11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)

Model Based Development of Quality-Aware Software Services

Miguel A. de Miguel[1], Philippe Massonet[2], Juan P. Silva[1], Javier Briones[1]
Departamento de Ingeniería de Sistemas Telemáticos[1],
ETSI Telecomunicación, Universidad Politécnica de Madrid,
Ciudad Universitaria s/n, E-28040, Madrid, Spain.
Centre d'Excellence en Technologies de l'Information et de la Communication[2],
Rue des Frères Wright, 29/3
Contact email: mmiguel@dit.upm.es

Abstract

Modelling languages and development frameworks give support for functional and structural description of software architectures. But quality-aware applications require languages which allow expressing QoS as a first-class concept during architecture design and service composition, and to extend existing tools and infrastructures adding support for modelling, evaluating, managing and monitoring QoS aspects. In addition to its functional behaviour and internal structure, the developer of each service must consider the fulfilment of its quality requirements. If the service is flexible, the output quality depends both on input quality and available resources (e.g., amounts of CPU execution time and memory).

From the software engineering point of view, modelling of quality-aware requirements and architectures require modelling support for the description of quality concepts, support for the analysis of quality properties (e.g. model checking and consistencies of quality constraints, assembly of quality), tool support for the transition from quality requirements to quality-aware architectures, and from quality-aware architecture to service run-time infrastructures. Quality management in run-time service infrastructures must give support for handling quality concepts dynamically.

QoS-aware modeling frameworks and QoS-aware run-time management infrastructures require a common evolution to get their integration.

1. Introduction: Strategic and Technical Objectives

Frequently the behaviour of a service is functionally correct, but the result it generates is nevertheless unacceptable because the result does not meet some quality criteria, such as the response time and accuracy (i.e., quality). One way to enhance the capability of the system to deliver results of acceptable quality is to use flexible services. A flexible service composition can trade off among different alternative behaviours and resources it uses to produce its results, the quality of its input, and the quality of its result.

Currently, service run-time infrastructures, and software development tools such as modelling languages, transformations of models and generators of code takes into account the functional description of services but not the non-functional properties of models.

Software quality must be handled in the software development phases and the run-time execution of services. Some quality properties of software that requires special attention are the quality-aware static and dynamic composition of services, and the description of quality in requirements, architectures and design models, and in the run-time specification of services.

In the last decade, the increased need for timely and dependable execution and communication support have led to established and improved software quality facilities (e.g. quality negotiation and adaptation algorithms, reservation protocols, resource brokers). They have been integrated in protocol stacks, operating systems kernels, middleware systems and some software development methods. These facilities provide support for the development of multimedia, real-time and complex systems in general. But the quality-aware software development is complex. Software architecture and software development must integrate quality support and analysis. Different levels of software infrastructures and software development require the integration of quality concepts, as described in Figure 1 below.

The scientific and technical research objectives objectives that we are introducing are:

1. *Building modelling languages for Quality*. The construction of languages and tools for supporting quality in modelling languages used for the specification of applications (requirements and architecture),
2. *Developing tools for evaluation of Quality contracts*. Tool support for the evaluation of quality of software architectures and quality contracts and

solutions to improve traceability of quality requirements,

3. *Adapting services infrastructures for the integration of quality management*. The improvement of services infrastructures for the integration of quality management. Specification of applications and services infrastructures will be designed taking into account quality infrastructure available in the other levels.

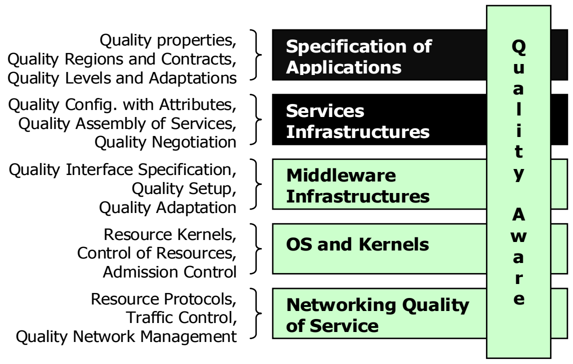
Quality properties,
Quality Regions and Contracts,
Quality Levels and Adaptations
Specification of Applications
Quality Config. with Attributes,
Quality Assembly of Services,
Quality Negotiation
Services Infrastructures
Quality Interface Specification,
Quality Setup,
Quality Adaptation
Middleware Infrastructures
Resource Kernels,
Control of Resources,
Admission Control
OS and Kernels
Resource Protocols,
Traffic Control,
Quality Network Management
Networking Quality of Service
Quality Aware

