. By assuming that the resource management service maintains a registry of such reservations, the time slices T_k on each machine can be grouped under common combinations of overlapping reservations. To formulate this process, we introduce the following notation:

- Let $G = \{g_1, g_2, \ldots g_v\}$ be the set of different groups of overlapping reservations in a given machine during $\mathsf{TR}_{\mathsf{new}}$. Each group g is a set, whose elements are the overlapping reservations. For each g, let rn_g be its cardinality ($\mathsf{rn}_g \equiv |g|$) which indicates the number of included reservations within g.

- Let T_g and tn_g be the set of time slices T_k and their number respectably inside a given group $g \in G$.

To clarify this grouping process, we give the following example: let us consider the case of a candidate machine which has two pre-existing advance reservations r_1 and r_2 with reserved time intervals $T_3 \leq \mathsf{TR}_1 \leq T_5$ and $T_5 \leq \mathsf{TR}_2 \leq T_7$ respectably. We examine the arrival of a request $r_{\mathsf{new}} = r_3$ with reserved time interval $T_2 \leq \mathsf{TR}_3 \leq T_6$. For the requested service we discover candidate hosted machines. As candidate machines at this early stage, we consider all the machines

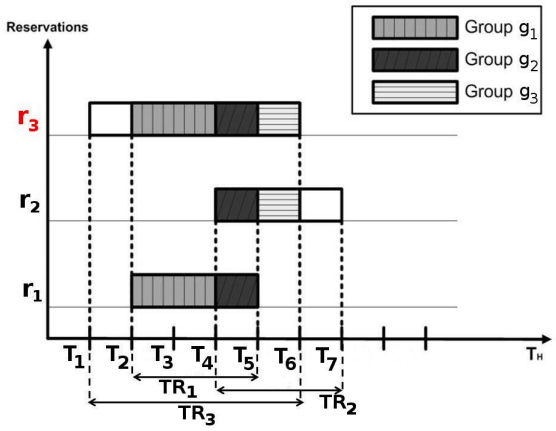

Figure 4.1: Grouping of time slices according to overlapping reservations

m that are hosting the described service on which the worst-case execution time of service C_{rs}^{m} doesn't exceed the deadline d_{rs} defined by the user.

This situation is depicted in Figure 4.1. As illustrated, the introduction of r_3 is translated into a set of three groups $G = \{g_1, g_2, g_3\}$. There is no pre-existing r during time slice T_2. Hence, T_2 is not included in any group since there is no conflict. On the other hand, during time slices T_3 and T_4 there is potential conflict between the new r_3 and the pre-existing r_1. Therefore, both these time slices are grouped under g_1, where $g_1 = \{r_1, r_3\}$, $T_{g_1} = \{T_3, T_4\}$, and so on. At the end of this process all time slices within the requested reserved time are grouped under common pre-existing reservations. This grouping process takes place for all candidate machines in the underlying infrastructure every time a new r arrives and helps to narrow down the complexity of the problem by avoiding identical calculations that take place during the next stages of the algorithm.

Probabilistic model

Any given group $g \in G$ that derives from the described grouping process can be divided into two subgroups: g_c, which contains the advance reservations whose services are executing in a given time slice and $g_{c'}$ containing the rest, i.e. those that are not executing:

$$g = g_c \cup g_{c'} \tag{4.2}$$

Focusing on a single time slice T_k of X units belonging to a group g, let E_{rs} be the event of having the service s (reserved for r) executing in the given time slice T_k. The arbitrariness of the start-time of each workflow instance is modeled as the knowledge of a probability π_{rs} that each service would actually be active in T_k if the underlying physical resource were utilized exclusively by it. As we focus on a single machine m, for the sake of notational brevity, we will omit the dependency from s in sums and products of related terms, but we will maintain the subscript

s in the notation, to stress that we refer to a service s inside r (deployed on m). Therefore, the probability $P(E_{rs})$ of the event $P(E_{rs})$ can be written as:

$$P(E_{rs}) = \begin{cases} \pi_{rs} & \forall T_k \in TR \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

For example, π_{rs} could be computed using queuing theory as the steady-state probabilities of each service having at least one item under processing, under an appropriate model for the stochastic process of the workflow activations. Using Eq. 4.2 and 4.3, the generic probability of having r_{new} grouped under common subgroup executing in any time slice, is given by the following generic equation:

$$P_{g_c} = \prod_{r \in g_c} P(E_{rs}) \prod_{r \in g_{c'}} P(\overline{E_{rs}}) \tag{4.4}$$

where $P(\overline{E_{rs}})$ is the complementary event of E_{rs}. It is worth to note that, in Eq. 4.4, $P(E_{r_{\text{new}}s})$ is included in the first term of equation.

4.2.3 Admission Control

Assuming entire execution on a single CPU among the available ones on a given machine allows for the use of efficient partitioned processor scheduling strategies such as EDF. With this scheduling strategy, it is possible to reach theoretical full saturation of each processor, along with a very simple utilization-based admission control test that simply checks if the sum of the computation requirements for all of the active services on the same processor is less than or equal to 1:

$$\sum_{r \in R} \frac{C_{rs}}{min\{T_r, d_{rs}\}} \leq 1 \tag{4.5}$$

Also, by enriching the scheduling strategies with RR, it is possible to provide the temporal isolation property, i.e., each r may receive scheduling guarantees independently of the behavior of the others. By assigning a budget Q_{rs} and a period P_{rs}, the associated service: (a) is guaranteed the possibility to consume Q_{rs} time units of the resource in every period P_{rs}; (b) is forced not to overcome the consumption of Q_{rs} time units of the resource in each period P_{rs}.

Under these premises, and in order to fulfill a service deadline constraint, it is sufficient to tune the scheduling parameters for a service within a reservation by assigning a budget Q_{rs} equal to the estimated WCET C_{rs}, and minimum period P_{rs} equal to the minimum between T_{rs} and d_{rs}. As P_{rs} represents also the time granularity by which we may control the actual finishing time of each activation, it is useful also to set it to a sub-multiple of such quantity. In any case, the reserved utilization is $U_{rs} = \frac{Q_{rs}}{P_{rs}}$.

Deterministic admission test

The admission control that checks whether a new r may be admitted on the node under consideration is given by

$$\sum_{r \in R} U_{rs} + U_{r_{\mathsf{new}}s} \leq U_{max} \tag{4.6}$$

where the parameter U_{max} has been introduced as the maximum capacity of a single processor, which may not necessarily be equal to 1. In fact, whenever scheduling overheads need to be accounted for, this value is set minor to 1, or the overhead has to be embedded in the WCET values. Actually, as the potentially active advance reservations are not the entire set R but vary dynamically in time depending on the overlapping at any given time slice T_k, the just shown admission test needs to be repeated for each g including r_{new}. Consider the set G_r of groups in G including reservation r, $G_r = \{g \in G | r \in g\}$, the admission test for considering the machine under examination as a candidate becomes:

$$\forall g \in G_{r_{\mathsf{new}}}, \sum_{r \in R} U_{rs} + U_{r_{\mathsf{new}}s} \leq U_{max} \tag{4.7}$$

Finally, it is important to note that, under a scheduling reservation (Q_{rs}, P_{rs}), the maximum time needed by the service to compute once it is started may be approximated as:

$$\rho_{rs} = \left\lceil \frac{C_{rs}}{Q_{rs}} \right\rceil P_{rs} \cong \frac{C_{rs}}{U_{rs}} \tag{4.8}$$

where the index k, relative to each activation of the service, is omitted for simplicity.

Probabilistic admission test

The above shown assignment for resource scheduling parameters is adequate whenever the system is subjected to such a load that, within each r, each service works almost continuously, i.e., an actual usage of the reserved amount of resources for each reservation that gets quite close to saturation. However, in the context of the problem formalized in Section 4.2.2, the services are not going to be continuously active within the time span TR_r due to their dependencies from other services in the workflow, or because the user do not simply use it. In such cases, it could be more convenient for the resource provider to exploit the expected statistical multiplexing among hosted applications (i.e. their activation patterns are supposed to be independent among each other) by offering a probabilistic guarantee at a lower rate, rather than a deterministic one at a higher rate. Of course, this would be translated in the opportunity to host within the same time slice a number of services that theoretically would not be allowed to do so, due to the processor's capacity.

By applying the described real-time scheduling strategies, service components have to share the underlying physical resource, so each service does not use it in an exclusive manner. For this reason, the probability defined in Eq. 4.3 must be properly modified, taking into account the CPU share of the service. In virtue of the impact of the share on the response times of the service (see Eq. 4.8), it is reasonable to assume that the actual probability of having the service component actually active in each time slice be inversely proportional to the resource share:

$$P(E_{rs}) = \begin{cases} \frac{\pi_{rs}}{U_{rs}} & \forall T_k \in TR \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

Now, let us focus on a single time slice T_k, and the group g it belongs to. For each subgroup g_c of g, $g_c \in \mathcal{P}(g)$, it is possible to compute the probability P_{g_c} of having as active at the same exactly all of the r in g_c using Eq. 4.4:

$$\forall g_c \in \mathcal{P}(g),\ P_{g_c} = \prod_{r \in g_c} \frac{\pi_{rs}}{U_{rs}} \prod_{r \in g_{c'}} \left(1 - \frac{\pi_{rs}}{U_{rs}}\right) \tag{4.10}$$

The available utilization by a new r can be considered as a discrete random variable $u(g_c)$. In fact, in the event of over-allocation, to comply with the time constraints of the existing advance reservations, this random variable may take a finite number of possible values, one for each subgroup g_c:

$$u_{g_c} = U_{max} - \sum_{r \in g_c \setminus \{r_{new}\}} U_{rs} \tag{4.11}$$

Therefore, the mean utilization U_o, related to the new r when experimenting resource overloads, is now given by the sum of possible values multiplied with the corresponding probability of the subgroups g_c that exhibit over-allocation. Denoting the set of these subgroups as $g_{over} = \{g_c \in \mathcal{P}(g) | r \in g, \sum_{rs} U_{rs} > U_{max}\}$, U_o can be obtained as

$$U_o = \sum_{g_c \in g_{over}} u_{g_c} P_{g_c}$$

Using the same reasoning and Eq. 4.8 it is also possible to estimate the expected response-time, conditioned to an activation of the workflow in any time slice T_k belonging to the group g:

$$E[\rho_{rs} | T_k \in T_g] = \sum_{g_c \in g_{over}} \frac{C_{rs}}{u_{g_c}} P_{g_c} + \frac{C_{rs}}{U_{r_{new}s}} \left(1 - \sum_{g_c \in g_{over}} P_{g_c}\right) \tag{4.12}$$

It is now possible to estimate the expected value of the overall execution response time for the new reservation during the time span P_{rs}:

$$E[\rho_{rs}] = \frac{1}{P_{rs}} \sum_{g \in G} tn_g E[\rho_{rs} | T_k \in T_g] + \left(P_{rs} - \sum_{g \in G} tn_g\right) \frac{C_{rs}}{U_{r_{new}s}} \tag{4.13}$$

The value obtained for the expected response time could be used by the resource management service to check whether or not a given allocation of the services of the new reservation could be made to certain host machines, giving that its cost is acceptable by the user. For example, if $D_{r_{\text{new}}}$ denotes the average end-to-end response time that the allocation should respect, then the test could be written as:

$$\sum_{s} E\left[\rho_{r_{\text{new}}s}\right] \leq D_{r_{\text{new}}} \tag{4.14}$$

Such a test can be easily incorporated as a QoS parameter inside the Service Level Agreement (SLA) established between the client and the provider prior to the reservation. It should be noted that the test given by Eq. 4.14 allows providers to admit more applications than the one given by Eq. 4.7, as expected [53].

