Elements of QoS-Aware Specification Languages

Miguel A. de Miguel
E.T.S.I. Telecomunicación
Ciudad Universitaria, 28040 Madrid Spain
mmiguel@dit.upm.es

July 2002

Abstract

This paper introduces the main elements of languages that support QoS specifications. These elements are the constructors of QoS-aware models. Different types of languages are used to specify QoS systems, the most common include extensions of Interface Description Languages, UML extensions and metamodels, and mathematical models. These are different approach, although they use some common key elements. These QoS specification methods support the description of QoS concepts that are used for different purposes: i) generation of code for the management of QoS concepts (e.g., negotiation, access to resource managers), ii) specification of QoS-aware architectures, and iii) management of QoS information in QoS reflective infrastructures (e.g., QoS adaptable systems).

1. Introduction

Frequently the behavior of a system component is functionally correct, but the result it generates is nevertheless unacceptable because the result does not meet some quality of service (QoS) 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 components. A *flexible* component can trade off among the amounts of time and resources it uses to produce its results, the quality of its input, and the quality of its result. (For example, by carrying out a preprocessing step to enhance the received video, a visual tracking task can compensate for a lower quality video it receives as input at the expense of the processor time spent on enhancement.) Flexible components are feasible in many application domains. Researchers in applications as diverse as real-time computing, multimedia and intelligent systems have documented the fact that flexible systems gain in availability and graceful degradation. A complex system typically contains components that implement diverse applications. The components share resources and their execution behavior and input/output qualities are interdependent.

In addition to its functional behavior and internal structure, the developer of each component must consider its QoS requirements. For example, components such as pattern recognizers or signal filters have temporal requirements (e.g., maximum response times and jitters and minimum execution frequencies) and input and output accuracy requirements (e.g. percent of error in the pattern recognition as a function of noise in the input). If the component is flexible, the output quality depends both on input quality and available resources (e.g., amounts of CPU execution time and memory). Most of modeling languages provide support for the description of functional behavior, they include the non-functional requirement such a simple comments or using informal structures. An example are the interfaces that provide support for the description of functional services in some modeling and interface description languages, but they do not specify non-functional properties of implementators. When a client defines a dependency of these interfaces, it has no information about the quality properties.

QoS is defined as a set of perceivable characteristics expressed in user-friendly language with quantifiable parameters that may be subjective or objective [26]. Examples of objective parameters are startup delay, and data sizes. Subjective factors are the overall cost or the factors of importance of other parameters. Examples of QoS parameters for system resources are jitters, delays, blocking time, and size of buffers.

The characteristics of quality and their parameters are based on two types of subjects: i) user satisfaction, these parameters are based on the user or client requirements, and ii) resource consumption and system parameters, these are the parameters that support the resource managers of system infrastructures. Sometimes the user parameters depend on some properties of the functional architecture (e.g., types of algorithms, data redundancy, and limited execution time). In the process of analysis of QoS, we must establish the mapping between different user parameters and the resource parameters or the functional architectures to achieve the user qualities based on system parameters and the functional implementation of the system.

The application $f(qi,r) \rightarrow qo$ does the quality characterization of software components or the entire system. Where qi are the quality attributes of other components or external environment that affect to the quality of this component, r are the resources used in the component that affect to its qualities, and qo are the qualities provided. Examples of input and output qualities are the precision of input/output arguments, maximum frequency of input/output data, and the accuracy of output results. Examples of resource qualities are the maximum response time in CPU executions and network bandwidth. The application depends on the functional behavior (e.g., to support reliability we must include some type of redundancy in the architecture or implementation).