Figure 1. Levels of Integration of Quality

Building modelling languages for Quality

Currently the quality management or the modelling elements for the description of quality is not available in general modelling languages. The quality management is developed as an activity not directly liked with architectural models (e.g. evaluation of quality of software architectures, and integration of quality in the description software concepts such as use cases and services). Some UML extensions gives support for the QoS description[OMG06], but these extensions do not support the integration of QoS analysis methods and QoS-aware platforms.

Developing tools for evaluation of Quality contracts

Currently the composition of services only takes into account functional description of services. But services are designed to support specific quality properties, not considered during the assembly or composition. The static and dynamic composition of quality-aware services takes into account the quality specification of services and guarantees that quality required in the client and quality offered by the server are compatible. Quality-aware services composition must be handled in modelling tools for software services and service runtime infrastructures. And static quality-aware composition in modelling tools and dynamic composition in run-time infrastructures must be compatible.

Traceability solutions that link functional requirements, use cases, logical services and physical services are available. But the traceability of extra-functional requirements, and the validation of quality properties taken into account in logical and physical architectures requires additional traceability infrastructures.

Adapting services infrastructures for the integration of quality management

Objective quality requirements have associated quantifiable parameters that can be monitored at runtime. The results of monitoring can be used for different purposes. Monitoring of quality parameters gives support for the testing of quality requirements. Some other applications are: i) handling exception of quality contracts, ii) quality adaptation algorithms, iii) improving the specification of quality constraints in software architectures. Quality constraints and quality specifications in general are reusable for the construction of quality monitoring.

2. Progress beyond the state-of-the-art

Service-oriented software development is a new paradigm that utilises services as the basic construct to support the development of distributed applications. It presents the vision of a world of cooperating services where application components are assembled with little effort in a network of services that can be coupled to create flexible dynamic business processes and applications that may span organisations and computing platforms [Pap06].

Figure 2, taken from [Soft06], presents a characterisation of the software-services domain in the past, present, and future. The lower left quadrant illustrates the state of services in the past, the so-called *First Generation Services*, independent non-integrated services with low requirements in relation to service management, quality, reliability, security, trust, and interoperability. The middle part corresponds to the *Second Generation Services*, vertically integrated services where, for instance, a single vendor bundles a series of related services. Second Generation Services represent the current state of software services, embracing issues such as service lifecycle management, quality of service, and service level agreement; an increase in the threshold level of quality, reliability, interoperability, security and trust, and service management in comparison with first generation services.

The top right quadrant presents the *Third Generation Services*, the desired state of software services, concerned with context-determined, consumer-driven, dynamically composed services. There is a dramatic increase in the threshold level of service management, quality, reliability, interoperability, security and trust across the heterogeneous enterprises involved.

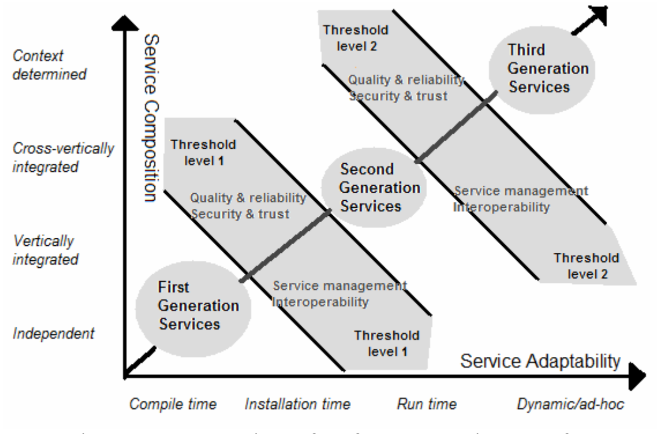

Figure 2. Maturation of Software Services [Soft06]

3. State of the Art and Advances in Software Services and Quality Requirements

A general research objective is to develop the technology for quality management for the Third Generation Services. As presented in [Soft06], there are several challenges in achieving Third Generation Services.