4.3 Virtual Machine Scheduling

The term *virtualization* refers here to the capability, for a computing machine (referred to as the *host*), to emulate the behavior of one or multiple computing machines (the *guests*), in such a way that any software capable of running on the raw hardware may also seamlessly run within the emulated machine (in particular, the applications running on a guest are not able to distinguish whether they are running in a virtual machine or on real hardware).

In a virtualized environment, multiple activities may be hosted on the same physical hardware in different ways. They may run in different VMs that are multiplexed on the same bare hardware (*inter-VM scheduling*), or they may coexist within the same VM where a OS-level scheduler multiplexes them on the same virtualized hardware (*intra-VM scheduling*), and other VMs may possibly be running concurrently on the same physical node. In all cases, appropriate inter-VM and intra-VM scheduling mechanisms are needed to guarantee that the individual activities exhibit the expected Quality of Service (QoS) levels, whenever timeliness requirements are in place.

For developers and designers of time-sensitive software components, virtualization adds a set of new challenging issues that need to be addressed by research.

First, new methodologies are needed to correctly account for the impact of the virtualization overhead on the execution time of real-time tasks, especially in presence of virtualized peripherals, that turn I/O intensive activities into CPU intensive ones (e.g., networking). The capability to migrate VMs on different types of hardware adds complexity to the problem. For example, in the context of SOAs, it is necessary to foresee how a software component would perform if deployed on various physical nodes, in order to choose the optimum deployment that provides the performance promised in the SLA. Common approaches based on direct measurement of the execution time distribution on the target hardware, is not sufficient. A hardware-independent characterization of the execution times, plus a hardware-specific model of the variability of execution times, may be needed.

Second, when the OS is being hosted along with other OSes concurrently running, the time measured in a VM may be discontinuous, or have a strange granularity. Timer devices are emulated by the virtualization engine, and their resolution may be dramatically affected by the inter-VM scheduler and timer virtualization mechanism (affecting the precision and granularity of timers and clocks). The progress rate of a virtualized OS is also not as uniform as expected, due to inter-VM scheduling, and this may have high impact on the response time of virtualized software components.

Finally, multiprocessor and multicore platforms add a new dimension to the problem of real-time virtualized computing. Software components written for high performance parallel machines may not run as expected when executing within a virtualized environment, especially if multiple multicore VMs are running concurrently. For example, spin-lock synchronization primitives (that usually rely on the assumption that the lock owner running on a different processor will release the lock in a short time) may cause problems if the virtualization layer schedules away the VM owning the lock. Suitable mechanisms are needed to mitigate such issues. For example, the VMWare ESX Server[1]embeds mechanisms to address such issues, but more investigations are needed to understand what solutions are most suitable for meeting real-time application requirements.

4.3.1 Problem Presentation

A host is modeled as a set of guest VMs $\{VM^k : k = a, b, \ldots\}$ scheduled by a root (or global) scheduler[2]. Each VM VM^k is modeled as a real-time system composed by a set $\mathcal{T}^k$ of real-time tasks $\mathcal{T}^k = \{\tau_i^k : i = 1, 2, \ldots\}$. Each one of such tasks is a stream of jobs $J_{i,j}^k$, characterized by an arrival time $r_{i,j}^k$, an execution time $c_{i,j}^k$, and an absolute deadline $d_{i,j}^k$ (which is respected if the finishing time $f_{i,j}^k$ of the job is smaller than it).

For the sake of simplicity, the following of this section will only consider periodic real-time tasks[3] $\tau_i^k = (C_i^k, T_i^k)$, with $C_i^k = max_j\{c_{i,j}^k\}$ (Worst Case Execution Time - WCET), and $d_{i,j}^k = r_{i,j+1}^k = r_{i,j}^k + T$.

All the tasks $\tau_i^k \in \mathcal{T}^k$ are scheduled by a local scheduler running in VM^k; hence, the root scheduler selects a VM first, and then the local scheduler selects one of the tasks which are running in such VM. This is a typical example of hierarchical scheduling. A hierarchical scheduling system is denoted from here on by the **X/Y** notation, where **X** denotes the inter-VM scheduling strategy on the host, while **Y** denotes the intra-VM scheduling strategy on the guests (assumed to be the same

[1] See "Co-scheduling SMP VMs in VMware ESX Server, version 3" at http://communities.vmware.com/docs/DOC-4960

[2] The root scheduler may either be implemented in a host OS so to perform inter-VM scheduling (e.g., the KVM approach), or it may be implemented in the virtualization layer (i.e., the Xen approach).

[3] Note that the techniques and results described here can be extended to sporadic real-time tasks, and to tasks with relative deadline different from the period.

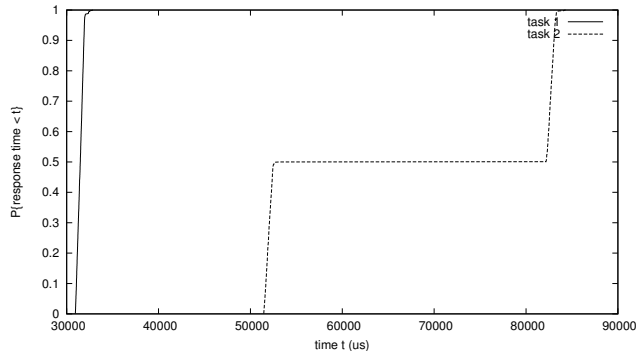

Figure 4.2: CDF of tasks' response times when scheduled on real hardware

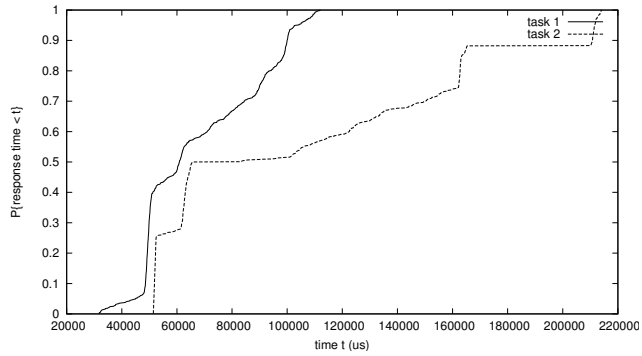

Figure 4.3: CDF of tasks' response times when executed in KVM

on all the guests for the sake of brevity).

As mentioned, an important feature of VMs is that the host behaves as a real PC, and applications running in a VM can be designed as if they ran on real hardware. However, this property does not apply to the temporal behavior inside the virtualized OS. For example, consider the periodic task set $\mathcal{T} = \{\tau_1 = (30ms, 150ms), \tau_2 = (50ms, 200ms)\}$: on a real hardware, real-time scheduling theory guarantees that, if tasks are scheduled with fixed priorities assigned according to RM, then all the deadlines are respected. In fact, this is verified through a simple experiment made by running the task set (with execution times forced as equal as possible to the mentioned WCET values) on a real hardware. Figure 4.2 shows the Cumulative Distribution Function (CDF) $C(x) = P\{\rho_i < x\}$ of the response times $\rho_{i,j} = f_{i,j} - r_{i,j}$ of the two tasks. By looking at the figure, it is possible to see that $C(x)$ arrives to 1 before the deadline of the task, hence all the deadlines are respected.

However, when the same task set $\mathcal{T}$ is run inside a VM (in this example, KVM [51] on Linux has been used, as described in the following), most of the deadlines are easily missed, as shown in Figure 4.3.

Such a behavior occurs every time a general-purpose scheduler is used to sched-

ule concurrent VMs, because of the unpredictability of the temporal interferences that each VM experiences due to the behavior of the other VMs on the same physical host. In the example of Figure 4.3, the VM was scheduled by using the standard completely fair scheduler currently present in the Linux kernel. The problem of correctly scheduling the VMs can be addressed by modeling the system (composed by the host and by the various guests) as a hierarchy of schedulers and by using the hierarchical scheduling analysis that has already been developed in the real-time community. This problem will be addressed in the next section.

Moreover, the overhead introduced by the VM must be proper taken into account. The CPU virtualization and I/O emulation overhead depends on the used virtualization technology and on the implementation of the VM. In this case KVM is used, that is based on the virtualization features provided by modern Intel and AMD CPUs, and emulates a standard PC by allowing to directly execute the guest code on the host in many situations. Hence, the CPU virtualization overhead is expected to be low, but the I/O performance can be an issue, as the guest devices are entirely emulated in software by the VM. The I/O overhead could be reduced by virtualizing the Operating System instead of emulating a full hardware machine, as done by OpenVZ [4]. The overhead of the CPU virtualization mechanism in KVM has been measured by running a simple CPU intensive task that executes a busy loop for $100ms$ and measures the number of cycles that it has been able to perform [5]. The task has been ran 10 times, resulting in an average value of 5390029 cycles on real hardware (with a 90% confidence interval of 2643) and an average value of 5251935 cycles on the virtual machine (with a confidence interval of 3598). These results show that for a CPU-intensive task KVM introduces an overhead of about 2.6%, and the execution times in the virtual machine are quite stable (the confidence interval is 0.069% of the average value). Instead, to understand the impact of the I/O overhead introduced by KVM, the `netperf` program can be used to measure the network performance of the host, and the network performance of a guest running on it. Early results (interested readers can refer to the work by Cucinotta et al. [26] for a deeper analysis) showed that the emulated network card is able to provide a good throughput, at the cost of a high CPU usage: netperf measured a throughput of $96Mbps$ on the host (close to the network capacity, which is $100Mbps$) and a throughput of $89Mbps$ on the guest (close to the value measured on the host). However, the confidence interval was very large (about 28%) and the VM consumed a large amount of CPU time (about 60% of the total CPU time). To work around the I/O overhead introduced by hardware devices emulation, KVM provides a mechanism called *virtio*, which allows to directly exchange data between the host and the guest without emulating a device. When using virtio for the network, the throughput measured by netperf did not change significantly (a value

[4] More information is available at the URL `http://wiki.openvz.org/`

[5] Since the performance provided by KVM looked compatible with the requirements for running real-time virtual machines, other virtualization technologies have not been tested. However, comparison with other virtualization mechanisms such as OS-level virtualization could be performed in a future work.

of $94Mbps$ was measured), but the confidence interval went down to 0.1% (indicating that virtio can make network performance very predictable) and the VM consumed only 1.5% of the CPU.

4.3.2 Approach

As discussed, one of the requirements for fixing the problem showed in Figure 4.3 is a proper scheduling of the VMs in the host, hence the root scheduler has to use a real-time scheduling algorithm.

A first approach to VM scheduling could be to schedule the various VMs with fixed real-time priorities. However, this solution can be problematic, especially when multiple VMs are ran simultaneously on the same host:

1. If a VM consumes more than the expected CPU time, it can stall the whole system (affecting the real-time performance of other unrelated VMs, and even preventing the system administrator from logging into the host)

2. Even if all the tasks respect their WCETs and all the VMs do not consume more than the expected time, this solution can result in a deadline miss.