Requirement engineering is the very first step of the system development process. It is concerned with the identification of stakeholders' goals about the intended system; the specification of services and constraints that make operational those goals; and the assignment of responsibilities for the resulting requirements to agents such as humans, devices and software [Lam00]. Requirements engineering is now widely recognized to be among the most critical steps of system development. In order to implement a system that satisfies the stakeholders' needs, those needs must be clearly understood and adequately mapped to specifications of required software behaviour. Inadequate requirements engineering has been repeatedly pointed out to be a major source of problems in software development. The cost of correcting errors or misconceptions in requirements increases exponentially along the software life-cycle if such errors are not handled during the requirements engineering stage. It is thus essential that requirements engineering be done with great care and precision.

In goal-oriented requirements engineering methodology [Lam00], goals play a prominent role in the requirements engineering process. Goals drive the elaboration of requirements to support them; they provide a completeness criterion for the requirements specification - the specification is complete if all stated goals are met by the specification [Yue87]; they provide a rationale for requirements - a requirement exists because of some underlying goal which provides a base for it; they are generally more stable than the requirements to achieve them [Ant94]. In short, requirements "implement" goals much the same way as programs implement design specifications.

Quality requirements are critical to the success of any application. Even if an application fulfils all of its functional requirements by providing all of its required features and implementing each and every one of its use cases, it can still be totally unacceptable if its availability is too little, its capacity is too low, its performance is to slow, it is not interoperable with other systems, it has numerous security vulnerabilities, and it is not considered to be user friendly by its end users. There are a large number of quite different types of quality requirements, and these different types of quality requirements require quite different types of analysis methods. For example, you can use asset-based threat analysis and anti-goals to analyse security requirements as shown in [Naq06], but these techniques will be quite inappropriate for analysing other types of quality requirements such as performance (e.g. throughput, response-time, jitter, and scheduling) requirements or dependability (e.g. availability, reliability, and robustness) requirements.

In practice, quality requirements are often retrofitted late in the development process or pursued in parallel but separately from functional design. These practices tend to result in systems which cannot be accredited, are more costly and less trustworthy. Modern quality requirements methods have advocated for solving this drawback by taking quality requirements into account since early stages in the development process. For instance, [Cys04] treats quality requirements as first class requirements; it presents a process to elicit them, analyse their interdependencies, and trace them to functional conceptual models, focusing on conceptual models expressed using UML. In [Chu04], Chung proposes a framework for representing and integrating quality and functional requirements in the UML use case model. The quality requirements can be implicitly associated with other related use case model elements based on the non-functional requirement propagation rules proposed to eliminate the need for redundant non-functional requirement specifications.

3.1 Advance in Quality Requirements for Software Services

As presented in [Soft06], requirements engineering needs to take into account the fact that not all of the requirements can be defined during design time, and that some of the requirements need to be negotiated at run-time to take into account the adaptable and dynamic nature of service based systems. As also stated in [Soft06], one of the key challenges in terms of quality and reliability is to be able to engineer service based systems that can deal with the change and complexity, yet still remain dependable.

One of the novelties that we propose is to use the specification of quality requirements for defining both design-time and run-time architectural artefacts that meet the quality requirements, as show in Figure 3. Quality requirements can be captured in a requirements model that can then be used to derive: i) design-time artefacts that satisfy the requirements (including the quality requirements) by design such as service specifications or design-time composition of services. ii) Artefacts to be used at run-time to check that run-time decisions do not violate the design-time requirements and additional requirements that are known at run-time. The two examples illustrated in the figure are run-time monitors, and policies for quality requirement negotiation. The run-time monitors monitor required properties of the system that cannot be guaranteed by design because they depend on assumptions made on the environment [Fea98]. The policies for quality requirement negotiation can be used at run-time to drive service level agreement negotiation [Mas05].

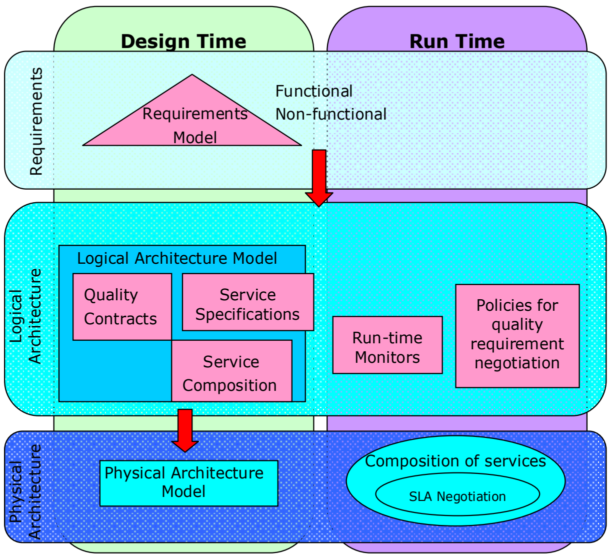

Figure 3. Handling Quality Requirements

Figure 3 also shows that the logical architecture model is refined into a physical architecture model by application of quality design patterns. The requirements are used to guide the application of quality patterns to ensure that the architecture refinement respects the requirements [Lam03].

The ability to elicit quality requirements and understand how to transform them into appropriate architectural solutions is one of the challenges of Third Generation Services [Soft06, page 8]. As far as we know, this is the first type of work to be developed in this area for the case of software services.

4. State of the Art and Advances in Quality-Aware Models

The design by contract is a subject well studied in the field of object-oriented and component oriented software development, but these approaches do not take into account extra-functional properties of components and services.

Innovation advances produce quality models based on quality contracts. The quality contracts of services must take into account two fundamental concepts: i) Definition of quality characteristics that represent a quantifiable aspect of quality, which are defined independently of the means by which it is represented or controlled. Levels of abstraction depend on the domain, applications and precision of requirements. ii) Quality constraints define any kind of restriction that services impose on quality characteristics.

The services have two view points: server and client, and the quality constraints are definable from the client or server view point. If the constraints that impose the client and the server are compatible, the contract is allowed. The restrictions express limitations in the parameters and methods of characteristics. They identify ranges of values allowed for one or multiple parameters and methods and their dependencies.

Examples of simple quality constraints are constraints that describe maximum response times, or the minimum number of errors supported. Sometimes the quality characteristics have associated interdependencies, for instance, in a compression algorithm; the response time depends on the compression degree (more level of compressions requires more computation time) or the functions for the description of subjective priority of qualities or for the description of quality optimal values.

Quality constraint can represent the dependencies and the allowed values taking into account the dependencies. Figure 4 represents the dependencies of qualities qx, qy and qz for a hypothetical implementation function - the maximum and minimum values and the dependencies of quality values; qx cannot have an arbitrary value when the values of qy and qz are fixed. Analytical methods are based on the optimisation of these functions and these functions can be restricted for specific analysis methods. The analytical methods must guarantee that clients and servers have common spaces in their allowed spaces and define the quality contract based on the allowed space.