Problem 2 is due to the fact that when scheduling the VMs with fixed priorities all the tasks running on the highest priority VM will have priority over all the tasks running in the other VMs, independently from their periods.

The alternative approach used in this work for VM scheduling is based on *resource reservations*, which allow to reserve a hardware resource (the CPU, in this case) to a task or application for a time Q in a period P. Although this abstraction can be very effective for serving real-time virtual machines, not all the reservation algorithms can be safely used. For example, to properly serve the real-time applications executing in a VM a reservation mechanism must be designed to correctly cope with aperiodic activations. However, most of the reservation techniques previously used to schedule VMs exhibit the same behavior of a Deferrable Server [96] and are not able to provide temporal isolation to tasks that activate and deactivate dynamically [4].

A resource reservation algorithm is said to be *hard* [83] if it guarantees that a resource will be allocated to a task (or to a set of tasks) for a time Q every reservation period P, and it does not allow to use the resource for more than the reserved amount of time. Our results (presented in Section 4.4) seem to indicate that a scheduler providing hard reservations is more appropriate for scheduling VMs: in fact, most of the hierarchical scheduling analysis for reservation-based systems is based on the assumption that a reservation provides *exactly* Q time units every P time units, and using a hard reservation algorithm is the easiest way to enforce this requirement (this requirement is not satisfied, for example, by some scheduling algorithms that allow to use the reserved time in advance, such as in the first conception of the CBS [1]). As an example, consider a VM VM^a with $\mathcal{T}^a = \{\tau_1^a = (15, 50), \tau_2^a = (75, 300)\}$ served by a reservation $RSV^a = (25, 50)$,

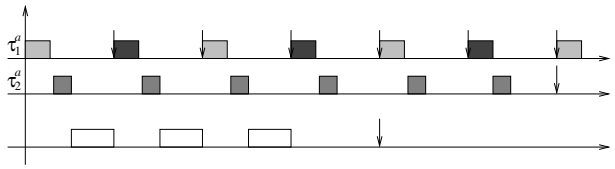

Figure 4.4: Schedule generated by hard reservations

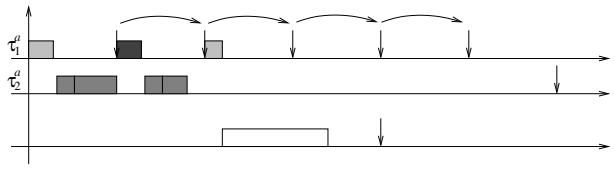

Figure 4.5: Schedule generated by soft reservations

scheduled together with a second reservation $RSV^b = (75, 150)$ which serves a time-consuming task. If hard reservations are used, it is possible to see that both τ_1^a and τ_2^a are able to respect their deadlines as shown in Figure 4.4 (note that this happens because the two tasks start at the same time and periods are harmonic). However, Figure 4.5 shows that if VM^a is scheduled by an algorithm that does not provide a hard reservation behavior (in this case, the CBS algorithm is used), then some tasks (in this case, τ_1^a) can miss a deadline.

Because of the reasons explained above, the solution presented in this dissertation is based on the CBS algorithm, modified to implement a hard reservation behavior. Although different algorithms have been proposed in the context of hierarchical systems, this work presents the correct application of reservation-based scheduling to real implementations of VMs running on real hardware.

Some previous works [69, 5] proposed to add a reclaiming mechanism to the hard reservation behavior, to overcome the throughput problems presented by hard reservations. However, this work is focused on *predictability* more than on performance, hence a reclaiming mechanism is not needed. Moreover, a reclaiming mechanism can make VMs execution less predictable, because the speed of a VM ends up depending on the host workload (the reservation mechanism provides a minimum performance, which is then improved in a non-predictable way by the reclaiming mechanism). If the only goal of the system is to respect the deadlines of tasks running in the various VMs (and not to keep the speed of each VM stable), then hard reservation mechanisms which provide reclaiming can be used.

Once proper real-time scheduling algorithms are used in the guest and in the host, it is possible to apply well-known techniques to analyze the system schedulability [25].

4.4 Experimental Results

This section presents some experimental results that has been performed for evaluating the proposed approach for scheduling VMs in order to support the execution of time-sensitive virtualized services.

Being able to provide accurate estimations of the response times is very important in a SOA environment, which is moving away from the old best-effort Internet model. In fact, a service provider has to take QoS into account, for example in order to meet business policies or because QoS guarantees are required by consumers. Hence, the proposed approach has been evaluated in a typical SOA scenario, with web servers running inside VMs. In particular, the Apache 2 web server has been selected as a representative of a typical SOA workload.

The main goal of such experiments is to show that service response times can be flattened by using the proposed approach, thus reducing uncertainty in response time estimations. In such a way, providers can offer strong QoS guarantees in the provisioning of services regulated by SLAs.

4.4.1 Benchmarks

First, a simple setup has been used. It is composed of a single VM (denoted as guest) running inside a host machine. In this experiment, a set of 10 clients requested a dynamic web page, generated by using a CPU-bound CGI script which rotates an image of 2000x2000 pixels by an angle $\alpha = 20°$. Each one of the 10 clients generates 10 requests, for a total of 100 requests per simulation, and each simulation has been repeated 20 times.

The first column pair of Table 4.1 reports statistics on the response times of the service concerning two cases: (a) the service is provided by the web server running on the host machine; (b) the service is provided by the web server running inside a KVM instance. The table reports the maximum and average response time per request, plus the standard deviation of such a value. The 90% confidence interval is 0.2% of the average value for the web server running on the host, and 1.3% of the average value for the web server running on the guest. It can be seen that the overhead due to virtualization is 7% for average times and 6.4% for maximum times. As this overhead does not affect service response times in a tangible way, it makes sense to exploit all the benefits of virtualization for this type of tasks.

However, the situation changes drastically when the host system is overloaded.

	Host (unloaded)	**Guest** (unloaded)	**Guest** (loaded)	**Guest-rsv** (loaded)
avg	1.14	1.22	11.367	2.044
max	7.91	8.42	89.880	10.832
std.dev	1.26	1.07	15.449	1.275

Table 4.1: Response times (in seconds) for single-VM setup

In fact, the third column of Table 4.1 shows how response times obtained in the guest (the 90% confidence interval is 0.4% of the value) increase when the host is put through a synthetic load that tends to saturate the CPU bandwidth. Note that, with respect to the case of the KVM instance running in an unloaded host, service response times increase by a factor of 10. Moreover, the standard deviation value is quite large, to indicate that fluctuations from average values often occur. This issue is particularly critical in SOA environments, where it could be necessary to provide guarantees in service provisioning: such fluctuations do not allow for precise estimations of service response times, what precludes the possibility for a provider to share the same physical node for multiple VMs that need to exhibit precise QoS levels.

These problems can be addressed by reserving a proper amount of execution time to the KVM instance. The fourth column of Table 4.1 reports service response times obtained by running the web server inside a KVM instance attached to a $(3ms, 5ms)$ hard CBS. In this case, the 90% confidence interval is 0.9% of the value. The results show how the response times scale to values much closer to that of the first column pair of Table 4.1, even when the host is overloaded. This fact, due to the temporal isolation property provided by the CBS, is particularly remarkable because it could allow service providers to offer services with QoS guarantees.

4.4.2 Performance Control

Then, a more complex set-up has been build, with multiple web servers executed in different VMs (e.g., each VM has its own external IP address directly accessible on the same LAN as the host, so that a client cannot distinguish a VM from a physical host). These experiments use two VMs VM^a and VM^b running an Apache 2 web server and the same CGI script of Section 4.4.1 for rotating large images. Two kinds of requests are performed by using the Apache 'ab' program: $req1$ (consisting in the rotation of a 1000x1000 image) and $req2$ (consisting in the rotation of a 2000x2000 image).

To reproduce a realistic scenario, each VM has been put through a different workload, obtained by varying the number of concurrent clients. In particular, VM^a has been tested in serving 10 concurrent clients and VM^b has been tested in serving 20 concurrent clients. Half of the clients of each VM performed 10 requests for the $req1$ service, and the other half performed 10 requests for the $req2$ service. For the sake of brevity, only response times related to $req2$ (the most computational intensive kind of request) has been reported.

The first column pair of Table 4.2 reports statistics on the response times of the service when each VM is executed alone on the host (the 90% confidence interval is 5.4% of the average value). All these values increase in an almost unpredictable way when the two VMs are executed simultaneously on the same host: the service times for VM^a and VM^b are reported respectively in the third and in the fourth column of Table 4.2 (in this case, the 90% confidence intervals are about 6.9%

	$\mathbf{VM^a}$ (alone)	$\mathbf{VM^b}$ (alone)	$\mathbf{VM^a}$ (concurrently)	$\mathbf{VM^b}$ (concurrently)
avg	0.768	1.603	1.564	2.416
max	7.7	14.114	13.253	23.056
std. dev	1.547	2.754	2.434	4.409

Table 4.2: Response times (in seconds) for multiple-VMs setup

	$\mathbf{VM^a}$	$\mathbf{VM^b}$
avg	1.961	3.146
max	10.424	19.508
std.dev	1.967	3.237

Table 4.3: Response times (in seconds) when each VM is served by a CBS

of the average values). This experiment shows that the behavior of each VM is affected by the interference from the other VM: as a result, average and maximum response times increase in a remarkable way. The standard deviations also increase, indicating that fluctuations from average values are large and frequent. As a result, it is not possible to control the response times for the hosted virtualized services.

The problem of interferences between VMs can be avoided by attaching each VM to a hard reservation, in order to provide temporal isolation between different VMs. This has been verified by measuring the response times when VM^a is served by a $(40ms, 100ms)$ hard CBS and VM^b is served by a $(50ms, 100ms)$ hard CBS. The response times collected by using this setup has been reported in Table 4.3 (the 90% confidence intervals are about 1.17% of the average values). Note that average response times are slightly increased respect to the previous example, but maximum response times are reduced. Standard deviation values are also very low, indicating that response times do not deviate too much from average values: as a result, in this case response times can be estimated with a higher degree of accuracy.

This degree of uncertainty is however considerable and can be attributed to the interferences between requests inside each VM. It could be pulled down by even attaching the requests to a reservation. For doing so, the difference in accuracy of timers in the host and in the guest must be taken into consideration, e.g. by reducing reservation period respect to the previous experiment. Consider VM^a served by a $(3ms, 10ms)$ hard CBS and VM^b served by a $(6ms, 10ms)$ hard CBS. Focusing on VM^b, if the Apache instances serving $req1$ and $req2$ are respectively attached to a $(4ms, 100ms)$ reservation and a $(5ms, 100ms)$ reservation, results give a standard deviation value of $0.977s$, which is a very low value. In this case average and maximum response time values of $req2$ are respectively of $3.881s$ and $6.793s$, sustaining that response times are **flattened** with respect to the previous

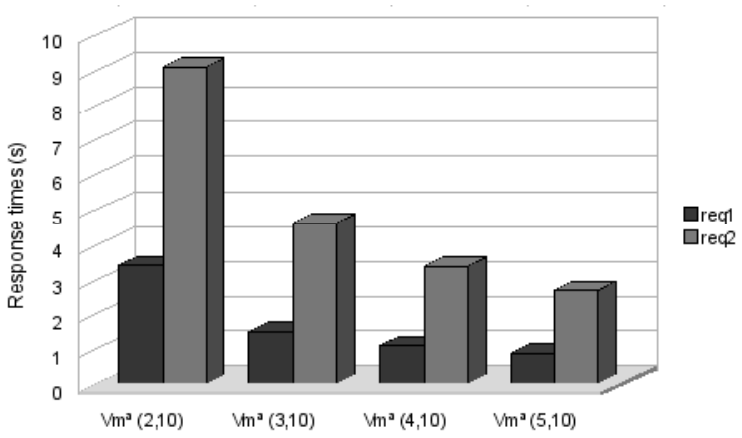

Figure 4.6: Response times varying Q

experiment (see column 2 of Table 4.3).

When using reservations to serve a VM, it is also possible to apply more flexible policies in resource provisioning. For example, it is possible to give more importance to requests towards VM^b by increasing the amount of time reserved to it and decreasing the amount of time reserved to VM^a (for instance, by assigning a reservation $(2ms, 10ms)$ to VM^a and a reservation $(7ms, 10ms)$ to VM^b).

Focusing on a single VM, it could be easily shown that response times can be controlled by modifying the parameters of a reservation. The experiment reported in Figure 4.6 shows how the average response times of VM^a change at varying assignments of the maximum budget Q^a (the reservation period P^a is kept constant).

4.5 Summary

In this chapter the problem of providing soft real-time guarantees for virtualized service components has been faced, with a particular focus on CPU guarantees. In particular, an approach has been presented for scheduling VMs and services hosted within, in a manner that temporal isolation is achieved not only for multiple VMs running on a host but also for virtualized services running inside a VM. Basically, such approach leverages theory on hierarchical real-time scheduling and combines it with the RR framework. Experimental results have been presented for highlighting that such approach is effective in achieving a better predictability for virtualized services concurrently running on heavy-loaded hosts. Moreover, it allows provider to control performance of each VM for fine-tuning service provisioning according to internal policies (e.g. offering better performances to “gold” users with respect to “silver” ones).

Chapter 5

QoS Management for Wireless Sensor Networks

The enterprise production processes strictly interact with the physical environment, and these interactions can have a significant impact on the quality of the final product, both directly, in case of outdoor production (e.g. agriculture, environmental protection, vehicular traffic monitoring), and indirectly, by affecting the correct functioning of the factory plant (e.g. in factory automation, monitoring and control). Nowadays, many low-cost technologies exist that permit the collection of data from the physical world. Among them, Wireless Sensor Networks (WSNs) can be considered the reference technology [112] for the data-gathering level. WSNs are characterized by some distinctive features (like the small size and the scarcity of energy and computational resources) that make them strictly bound to hardware components and/or embedded operating systems. Thus it is difficult to integrate them into enterprise information systems.

One of the main problems is that performance control and the QoS management of the results are obtained by manual ad-hoc programming and configuration. For example, most WSN devices are powered by batteries, and it is therefore important both to minimize their energy consumption, and to monitor and estimate their lifetime. The power consumption depends on the rate at which the data is sampled and sent via radio. Therefore, the final user may want to control and trade off sampling frequency against device lifetime. In addition, for some application it may be important to change the monitored data and area during the system lifetime. However, every time one of the parameters of the monitoring application has to be changed, it is necessary to access the device with its own interface and reprogram it. To simplify integration with higher level software layers, an abstract interface of the WSN is needed, in order to hide the low-level details while maintaining full control over the management of WSN applications.

In the author's opinion, a general solution to this challenging task passes through the adoption of the SOA design methodology for abstracting the data-gathering level, as it permits to build flexible and interoperable systems in which perva-

sive technologies can be integrated in a seamless way for assuring both intra-organizational and inter-organizational cooperation and collaboration.

In this chapter a service-oriented, flexible and adaptable middleware (SensorsMW) is proposed for allowing high-level applications to easily configure the data-gathering level and exploit provided functionality in an effortless manner. In the remainder of the chapter, related work is briefly analyzed in Section 5.1, whilst Section 5.2 describes the architecture of the proposed SOA middleware for WSN. Section 5.3 details a case study that has been built to show the effectiveness of the proposed solution, and Section 5.4 summarizes the chapter.

5.1 Related Work

SensorsMW is a middleware proposed for the ART-DECO [1] Italian research project, to allow an easy and seamless integration of pervasive technologies into the informative system of networked enterprises. For this reason, a service-oriented middleware is a natural choice, as it assures both intra-organizational and inter-organizational interoperability, making available precious information to applications, that can take advantage of them in an effortless manner.

The proposed approach does not aim to implement a service-oriented middleware directly on sensor nodes, forcing SOA-compatible protocol stacks in resource constrained devices [86] [29]. In my opinion, the latter approach has the major drawback to impose too much complexity in devices that are not enough powerful to transmit and elaborate XML messages. These constraints often lead developers to adopt a-priori knowledge in XML message definition, and thus loosing middleware flexibility. Moreover, usage of web services in resource constrained devices imposes a certain energy and latency overhead (as an example, cost for such implementations has been quantified in the work by Priyantha et al. [82]) that could be unacceptable in some cases.

Instead, the proposed middleware allows high-level applications to exploit data-centric network functionalities and configure a WSN according to their needs. It is thus devoted to expose network functionalities as **services**, besides of the low-level technologies used for programming the WSN.

In fact, the logic that allows to abstract the WSN is concentrated on a powerful gateway, to which the sink node is connected. The traditional technique of backup nodes is used to overcome the single-point-of-failure issue, in case it arises. The gateway solution is not new, for example it has been used by Kansal et al. [42] for building a peer-to-peer infrastructure for sharing sensors through the Internet. However, as their work covers a wide range of sensors, it does not explicitly address typical WSN issues. A gateway-based solution has been also proposed by Moeller and Sleman [73], aiming at integrating WSNs into other existing IP-based networks. However, their work is oriented to ambient intelligence at home, so they do not abstract functionalities of the whole network but only of single sensors.

[1] More information is available at the URL `http://artdeco.elet.polimi.it/`

Also, they do not offer a Web Service interface, thus making it difficult to integrate and compose such services.

Moreover, a common criticism of previous mentioned works is that they do not allow applications to reconfigure WSNs according to their needs and are not flexible with respect to existing network protocols. In supporting these features, our approach has some similarities with MiLAN [44], a middleware that allows applications to specify their QoS requirements and configure the network to maximize the application lifetime while providing the required QoS level. However, in MiLAN applications specify their requirements by means of graphs that have to be specialized for each particular sensing scenario. Moreover, data directly flow from each single node to applications, and thus a-posteriori treatments of data cannot be exploited for transparently addressing different application requirements related to same nodes. For these reasons, this approach is less suitable for ensuring integration and interoperability.

We instead allow applications to specify their requirements in a standardized way, by means of SLAs that each application can independently negotiate at runtime, in such a way that an application does not need to know the QoS requirements of other applications. In addition, our architecture allows applications to exploit gathered data by means of Web Services technologies, both for ensuring flexibility in data delivery and guaranteeing integration and interoperability.

It is worth to note that our approach completely differs from that of querying systems like TinyDB [65]. In fact, such systems permits to extract data from a WSN but they do not generally provide high-level interfaces for QoS configuration and management. Moreover, such systems usually exploit low-level techniques for gathering data and can thus be considered as tight extensions of a particular WSN technology. For this reason, they could in turn be used for developing a WSN whose configuration and management are provided by our architecture, that is, as explained in the next section, independent by design of the underlying WSN technology.

5.2 Middleware Description

SensorsMW is characterized, with respect to the state of the art, by the following innovative features.

Service-orientation It allows for a fruitful exploitation of pervasive technologies in enterprise contexts, by abstracting WSNs as a collection of services.

Flexibility It can be used in many contexts or domains, even when specific network critical issues have to be addressed.

Adaptability It can support well-known low-level techniques or legacy deployments that can be already in-place.

The proposed middleware has been designed keeping in mind the main issues of this domain [109, 70], as highlighted by its key features, that can be summarized as follows:

- it supports **QoS specification and management** by using a contract negotiation scheme based on SLAs. As an example, it permits an easy access to network-provided data with different time and space granularity; it supports time and space recognition of network events; it provides both periodic data sampling and event-driven notifications.

- it allows applications to **reconfigure and maintain** the network during its lifetime. As an example, the middleware supports fault detection management by signaling when the number of active devices in a certain area goes below a certain threshold specified in a special contract. Other contracts permits the energy monitoring, by treating it as a special case of generic data monitoring. Moreover, it is possible to implement energy-aware data collection and data fusion in the network itself to spare energy depending on the user requirements.

- it is **independent** of the underlying WSN technology. In fact it does not depend on the network size and topology and porting from one technology to a different one implies the porting of just two sub-components (see the WSNGateway component). It is possible to transparently perform services within the network (e.g. data fusion and filtering) or in the gateway, depending on the services available in the low level WSN technology.

It is also noteworthy to mention that, thanks to its flexibility, the middleware can always embed features that are not explicitly supported by design, as it is completely independent from low-level techniques implemented at the device level.

In this section, the proposed architecture is over-viewed by describing its components in details. Moreover, the specifications of contracts that regulate provided services are presented, in order to illustrate the different capabilities of SensorsMW.

5.2.1 Architecture

The architecture design of SensorsMW follows the guidelines presented in Chapter 2, especially regarding the division in the QoS Negotiation Layer and QoS Provisioning Layer. However, the architecture takes into account and support specific issues of pervasive environments, as can be seen in Figure 5.1.

In fact, it provides the possibility to configure a WSN according to the needs of client applications by leveraging the WS-Agreement framework for an easily creation, management and monitoring of SLAs. The SensorsMW acts as a service provider, as it provides services to client applications, that have to negotiate SLAs

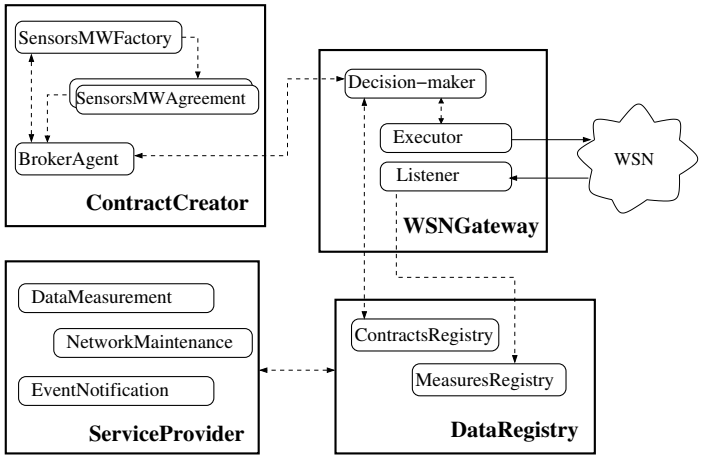

Figure 5.1: SensorsMW architecture

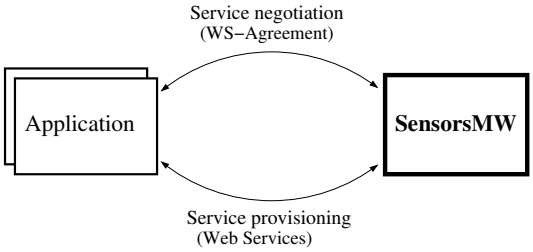

Figure 5.2: SensorsMW interactions with client applications

(also called agreements or contracts in this scope) before consuming services. The interaction scheme with clients is depicted in Figure 5.2.