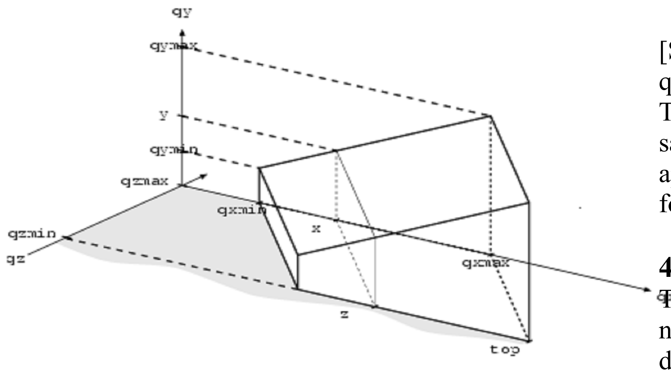

Figure 4. Constraints and relationships of qualities

Examples of quality-enabled modelling languages are QML (QoS Modelling Language) [Fro98] and CQML (Component Quality Modelling Language) [Aaf02]. QML and CQML are languages with a BNF grammar. Other similar approaches are based on metamodels [Ase00] [Bor00]. These languages provide support for the description of user defined quality categories and characteristics, quality contracts and quality bindings. They are frameworks for the description of quality catalogs [Bra02] of general quality parameters, or application specific quality parameters. They do not provide support to optimise the resource allocation, or evaluate the levels of quality provided. They address the problem from the specification point of view. Two OMG standards designed for handling quality specifications in UML models are [OMG06] [OMG02].

Another approach is the description of resource services quality. [Raj97][Sel00] provide support for describing quality based on resource services, and the relation with analytic methods of performances such as latencies and throughputs.

Quality-enabled modelling languages pay special attention to the specification of quality characteristics and parameters, quality contracts for the description of restrictions or quality values, and binding of quality between components, resources and subsystems.

Analytical models for quality management provide support for the application of metric evaluations and resource allocation optimisation. [Raj97] proposes a general quality analytical model for the optimisation of resource allocation. The model assumes a system with multiple resources and dynamic applications, each of which can operate at different levels of quality, based on the system resources available to it. Reward functions describe the interdependencies of quality levels and resource allocation, utility functions and weighted utility functions evaluate the application and system quality. The optimisation of these functions provides the optimal resource distribution.

Other approaches are domain specific. [Sta95][Ven97] are analytical models to support the quality metrics of video and multimedia applications. They identify the quality parameters for user satisfaction and resource consumption in these types of applications (video and multimedia), and the functions for the relationships of resources and user satisfaction.

4.1 Advance in Quality-Aware Models

Three different points of view of quality specification not combined in a common solution yet are: i) the description of user perceptible quality characteristics and their contracts; ii) the definition of system and resource levels quality characteristics and contracts; and iii) the analytic methods that provide support for the optimal resource allocation and quality levels identifications.

New progresses innovate in proposing a development process that integrates above points of view. We have identified four basic modelling activities that must be integrated into general development processes:

- *Quality Requirements*. Quality requirements will be integrated in use cases models and goal models to represent extra-functional requirements.
- *Quality Model*. A quality-aware project includes specific quality characteristics, or can reuse some general characteristics. The quality model defines the set of characteristics that define the quality language for the project. Quality characteristics model the quantitative and qualitative dimensions for the specification of extra-functional requirements.
- *Quality Interfaces of Quality Software Elements*. UML interface specification includes the functional properties of some software elements (e.g. classes and components). Quality interface extends this specification to define quality dependencies of software element, from user and implementation perspectives.
- *Quality Behaviour*. The quality behaviour models the specific behaviour of software elements that support the qualities provided.

5. State of the Art and Advances in Quality-Based Service Composition

Service composition is concerned with the aggregation of multiple services into a single composite service. Resulting composite services may be used by service as basic services in further service compositions or may be offered as complete applications/solutions to service clients. Service aggregators thus become service providers by publishing the service descriptions of the composite service they create. Service aggregators develop specifications and/or code that permit the composite service to perform functions that are based on features such as meta-data descriptions, standard terminology and reference models and service conformance.

The Web, as the dominating Internet service, has evolved into the most popular and widespread platform for world wide global information systems. Because of its diffusion, the Web has been widely used as a platform for dynamic, distributed applications. Different areas (i.e., e-science, e-business, e-learning, etc.) are adopting the Web protocols as a basis for their service oriented middleware infrastructures. The Web Service Architecture (WSA) and the Open Grid Service Architecture (OGSA) are both Web based instances of SOA related, respectively, to the e-business and e-science scenario. In the Web, the terms “orchestration” and “choreography” have been used to describe business interaction protocols comprising collaborative services. While orchestration characterizes executable business processes that embody the perspective of a particular party, choreography describes the interaction pattern among different distributed business processes.