The SensorsMW architecture is comprised by four main components that are described in the following.

ContractsCreator

This component is responsible for interacting with client applications in all the operations that regard contract creation and management. It comprises the following sub-components.

SensorsMWFactory. It interacts with the client in the agreement creation process and is responsible for publishing the agreement templates related to services provided by the system (see section 5.2.2 for details about template specifications). The templates are fulfilled by clients according to their needs and then evaluated by the system. In case the client proposal can be satisfied, the SensorsMWFactory interacts with the SensorsMWAgreement in creating the agreement.

SensorsMWAgreement. It realizes all the operations related to an agreement, by providing status information for agreements and allowing for an anticipate ending of them. It appears only if an agreement creation process has been successful completed by the SensorsMWFactory component. There will be an instance of SensorsMWAgreement for each contract that is actually in-place.

BrokerAgent. It is responsible for forwarding requests of SensorsMWFactory and SensorsMWAgreement to the lower levels of the architecture. In particular, it forwards:

- admission requests coming from the SensorsMWFactory when a new contract has to be admitted;
- delete requests coming from the SensorsMWAgreement when a contract has to be deleted.

The introduction of this sub-component allows to improve responsiveness of the SensorsMWFactory and SensorsMWAgreement sub-components, that have to interacts with clients, and also allows for decoupling the ContractsCreator and the WSNGateway, that could be deployed in two different physical hosts.

ServiceProvider

This component is responsible for providing services to client applications, in accordance with established contracts. Services are made available by using Web Services technologies, that permits an easily integration and interoperability in enterprise systems. In SensorsMW, three main services have been identified as essentials: they exploit the database provided by the DataRegistry component for providing their functionalities, as described in the following. Please note that contracts related to such services will be described in Section 5.2.2.

DataMeasurement. This service allows client applications to obtain presently gathered data, by presenting the identifier of the previously established contract, that contains all the configuration parameters used by the WSN for gathering measurement data related to a certain physic quantity.

EventNotification. This service allows client applications to receive notifications about events of interest, related to the measurement of a certain physic quantity. Applications can configure events they are interested on, and subscribe to them by means of specific contracts.

NetworkMaintenance. This service allows client applications to perform network maintenance by measuring and monitoring quantities that are necessary for a proper WSN functioning, like the battery level or the number of active sensors in a certain region. Applications can exploit this SensorsMW features by establishing proper contracts.

DataRegistry

This component is responsible for managing all the data that have to be persistently stored for the proper functioning of SensorsMW. It comprises the following subcomponents.

ContractsRegistry. This sub-component maintains the registry of all contracts presently established with client applications. Each contract is represented by an unique identifier plus the featuring parameters, that depend on the type of contract (see Section 5.2.2 for a detailed description of such parameters). The knowledge contained in this registry can be used for admitting new contracts and for providing applications with information regarding established contracts.

MeasuresRegistry. This sub-components maintains the registry of measures gathered by the WSN, in accordance with the presently established contracts. A measure is represented in the registry by the following information:

- the identifier of the measured physical quantity,
- the measure value,
- the datum aggregation type,
- the location in which the datum has been gathered,
- the time at which the datum has been gathered,
- the date in which the datum has been gathered.

In order to provide applications with requested data, the knowledge contained in the MeasuresRegistry is leveraged by the ServiceProvider component, that also contains the logic for binding data with contracts and for correlating data in order to respect established contracts: as an example, the ServiceProvider component may aggregate data *a-posteriori* if such in-network processing feature is not available in the WSN.

WSNGateway

This component is responsible for acting as a gateway respect to the WSN, in the sense that all the communications to and from the WSN pass from this component. It comprises the following sub-components.

Decision-maker. This sub-component appears when a new contract has to be admitted and it decides if a service requested by a client with a certain parameter configuration can be provided by the system. This can comprises both an analysis of existing contracts and of the current status of the WSN. When the component takes decision about an high-level request, it communicates the response to the BrokerAgent, that in turn forwards it to the SensorsMWFactory. If the response is negative, the Decision-maker does not take any further action; if positive, it triggers the creation of a new contract in the DataRegistry and interacts with the Executor for triggering tasks for the WSN, in order to fulfill new requirements of applications.

Executor. This sub-component receives commands from the Decision-maker and translates them into a language understandable from the sensor nodes. This level of indirection allows the independence of the admission control logic from the low-level technology used for the WSN programming, to whom the Executor is strictly bound. It is worth to note that, in order to port SensorsMW to another WSN technology, the Executor and the Listener, described later, are the only sub-components that need to be customizable.

Listener. This sub-component receives data gathered from sensor nodes and store them in the DataRegistry. It can be subdivided in two main modules (not highlighted by figure 5.1): one is responsible for listening data communications from sensors and it is strictly dependent to the low-level WSN technology, the other one is responsible for binding data coming from nodes with respective locations, formatting measures as specified by the MeasuresRegistry and triggering storage.

5.2.2 Contract Specification

SensorsMW allows applications to configure the WSN according to their needs before service provisioning. In particular, for each kind of service, SensorsMW provides an agreement template that has to be fulfilled by applications in order to create agreement proposals. If an agreement proposal is accepted, a contract is established with the client application, and both parties are obliged to honor it.

The agreement templates provided by the SensorsMW layer have been designed by keeping in mind the following principles:

1. they should be well-structured and easily usable by clients;

2. they should be compatible with the limited hardware resources of sensor nodes (e.g. battery power).

For these reasons, templates are specified by using the WS-Agreement [10] framework and are characterized by a certain time span of validity Δt, that is based on an estimation of the remaining lifetime of nodes as a function of the current battery level and the requested sampling time. By assuming an exponential discharge of the battery, the voltage over time obeys to the law

$$V(t) = c(s)e^{\alpha(s)t} \tag{5.1}$$

with $c(s)$ and $\alpha(s)$ being coefficients depending on the sampling period s, whose values can be estimated through experimental measurements [8]. Notice that $\alpha(s) < 0$ since the battery discharges over time. From Eq. (5.1), the validity interval of the contract can be computed as:

$$\Delta t = \frac{1}{\alpha(s)} \log \frac{V_{\text{min}}}{V_0} \tag{5.2}$$

where V_0 is the current measure of the voltage and V_{min} is the minimum operative threshold for the node.

In SensorsMW, we define three different types of services, whose execution parameters can be negotiated by means of templates. Correspondingly, three different kind of contracts have been specified, that can be summarized as follows:

1. *periodic measurement* contract, to periodically measure a certain physical quantity;

2. *event monitoring* contract, to monitor specific events related to quantity measurement;

3. *network management* contract, to control and maintain particular situations related to WSN functioning.

Each kind of contract is characterized by key parameters, that will be described in the following. In particular, as such parameters are negotiated by using templates specified with WS-Agreement, we will concentrate on the `Service Description Term` (SDT) section of each template type, as it is devoted to contain service-related parameters (see also Section 2.2.1 and Figure 2.2). The time span of validity of a contract is instead stored into the `Context` section, as it refers to the contract as a whole.

Periodic measurement contract

The periodic measurement contract allows to periodically measure a certain physical quantity. It requires specifying some parameters that characterize the WSN behavior during service provisioning.

For this kind of contract the following parameters have been defined:

- physical quantity to be measured
- time span for which the measurement has to be done
- sampling period
- type of data aggregation
- region to be measured
- QoS level

SensorsMW allows applications to negotiate such parameters by formally describing them with XML Schema elements, that are inserted in the SDT section of an Agreement. A single SDT can refer to only one physical quantity, as the different quantities that a WSN can measure, can have very different features from one each other. A possible SDT for a template related to a periodic measurement service can be the following.

```
<wsag:ServiceDescriptionTerm wsag:Name="temperature_measurement"
                             wsag:ServiceName="data_measurement">
  <smw:DataMeasurement xmlns:smw="schemas.sensor_mw">
    <smw:Measure>Temperature</smw:Measure>
    <smw:AggregationPeriod>PT1H10M</smw:AggregationPeriod>
    <smw:SamplingTime>PT10S</smw:SamplingTime>
    <smw:Aggregation>avg</smw:Aggregation>
    <smw:Aggregation>max</smw:Aggregation>
    <smw:Region>
      <smw:Location>North Area</smw:Location>
      <smw:Location>Sensor185</smw:Location>
    </smw:Region>
    <smw:QoSLevel>100</smw:QoSLevel>
  </smw:DataMeasurement>
</wsag:ServiceDescriptionTerm>
```

Referring to the proposed example, the meaning of values associated to each element is explained below.

Measure. It expresses the physical quantity to be measured as enumerate. The example specifies the temperature as the quantity of interest.

AggregationPeriod. It expresses the period for data aggregation, by using the *duration* XML data type [66]. In the example, data are aggregated each 1h and 10min. In the simplest case, this value is equal to the `SamplingTime`.

SamplingTime. It expresses the sampling period of sensing by using the *duration* XML data type. In the example, data are sampled by sensors each 10 seconds.

Aggregation. It expresses the aggregation mode of data collected in the same location, by using an enumerate data type (possible values could be `avg`, `max`, `min`).

Region. It expresses the list of locations we are interested to monitor.

Location. It expresses the location of interest by using an unique identifier (it could also be a human-readable name).

QoSLevel. It expresses the QoS level to be provided, by using values belonging to the set $\{x \in N : 0 \leq x \leq 100\}$. A QoS level equal to 100 is equivalent to the maximum quality of service.

It is worth to note that the `QoSLevel` parameter gives an high level of flexibility to SensorsMW. In fact, it has been introduced for conveying other non-functional parameters besides of those explicitly considered in the contract. In this way, the middleware can address different contexts and clients can specify application-dependent QoS requirements (e.g. minimum coverage area, accuracy of measurements). Thus, the mapping between the values assumed by the `QoSLevel` parameter and the provided QoS varies according to the application domain.

Depending on the particular service configuration, some parameters could not be negotiated during the agreement phase: as they are bind to the physic quantity to be measured, a different SDT is used for each quantity. Other parameters are instead negotiable and their default values can be modified by applications when presenting an agreement proposal. In particular, for being adherent to WS-Agreement specification, the `CreationConstraints` template section must contain an `Item` element for each SDT parameter that can be modified.

By using the `Item` element, possible values for variable parameters can also be specified, as highlighted in the following example, in which usable values for the `Aggregation` item are limited to `min`, `max` and `avg`.

```
<wsag:Item wsag:Name="AggregationItem">
  <wsag:Location>
    //wsag:ServiceDescriptionTerm[@Name=
'temperature_measurement']/smw:DataMeasurement/smw:Aggregation
  </wsag:Location>
  <wsag:ItemConstraint>
    <xs:simpleType xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:restriction base="xs:string">
        <xs:enumeration value="min"/>
        <xs:enumeration value="max"/>
        <xs:enumeration value="avg"/>
      </xs:restriction>
    </xs:simpleType>
  </wsag:ItemConstraint>
</wsag:Item>
```

This fragment also highlights as restrictions on values can be specified in an Agreement Template by following the XML Schema model.