Some of the research effort in service composition has been concentrated on solving the problem of dynamic composition [Yan04], modularising compositions [Cha04] and formal verification of service composition [Sol04]. There are also some efforts in applying AI techniques to automate the retrieval and composition of Web services [Pis05].

However, many of the approaches to service composition neglect the context in which the composition takes place. This is still a challenge that needs to be overcome if we want to have full adoption of service-oriented technology. The service composition needs to understand and respect participants’ policies, performance levels, security requirements, SLA stipulations, and so forth. Initial efforts to deal with quality in service composition include [Zen03, Mac02, Mes02]. In [Zen03], it is proposed a global planning approach to optimally select component services during the execution of a composite Web service considering quality criteria, and global constraints and preferences set by the user. The concept of Quality of Business is introduced in [Mac02], where it is introduced a methodology to direct service composition taking into account quality by using ideas from utility computing. [Mes02] develops a probability model for composing Web services taking into consideration quality of services attributes.

5.1 Advance in Quality-Based Service Composition

Quality-based service composition has been identified as one of the research challenges of the Service-Oriented Computing Research Roadmap [Pap06]. New issues tackle this challenge by developing methods and tools for design-time and run-time composition of services. In contrast to previous work, especially important are dynamic services where quality attributes can vary throughout the execution of a system.

6. State of the Art and Advances in Run-Time Quality Management for Services

During the last decade quality management has been integrated in different types of middleware infrastructures. The integration of QoS facilities into middleware systems was identified as a challenge for the middleware infrastructures [Gei01][Sch00], and during the last years it has been integrated in component and object oriented middleware. But the integration in service run-time infrastructures (e.g. SOA) does not have well established solutions for the quality management. In some solutions, QoS middleware cooperates with existing solutions at Operating System and network levels, and proposes the middleware layer to support other facilities. Some proposals study the integration of QoS facilities in component models such as CCM and EJB.

6.1 Advance in Run-Time Quality Management for Services

New innovative solutions for some research challenges in quality management in service run-time infrastructures are:

- *QoS dynamic composition of services*. Quality-aware service composition is a basic objective and it will propose solutions for the dynamic composition integrated in run-time infrastructures.
- *QoS run-time negotiation of QoS, and service level agreements*. Quality contracts can change dynamically, and the negotiation of service level agreements at run-time provides support for this dynamic reconfiguration.
- *Integration of QoS management for services and network level quality support*. Basic networking and operating systems infrastructures include solutions for handling quality.
- *Compatibility of quality modelling concepts and run-time concepts*. Quality modelling elements and quality management concepts must be compatible and coordinated based on model driven solutions.

7. Final Remarks

The integration of QoS modelling frameworks and Services management infrastructures requires the assimilation of service management concepts into QoS modelling elements. The analysis methods used during design of modelling phases (e.g. analysis of QoS composition) must be compatible with run-time processes (e.g. QoS negotiation).

QoS composition in modeling frameworks and service management require specific advances that verify the quality compatibility of components. The model of QoS composition must be the same model for compositions described in model and the dynamic composition supported at run-time.

8. References

[Aaf02] J. Aagedal and E. Ecklund, "Modeling QoS: Toward a UML Profile", Proc. UML-2002 Conference, Springer Verlag (2002).

[Ant94] A.I. Anton, W.M. McCracken, C. Potts. Goal Decomposition and Scenario Analysis in Business Process Engineering. CAiSE'94, LNCS 811, Springer-Verlag.

[Ase00] J. Asensio and V. Villagrá, "A UML Profile for QoS Management Information Specification in Distributed Object-based Applications", Proc. 7th Workshop HP Open View University Association (2000).

[Bor00] M. Born, A. Halteren and O. Kath, "Modeling and Runtime Support for Quality of Service in Distributed Component Platforms", Proc. 11th Annual IFIP/IEEE Workshop on Distributed Systems: Operations and Management, (December 2000).

[Bra02] G. Brahnmath, R. Raje, A. Olson, M. Auguston, B. Bryant and C. B¡urt, "A Quality of Service Catalog for Software Components", Proc. Southeastern Software Engineering Conference 2002, (April 2002).