Event monitoring contract

The event monitoring contract allows applications to express an interest in certain events, and, in particular, to monitor specific events related to quantity measurement.

An event monitoring contract has some similarities with the periodic measurement contract, as some of its key features are the same. In particular, the following key parameters have been considered:

- the physic quantity of interest
- the event triggering condition
- the event notification delay
- the data aggregation mode
- the region of interest
- the QoS level

The condition that triggers the event has been specified in the formal definition as an interval on values of the physical quantity of interest, in order to express comparative and equality conditions in the same way.

An agreement template provided by SensorsMW for this service can contain many `ServiceDescriptionTerms`, where each SDT is relative to a single quantity and can specify a single event triggering condition. A possible SDT section, containing default values for each element, can be the following one.

```
<wsag:ServiceDescriptionTerm wsag:Name="temperature_monitoring"
                             wsag:ServiceName="event_monitoring">
  <smw:EventMonitoring xmlns:smw="schemas.sensor_mw">
    <smw:Measure>Temperature</smw:Measure>
    <smw:MeasurementInterval>
      <smw:LowerBound>20.0</smw:LowerBound>
      <smw:UpperBound>INF</smw:UpperBound>
    </smw:MeasurementInterval>
    <smw:NotificationDelay>PT15S</smw:NotificationDelay>
    <smw:Aggregation>avg</smw:Aggregation>
    <smw:Region>
      <smw:Location>1</smw:Location>
      <smw:Location>3</smw:Location>
    </smw:Region>
    <smw:QoSLevel>100</smw:QoSLevel>
  </smw:EventMonitoring>
</wsag:ServiceDescriptionTerm>
```

Besides already described elements (see section 5.2.2), specific elements for the event monitoring contract are the following.

MeasurementInterval. It expresses the condition that triggers the event as an interval on values of the physical quantity of interest. The event is generated when the measured value falls into the closed interval. Such interval is specified by the `LowerBound` and `UpperBound` elements, whose values are interpreted according to the International System of Units (for temperature we consider Celsius temperature). In the example, the event is triggered when the temperature reaches a value greater than 20°C (in fact the upper bound of the interval is $+\infty$).

NotificationDelay. It expresses the granted delay from when an event occurs to when the same event is notified. It is specified by using the *duration* XML data type. The example shows a delay of 15 seconds.

Network maintenance contracts

The correct behavior of a WSN can be compromised by many events, like sensor node failures, battery discharges, node displacements or additions. Thus, this kind of contract allows to configure services for controlling and maintaining a WSN, in order to prevent dangerous events or take proper actions in case they happen.

SensorsMW allows to negotiate both measurement services and event-based services on critical quantities for the WSN maintenance, like energy consumption and the number of sensors in a certain region.

The negotiation of network maintenance services is very similar to that described for periodic measurement services and event monitoring services (please refer to Sections 5.2.2 and 5.2.2), as the number of sensors or the battery level can be treated as quantities to be measured in the network.

As an example, it is possible to establish a contract for monitoring the number of sensors on a region and triggering an event when it is behind a certain threshold by using the following excerpt:

```
<wsag:ServiceDescriptionTerm wsag:Name="sensor_number"
                     wsag:ServiceName="network_maintenance">
  <smw:NetworkMonitoring xmlns:smw="schemas.sensor_mw">
    <smw:Measure>SensorNumber</smw:Measure>
    <smw:MeasurementInterval>
      <smw:LowerBound>20.0</smw:LowerBound>
      <smw:UpperBound>INF</smw:UpperBound>
    </smw:MeasurementInterval>
    <smw:NotificationDelay>PT15S</smw:NotificationDelay>
    <smw:Region>
      <smw:Location>1</smw:Location>
      <smw:Location>3</smw:Location>
    </smw:Region>
  </smw:NetworkMonitoring>
</wsag:ServiceDescriptionTerm>
```

In that case the event is triggered when there are less than 20 nodes in the region formed by locations tagged as 1 and 3.

5.3 Case Study

In this section a case study is presented to show the effectiveness of the proposed architecture and demonstrate its adaptability and flexibility for building a middleware for networked enterprises.

As a candidate application, we consider the temperature monitoring in a certain region of a vineyard and we study the behavior of a concrete implementation of SensorsMW, tailored onto TinyOS 2.x.

For the purpose of this case study, we also designed and deployed a WSN testbed with a star topology, in which a node acts as a coordinator and the other ones act as end-devices. This topology has been chosen for the sake of simplicity, as it does not require the use of routing algorithms and can be easily implemented. In any case, other more complex topologies, better suited to particular applications (e.g. the monitoring of a vast vineyard), can be used in conjunction with the proposed middleware, as it does not rely on any low-level technique.

The nodes of the WSN are deployed all around the monitored area and have different tasks according to their category:

- The `Coordinator` node, connected to a resource-unconstrained machine, is responsible for interfacing the WSN with the SensorsMW architecture. In particular, it receives data coming from nodes and forwards them to the WSNGateway. Also, it receives commands from the WSNGateway and forwards them to proper end-devices.

- An `End-Device` node is responsible for gathering data from active sensors and sending them to the Coordinator.

For tailoring SensorsMW onto TinyOS, only the Listener and the Executor sub-component (see Section 5.2) of SensorsMW has been modified, by properly adapting the *Listen* and *Send* TinyOS applications.

In this case study, applications can require DataMeasurement services, that can be configured by filling a periodic measurement contract (see Section 5.2.2) with the desired values. Applications can require QoS-enabled or QoS-disabled services and the application-dependent QoS parameter is considered the reliability of the measure. A QoS-enabled service can be requested by setting the `QoSLevel` parameter equal to 100, whilst a QoS-disabled one can be requested by setting `QoSLevel` equal to 0. When applications require QoS-enabled services, SensorsMW sets the sampling time of nodes in a certain location to the minimum value necessary to gather data at the exact instants of time. Instead, in case of QoS-disabled services, data may be gathered at instants different than required.

Application	Agreement	Sampling (s)	Location
app1	agr1	30	loc1, loc3
app2	agr2	20	loc2
app3	agr3	10	loc1

Table 5.1: Application requested parameters for the SensorsMW case study

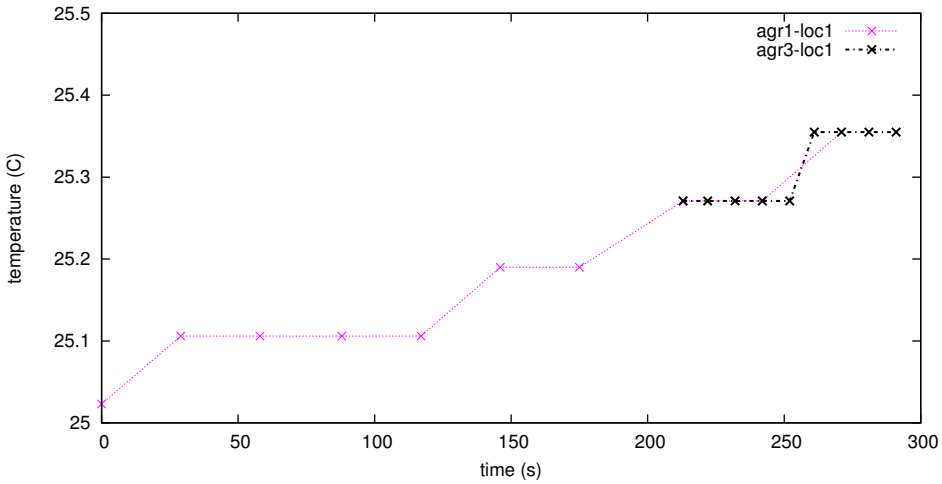

Figure 5.3: Varying of temperature for location `loc1`

As a possible scenario, consider the situation in which no contract has been stipulated and three client applications require a QoS-enabled DataMeasurement service related to the temperature monitoring. Applications configure the service by creating contracts that differ for the parameters described in Table 5.1.

By analyzing these parameters, it can be noticed that each application can specify the desired values independently from the other applications. In fact, applications can specify a different sampling period, even in case they choose the same location for monitoring. This capability is highlighted by Figure 5.3, in which the temperature obtained by *app1* and *app3* for location `loc1` is plotted as a function of time. It can be seen that both applications receive data according to their requirements.

What happens behind the scene is that, when *app3* requires a sampling of $10s$ for location `loc1`, the sampling time of nodes in that location is set to $10s$, in

a manner that both *app1* and *app3* can obtain data gathered at the required time instants. Then, the ServiceProvider component is responsible for analyzing the contract established by each application and for providing data following their requirements.

This scenario is particularly suited for highlighting as SensorsMW can transparent support different requirements of applications, even when consuming services related to the same areas of a WSN.

5.4 Summary

In this chapter, a service-oriented, flexible and adaptable middleware for QoS configuration and management of WSNs has been presented. Thought its architecture design follows the guidelines presented in Chapter 2, it has been extended and tailored to support specific issues of pervasive environments. A case study has been also built and presented to show the effectiveness of the proposed solution.

In particular, the architecture supports QoS specification and management by using a contract negotiation scheme based on SLAs; it allows applications to reconfigure and maintain the network during its lifetime and it is independent of the underlying WSN technology. Moreover, it is characterized by an accurate design that permits to both abstract WSNs for a seamless integration into enterprise information systems and address specific low-level features that must be taken into consideration for guaranteeing certain QoS levels.

Chapter 6

Conclusion

In this dissertation the importance of QoS management in SOA has been stressed, as it allows service providers to offer strong guarantees and to be flexible and adaptable in applying business strategies. In particular, it has been highlighted that respecting QoS guarantees for service provisioning is a critical issue, especially when SLAs are put in action and the violation of such guarantees could cause penalties or money losses. The benefits of embracing SOA have been also described and a panoramic of the SOA application on different environments has been given (see Chapter 1). However, particular issues arise for the QoS management in SOAs when the application environment changes. This dissertation presents a general approach for the problem but also faces specific issues for the next generation industrial automation platforms, for the virtualized environments characterizing the forecoming Cloud computing era and, finally, for the pervasive environments focusing on WSNs.

A general layered architecture has been proposed for negotiating QoS guarantees and providing services respecting such guarantees. An implementation for Linux has been also provided and, in particular, the *mod_reserve* module for Apache 2 has been implemented for providing temporal isolation to services and thus for respecting CPU QoS guarantees. Realistic scenarios tied to the industrial automation world has been built for validating the effectiveness of the proposed approach based on the use of proper soft real-time scheduling techniques. In Chapter 2, experiments are mainly focused on providing low-level QoS parameters (i.e. CPU and network bandwidth shares), whilst in Chapter 3 experiments providing more high-level QoS parameters (i.e. service response time) are presented, together with the QoS registry leveraged in the proposed algorithm for allowing service providers to offer such guarantees.

The problem of providing QoS guarantees has been also faced in the case of virtualized service components (see Chapter 4) and an approach has been presented for scheduling VMs and services hosted within, in a manner that temporal isolation is achieved not only for multiple VMs running on a host but also for virtualized services running inside a VM. The presented experimental results have highlighted

that such approach is effective in achieving a better predictability for virtualized services concurrently running.