[Cys04] L.M. Cysneiros, J.C.S. do Prado Leite, J C S Nonfunctional Requirements: From Elicitation to Conceptual Models. IEEE Transactions on Software Engineering. Vol. 30, no. 5, pp. 328-350. May 2004

[Chu04] L. Chung, S. Supakkul: Representing NFRs and FRs: A Goal-Oriented and Use Case Driven Approach. Software Engineering Research, Management and Applications, Lecture Notes in Computer Science, 3647, Springer, 2005.

[Fea98] M.S. Feather, S. Fickas, A. van Lamsweerde, C. Ponsard, Reconciling System Requirements and Runtime Behaviour Proceedings de IWSSD'98 - 9th International Workshop on Software Specification and Design, IEEE, Isobe, Japan, April 1998.

[Fro98] S. Frolund and J. Koistinen, "Quality of Service Specification in Distributed Object Systems", Distributed Systems Engineering Journal, Vol. 5(4), (December 1998).

[Lam03] A. van Lamsweerde. From System Goals to Software Architecture. In Formal Methods for Software Architectures, M. Bernardo & P. Inverardi (eds), LNCS 2804, Springer-Verlag, 2003, 25-43.

[Lam00] A. van Lamsweerde. Requirements Engineering in the Year 00: a Research Perspective. International Conference on Software Engineering, 5-19, 2000.

[Mas05] P. Massonet, C. Ponsard. "A Scenario and Goal based Approach for Guaranteeing Quality of Service for Negotiated GRID Service Level Agreements: An Experience Report". Proceedings 1 rst International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements (in conjunction with RE05), August 30th, 2005, Paris, France.

[Mes02] D.A. Menascé, "QoS Issues in Web Services," IEEE Internet Computing, vol. 6, no. 6, 2002, pp. 72–75.

[Naq06] S. Naqvi, P. Massonet, A.E. Arenas. Security Requirements Model for Grid Data Management Systems. International Workshop on Critical Information Infrastructures Security (CRITIS 2006), Lecture Notes in Computer Science, vol. 4347, 2006.

[OMG02] Object Management Group, UML Profile for Scheduling, Performance, and Time, Draft Adopted Specification, OMG document number ptc/2006-12-03 (November 2002).

[OMG06] Object Management Group, UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Revision Task Froce Specification, OMG document number ptc/2006-11-01 (December 2006).

[Pap06] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, B. J. Krämer. Service-Oriented Computing: A Research Roadmap. In Service Oriented Computing, 2006.

[Pis05] M. Pistore, P. Traverso, P. Bertoli, A. Marconi. Automated Synthesis of Executable Web Service Compositions from BPEL4WS Processes" Special Track at the International World Wide Web Conference, 2005.

[Raj97] R. Rajkumar, K. Juvva, A. Molano, S. Oikawa, Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems, Tech. report Carnegie Mellon University, 1997.

[Sel00] B. Selic. A Generic Framework for Modeling Resources with UML, IEEE Computer, Vol. 33(.6), (June 2000).

[Sol04] M. Solanki, A. Cau, H. Zedan. Augmenting Semantic Web Service Descriptions with Compositional Specification. WWW '04: 13th international conference on World Wide Web,New York, NY, USA, 2004. ACM Press.

[Sta95] R. Staehli, J. Walpole and D. Maier, Quality of Service Specification for Multimedia Presentations, Multimedia Systems, Vol. 3 (5/6), 1995.

[Ven97] N. Venkatasubramanian and K. Nahrstedt, "An Integrated Metric for Video QoS", Proc. ACM Multimedia 97, 1997.

[Yan04] J. Yang, M.P. Papazoglou. Service Components for Managing the Life-Cycle of Service Compositions. Information Systems, vol. 28, no. 1, 2004.

[Yue87] K. Yue, What Does It Mean to Say that a Specification is Complete?, Fourth International Workshop on Software Specification and Design, Monterey, 1987.

[Zen03] Zeng, Liangzhao and Benatallah, Boualem and Dumas, Marlon and Kalagnanam, Jayant and Sheng, Quan Z. (2003) Quality Driven Web Services Composition. 2003.