Finally, the issues related to pervasive environments have been treated in Chapter 5, where the proposed QoS architecture has been tailored for building a service-oriented middleware that permits to both abstract WSNs for a seamless integration into enterprise information systems and to address specific low-level features necessaries for guaranteeing certain QoS levels. A case study has been also built for showing the effectiveness of the proposed solution.

Considerations on future directions for this work can start by considering that here the use of soft real-time techniques for QoS management has been advocated for providing service guarantees and the Resource Reservation (RR) approach has been adopted for providing temporal isolation in the underlying OS. In particular, a strong focus has been put on offering CPU guarantees for stand-alone services and experimental results related to real-world scenarios have been gathered for proving the effectiveness of such approach. Moreover, some work on network guarantees has been carried on, showing the viability of the RR approach for providing such guarantees. However, only a simple Switched Ethernet (SE) network model has been considered and a future work in this direction could be done by considering a more detailed SE model and/or other network models. Also, the provisioning of disk guarantees could be taken into consideration, especially when providing services that have to load/save a large amount of byte from/to the local hard disk. To this purpose, the Budget Fair Queuing (BFQ) [99] disk scheduler could be effectively leveraged and easily integrated. In fact, it is also based on the concept of bandwidth assignment and, as most of our work, it has been implemented on Linux.

Moreover when a service consumer requests real-time applications composed by several services, more aspects must be considered for the QoS management. A possible approach for providing end-to-end guarantees for such applications comprises the operation of composition, by choosing between different service implementations that may be provided by different nodes. Each implementation is characterized by different real-time requirements and thus applications can be dynamically composed for respecting end-to-end deadlines. The work by Estevéz-Ayres et al. [35] already addressed the problem of time-bounded service composition and may be interesting to apply such algorithm in the proposed QoS architecture.

Instead, an application can be already defined by a graph of dependencies that includes precedences among services and messages exchanged, because it is a user submitted workflow or it has been composed offline by the provider. In such a case, allocating the “right” resource share for the execution of each service becomes crucial for respecting end-to-end deadlines. In this direction, some preliminary work [23] has been conducted by the author for creating an heuristic algorithm that allocates services on nodes and assigns resource shares to each service in a manner that end-to-end deadline constraints are respected. Such heuristics, that promises to be very efficient so as to be usable for on-line allocation decisions,

could be integrated in the proposed QoS architecture for evaluating its effectiveness in the QoS management of real-time service-oriented applications with end-to-end guarantees.

Bibliography

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 4 –13, dec 1998. 16, 63, 75

[2] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli. QoS management through adaptive reservations. *Real-Time Systems*, 29:131–155, 2005. 10.1007/s11241-005-6882-0. 16, 47

[3] Luca Abeni and Giorgio Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, RTCSA '99, pages 70–, Washington, DC, USA, 1999. IEEE Computer Society. 47

[4] Luca Abeni and Giuseppe Lipari. Implementing Resource Reservations in Linux. In *Real-Time Linux Workshop*, 2002. 75

[5] Luca Abeni, Claudio Scordino, Giuseppe Lipari, and Luigi Palopoli. Serving non real-time tasks in a reservation environment. In *Proceedings of the 9th Real-Time Linux Workshop (RTLWS)*, Linz, Austria, November 2007. 76

[6] Werner Almesberger. Linux traffic control - next generation. In *9th International Linux System Technology Conference*, September 2002. http://tcng.sourceforge.net/doc/tcng-overview.pdf. 35

[7] A. Alonso, E. Salazar, and J. López. Resource management for enhancing predictability in systems with limited processing capabilities. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–7, 9 2010. 15, 47

[8] Gaetano F. Anastasi, Enrico Bini, Antonio Romano, and Giuseppe Lipari. A service-oriented architecture for QoS configuration and management of Wireless Sensor Networks. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1 –8, sept. 2010. 19, 89

[9] Gaetano F. Anastasi, Tommaso Cucinotta, Giuseppe Lipari, and Marisol García-Valls. A QoS registry for adaptive real-time service-oriented applications. In *Proceedings of the IEEE International Conference on Service-*

Oriented Computing and Applications, SOCA 2011 (To appear), December 2011. 19

[10] 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. http://www.ogf.org/documents/GFD.107.pdf. 26, 89

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP*, 2003. 61, 63

[12] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 345–354, New York, NY, USA, 1993. ACM. 63

[13] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating system support for planetary-scale network services. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association. 63

[14] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI veritas: Realistic and controlled network experimentation. In *SIGCOMM*, 2006. 63

[15] M. Bichier and K.-J. Lin. Service-oriented computing. *Computer*, 39(3):99 – 101, march 2006. 13

[16] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency (Series in Computer Science)*. Plenum Publishing Co., 2005. 14

[17] Giorgio C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004. 14, 17, 24

[18] Fabio Checconi, Tommaso Cucinotta, Dario Faggioli, and Giuseppe Lipari. Hierarchical multiprocessor cpu reservations for the linux kernel. In *International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, 2009. 64

[19] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI version 3.0.2. OASIS specification, OASIS, October 2004. http://www.uddi.org/pubs/uddi_v3.htm. 47

[20] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina. A real-time service-oriented architecture for industrial automation. *Industrial Informatics, IEEE Transactions on*, 5(3):267–277, Aug. 2009. 19

[21] T. Cucinotta, L. Palopoli, L. Abeni, D. Faggioli, and G. Lipari. On the integration of application level and resource level QoS control for real-time applications. *Industrial Informatics, IEEE Transactions on*, 6(4):479–491, November 2010. 16, 45

[22] T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari, and L. Abeni. Adaptive reservations in a linux environment. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 238 – 245, May 2004. 16

[23] Tommaso Cucinotta and Gaetano F. Anastasi. A heuristic for optimum allocation of real-time service workflows. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2011 (To appear)*, December 2011. 98

[24] Tommaso Cucinotta, Gaetano F. Anastasi, and Luca Abeni. Real-time virtual machines. In *Proceedings of the 29th IEEE Real-Time Systems Symposium, Work In Progress Session*, 2008. 19

[25] Tommaso Cucinotta, Gaetano F. Anastasi, and Luca Abeni. Respecting temporal constraints in virtualised services. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 73–78, July 2009. 19, 63, 76

[26] Tommaso Cucinotta, Dhaval Giani, Dario Faggioli, and Fabio Checconi. Providing performance guarantees to virtual machines using real-time scheduling. In *Proceedings of the 5th Workshop on Virtualization and High-Performance Cloud Computing (VHPC 2010)*, pages 657–664, 2010. 74

[27] Alfredo Cuzzocrea. Towards RT Data Transformation Services over Grids. In *Proceedings of the* 32^{nd} *Annual IEEE Internat. Computer Software and Applic. Conf.*, pages 1143–1149, 2008. 62

[28] I. M Delamer and J. L. Martinez Lastra. Quality of service for CAMX middleware. *International Journal of Computer Integrated Manufacturing*, 19(8):784–804, December 2006. 24

[29] F.C. Delicato, P.F. Pires, L. Pinnez, L. Fernando, and L.F.R. da Costa. A flexible web service based architecture for wireless sensor networks. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 730–735, May 2003. 82

[30] Martin Devera. Htb linux queuing discipline manual - user guide, May 2002. http://luxik.cdi.cz/ devik/qos/htb/userg.pdf. 35

[31] Peter A. Dinda et al. Resource virtualization renaissance. *Computer*, 38(5):28–31, May 2005. 61, 62

[32] J. Domingue, D. Fensel, and R. Gonzalez-Cabero. SOA4All, Enabling the SOA Revolution on a World Wide Scale. pages 530 –537, Aug. 2008. 13

[33] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. 13

[34] I. Estévez-Ayres, L. Almeida, M. García-Valls, and P. Basanta-Val. An architecture to support dynamic service composition in distributed real-time systems. In *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07. 10th IEEE International Symposium on*, pages 249 –256, may 2007. 62

[35] I. Estévez-Ayres, P. Basanta-Val, M. García-Valls, J.A. Fisteus, and L. Almeida. QoS-aware real-time composition algorithms for service-based applications. *Industrial Informatics, IEEE Transactions on*, 5(3):278 –288, aug. 2009. 18, 98

[36] Xiang Feng and Aloysius K. Mok. A model of hierarchical real-time virtual resources. In *Proc. 23rd IEEE Real-Time Systems Symposium*, 2002. 63

[37] T. Freeman, K. Keahey, I. Foster, A. Rana, B. Sotomoayor, and F. Wuerthwein. Division of labor: Tools for growing and scaling grids. In *of Lecture Notes in Computer Science*, pages 40–51. Springer, 2006. 63

[38] M. Garcia-Valls, P. Basanta-Val, and I. Estevez-Ayres. Real-time reconfiguration in multimedia embedded systems. *Consumer Electronics, IEEE Transactions on*, 57(3):1280 –1287, august 2011. 18

[39] Marisol Garcia-Valls, Pablo Basanta-Val, and Iria Estévez-Ayres. Supporting service composition and real-time execution throught characterization of qos properties. In *Proceeding of the 6th international symposium on Software engineering for adaptive and self-managing systems*, SEAMS '11, pages 110–117, New York, NY, USA, 2011. ACM. 18

[40] M. García-Valls, I. Estévez-Ayres, and P. Basanta-Val. Dynamic priority assignment scheme for contract-based resource management. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1987 –1994, 7 2010. 45

[41] Marisol García-Valls, Alejandro Alonso, José Ruiz, and Angel Groba. An architecture of a quality of service resource manager middleware for flexible embedded multimedia systems. In Alberto Coen-Porisini and André van der

Hoek, editors, *Software Engineering and Middleware*, volume 2596 of *Lecture Notes in Computer Science*, pages 36–55. Springer Berlin / Heidelberg, 2003. 15, 47

[42] W.I. Grosky, A. Kansal, S. Nath, Jie Liu, and Feng Zhao. Senseweb: An infrastructure for shared sensing. *Multimedia, IEEE*, 14(4):8 –13, oct.-dec. 2007. 82

[43] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1 – 18, 2005. 14

[44] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, and M.A. Perillo. Middleware to support sensor network applications. *Network, IEEE*, 18(1):6–14, Jan/Feb 2004. 83

[45] E. Hide, T. Stack, J. Regehr, and J. Lepreau. Dynamic cpu management for real-time, middleware-based systems. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 286 – 295, May 2004. 47

[46] Hoai Hoang, M. Jonsson, U. Hagstrom, and A. Kallerdahl. Switched real-time ethernet with earliest deadline first scheduling protocols and traffic handling. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 94 –99, 2002. 25

[47] Bert Hubert, Thomas Graf, Gregory Maxwell, Remco van Mook, Martijn van Oosterhout, Paul B Schroeder, Jasper Spaans, and Pedro Larroy. Linux advanced routing and traffic control howto, March 2004. http://lartc.org/lartc.html. 25

[48] M.N. Huhns and M.P. Singh. Service-oriented computing: key concepts and principles. *Internet Computing, IEEE*, 9(1):75 – 81, Jan-Feb 2005. 13

[49] F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *Industrial Informatics, IEEE Transactions on*, 1(1):62–70, Feb. 2005. 14

[50] François Jammes, Antoine Mensch, and Harm 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*, MPAC '05, pages 1–8, New York, NY, USA, 2005. ACM. 24

[51] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. volume Proc. of the Linux Symposium, Ottawa, Ontario, Canada, 2007. 73

[52] N. Komoda. Service oriented architecture (SOA) in industrial systems. *Industrial Informatics, 2006 IEEE International Conference on*, pages 1–5, Aug. 2006. 24

[53] Kleopatra Konstanteli, Dimosthenis Kyriazis, Theodora Varvarigou, Tommaso Cucinotta, and Gaetano F. Anastasi. Real-time guarantees in flexible advance reservations. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 67–72, July 2009. 19, 71

[54] Yamuna Krishnamurthy, Vishal Kachroo, David A. Karr, Craig Rodrigues, Joseph P. Loyall, Richard E. Schantz, and Douglas C. Schmidt. Integration of QoS-enabled distributed object computing middleware for developing next-generation distributed application. In *LCTES/OM*, pages 230–237, 2001. 17

[55] Yamuna Krishnamurthy, Irfan Pyarali, Christopher D. Gill, Louis Mgeta, Yuanfang Zhang, Stephen Torri, and Douglas 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*, pages 121–129, 2004. 17

[56] J.L.M. Lastra and M. Delamer. Semantic web services in factory automation: fundamental insights and research roadmap. *Industrial Informatics, IEEE Transactions on*, 2(1):1–11, Feb. 2006. 24

[57] Youngkon Lee. Quality-context based "SOA" registry classification for quality of services. In *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, volume 03, pages 2251 –2255, feb. 2009. 47

[58] Jing Li, Yongwang Zhao, Jiawen Ren, and Dianfu Ma. Towards adaptive web services QoS prediction. In *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*, pages 1 –8, dec. 2010. 47

[59] Kwei-Jay Lin and M. Panahi. A real-time service-oriented framework to support sustainable cyber-physical systems. In *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, pages 15 –21, july 2010. 18

[60] Giuseppe Lipari and Sanjoy Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *Proceedings of the 12th Euromicro conference on Real-time systems*, Euromicro-RTS'00, pages 193–200, Washington, DC, USA, 2000. IEEE Computer Society. 16

[61] Giuseppe Lipari and Enrico Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2), 2004. 63

[62] L. Lo Bello, G.A. Kaczynski, and O. Mirabella. Improving the real-time behavior of ethernet networks using traffic smoothing. *Industrial Informatics, IEEE Transactions on*, 1(3):151 – 161, aug. 2005. 25

[63] J. Loeser and H. Haertig. Low-latency hard real-time communication over switched ethernet. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 13 – 22, june 2004. 25

[64] H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck. Web Service Level Agreement (WSLA) Language Specification. http://www.research.ibm.com/wsla, 2003. 27

[65] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005. 83

[66] Ashok Malhotra and Paul V. Biron. XML schema part 2: Datatypes second edition. W3C recommendation, W3C, October 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/. 90

[67] R. Marau, L. Almeida, and P. Pedreiras. Enhancing real-time communication over cots ethernet switches. In *Factory Communication Systems, 2006 IEEE International Workshop on*, pages 295 –302, 2006. 25

[68] R. Marau, L. Almeida, P. Pedreiras, K. Lakshmanan, and R. Rajkumar. Utilization-based schedulability analysis for switched ethernet aiming dynamic qos management. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1 –10, 2010. 25

[69] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. IRIS: a new reclaiming algorithm for server-based real-time systems. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 211 – 218, may 2004. 16, 76

[70] W. Masri and Z. Mammeri. Middleware for wireless sensor networks: A comparative analysis. In *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 349–356, 2007. 84

[71] Carolyn McGregor and J. Mikael Eklund. RT SOAs to Support Remote Critical Care: Trends and Challenges. In *COMPSAC '08: Proc. of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1199–1204, Washington, DC, USA, 2008. 62

[72] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. Processor capacity reserves for multimedia operating systems. Technical Report CMU-CS-93-157, Carnegie Mellon University, Pittsburgh, 1993. 16

[73] R. Moeller and A. Sleman. Wireless networking services for implementation of ambient intelligence at home. In *Devices, Circuits and Systems, 2008. ICCDCS 2008. 7th International Caribbean Conference on*, pages 1 –5, 2008. 82

[74] Aloysius K. Mok and Xiang (Alex) Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, 2001. 63

[75] Hachem Moussa, Tong Gao, I-Ling Yen, Farokh B. Bastani, and Jun-Jang Jeng. Toward effective service composition for real-time SOA-based systems. *Service Oriented Computing and Applications*, 4(1):17–31, 2010. 18

[76] P. Neumann, A. Poeschmann, and R. Messerschmidt. Architectural concept of virtual automation networks. In 17^{th} *IFAC World Congress*, Seoul, Korea, July 2008. 24

[77] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari. AQuoSA-adaptive quality of service architecture. *Softw. Pract. Exper.*, 39:1–31, January 2009. 16, 34, 64

[78] Mark Panahi, Weiran Nie, and Kwei-Jay Lin. RT-Llama: Providing Middleware Support for Real-Time SOA. *IJSSOE*, 1(1):62–78, 2010. 18

[79] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16:389–415, July 2007. 13

[80] Paulo Pedreiras, Ricardo Leite, and Luis Almeida. Characterizing the real-time behavior of prioritized switched-ethernet. In *In 2nd International Workshop on Real-time LANs in the Internet Age (RTLIA)*, 2003. 24

[81] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of the 1st ACM Work on Hot Topics in Networks*, 2002. 63

[82] Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 253–266, New York, NY, USA, 2008. ACM. 82

[83] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, pages 150–164, 1998. 16, 75

[84] WinterGreen Research. Services oriented architecture (soa) market opportunities, strategies, and forecasts, 2006 to 2012. 2006. 13

[85] Saowanee Saewong, Ragunathan (Raj) Rajkumar, John P. Lehoczky, and Mark H. Klein. Analysis of hierar hical fixed-priority scheduling. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pages 173–, Washington, DC, USA, 2002. IEEE Computer Society. 63

[86] Ioakeim K. Samaras, John V. Gialelis, and George D. Hassapis. Integrating wireless sensor networks into enterprise information systems by using web services. In *SENSORCOMM '09: Proceedings of the 2009 Third International Conference on Sensor Technologies and Applications*, pages 580–587, Washington, DC, USA, 2009. IEEE Computer Society. 82

[87] Richard E. Schantz, Joseph P. Loyall, Craig Rodrigues, Douglas C. Schmidt, Yamuna Krishnamurthy, and Irfan 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*, pages 374–393, New York, NY, USA, 2003. Springer-Verlag New York, Inc. 17

[88] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The design of the TAO real-time object request broker. *Computer Communications*, 21:294–324, 1997. 17

[89] C. Schroth and T. Janner. Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services. *IT Professional*, 9(3):36 –41, may-june 2007. 14

[90] Amit Sheth, Jorge Cardoso, John Miller, and Krys Kochut. QoS for service-oriented middleware. In *The 6th World Multiconference on Systemics, Cybernetics and Informatics, Proceedings Vol. 8*, pages 528 – 534, Orlando, FL, 7 2002. 46

[91] Insik Shin and Insup Lee. Compositional real-time scheduling framework. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 57 – 67, dec. 2004. 63

[92] M. Sojka and Z. Hanzalek. Modular architecture for real-time contract-based framework. In *Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on*, pages 66 –69, 7 2009. 47

[93] Michal Sojka, Pavel Píša, Dario Faggioli, Tommaso Cucinotta, Fabio Checconi, Zdenk Hanzálek, and Giuseppe Lipari. Modular software architecture for flexible reservation mechanisms on heterogeneous resources. *Journal of Systems Architecture*, 57:366–382, April 2011. 16

[94] OASIS Standard. Devices profile for web services (dpws) version 1.1, July 2009. 24

[95] I. Stoica, H. Abdel-Wahab, K. Jeffay, S.K. Baruah, J.E. Gehrke, and C.G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Real-Time Systems Symposium, 1996., 17th IEEE*, pages 288 –299, dec 1996. 63

[96] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard-real-time environments. *IEEE Transactions on Computers*, 4(1), January 1995. 75

[97] Ching-Liang 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.*, 33(4):437–451, 2002. 38

[98] W.T. Tsai, Yann-Hang Lee, Zhibin Cao, Yinong Chen, and Bingnan Xiao. RTSOA: Real-Time Service-Oriented Architecture. *Service-Oriented System Engineering, IEEE International Workshop on*, 0:49–56, 2006. 18, 62

[99] P. Valente and F. Checconi. High throughput disk scheduling with fair bandwidth distribution. *Computers, IEEE Transactions on*, 59(9):1172 –1186, sept. 2010. 98

[100] S. Varadarajan and T. Chiueh. Ethereal: a host-transparent real-time fast ethernet switch. In *Network Protocols, 1998. Proceedings. Sixth International Conference on*, pages 12 –21, October 1998. 25

[101] J. Vila-Carbo, J. Tur-Masanet, and E. Hernandez-Orallo. An evaluation of switched ethernet and linux traffic control for real-time transmission. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 400 –407, 2008. 25

[102] J. Vila-Carbo, J. Tur-Masanet, and E. Hernandez-Orallo. Real-time transmission over switched ethernet using a contracts based framework. In *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1 –8, 2009. 25

[103] M.A. Vouk. Cloud computing: Issues, research and implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 31 –40, june 2008. 14

[104] Priscilla Walmsley and David C. Fallside. XML schema part 0: Primer second edition. W3C recommendation, W3C, October 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/. 31

[105] Qixin Wang, S. Gopalakrishnan, Xue Liu, and Lui Sha. A switch design for real-time industrial networks. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08. IEEE*, pages 367 –376, 2008. 25

[106] Victor Fay Wolfe, Lisa Cingiser DiPippo, Roman Ginis, Michael Squadrito, Steven Wohlever, Igor Zykh, and Russell Johnston. Real-time CORBA. In *IEEE Real Time Technology and Applications Symposium*, pages 148–, 1997. 17

[107] I-Ling Yen, Hui Ma, F.B. Bastani, and Hong Mei. QoS-reconfigurable web services and compositions for high-assurance systems. *Computer*, 41(8):48 –55, aug. 2008. 46

[108] Tao Yu and Kwei-Jay Lin. QCWS: an implementation of QoS-capable multimedia web services. *Multimedia Tools Appl.*, 30:165–187, August 2006. 46

[109] Yang Yu, Bhaskar Krishnamachari, and V.K. Prasanna. Issues in designing middleware for wireless sensor networks. *Network, IEEE*, 18(1):15–21, Jan/Feb 2004. 84

[110] Yingchun Yuan, Xiaoping Li, Qian Wang, and Xia Zhu. Cost optimization method for workflows with deadline constraints in grids. In *Computer Supported Cooperative Work in Design, 2007. CSCWD 2007. 11th International Conference on*, pages 783 –788, april 2007. 66

[111] Liangzhao Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311 – 327, may 2004. 46

[112] Lei Zhang and Zhi Wang. Integration of rfid into wireless sensor networks: Architectures, opportunities and challenging problems. *Grid and Cooperative Computing Workshops, International Conference on*, 0:463–469, 2006. 81