2. Solutions for Modeling QoS

Four general approach for the specification of QoS are: i) modeling languages, ii) interface languages, iii) application interfaces and component infrastructures, and iv) mathematical models. QoS infrastructures (e.g., middleware and component infrastructures), languages for the description of QoS architectures, or analysis methods use some of these four approaches for to description of QoS-aware systems or components. QoS infrastructures [28][27] provide some basic facilities for the QoS management (e.g., negotiation, adaptation, and monitoring); they represent the QoS requirements of software elements that they support (in general objects or components). Languages for QoS architectures [8][1][4][20] provide support to capture QoS aspects in detailed designs and architectures and express decisions about the component and subsystem structure. Some analytical methods of QoS provide metrics of designs and implementations. These methods are general QoS solutions [25] or specific of domains [26][23]. They provide support to make optimal the resource distribution or improve the user perceptible QoS attributes. The four approaches require a support for the description of QoS aspects and some of them require a run-time support to exchange and monitor QoS information.

ISO reference model for QoS [11] introduces some concepts (i.e., QoS Characteristics, QoS Contracts and QoS Capabilities) and a basic architecture that are basic elements of QoS specification.

2.1. QoS-enabled Modeling Languages

Examples of QoS-enabled modeling languages are QML (QoS Modeling Language) [8] and CQML (Component Quality Modeling Language) [1]. QML and CQML are languages with a BNF grammar. Other similar approaches are based on metamodels [4][2]. These languages provide support for the description of user defined QoS categories and characteristics, quality contracts and quality bindings. They are frameworks for the description of QoS Catalogs [6] of general QoS parameters, or application specific quality parameters. They do not provide support to optimize the resource allocation, or evaluate the levels of quality provided. They address the problem from the specification point of view.

Another approach is the description of resource services quality. [17][20] provide support for the description of quality based on resource services, and the relation with analytic methods of performances such as latencies and throughputs.

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

In Figure 1 a set of components Ci provide some services with quality attributes that can affect to the component C, which provide services with quality attribute to other components. C uses a ser or of resources that provide services with some quality attributes.

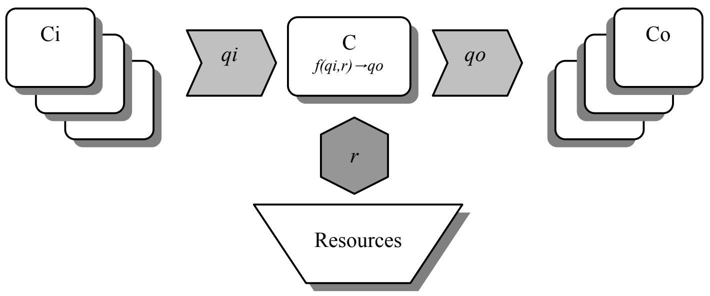

Figure 1. Mapping of Input, Output Qualities and Resources.

2.2. QoS-enabled Interface Description Languages

Examples of Interface Description Languages (IDL) QoS-aware are CDL (Contract Description Language) [14] and QIDL (Quality Interface Description Language) [15]. Both are languages integrated in object-oriented middleware frameworks for the support of QoS. CDL is part of QuO [28] and QIDL is included in MAQS [15]. MAQS and QuO use CDL and QIDL for the automatic generation of stubs and skeletons that support the management of some basic QoS functions (i.e., QoS negotiation, adaptation, and monitoring).

The integration of QoS facilities into middleware systems has been identified as a future challenge for the middleware infrastructures [9][21]. General middleware architectures [10][16][21] introduce general facilities, but their architectures are not dependent of communication middleware facilities. In some solutions [10][16], QoS middleware cooperates with existing solutions at OS and network levels, and proposes the middleware layer to support other facilities (e.g. adaptation). Other solutions are adapted to specific middleware environments [28][15].

The QoS-enabled IDLs support the description of *regions* that represent state of QoS components or objects. Constraint expressions describe the possible *regions*. The states have associated transitions that provide support for description of QoS *adaptation*. QoS-enabled IDLs include support to access the current state of system resources.

2.3. QoS Component Infrastructures

Some proposals study the integration of QoS facilities in component models such as CCM (CORBA Component Model) [27]. Wang et alt proposal [27] pays special attention to the QoS-enabled location transparency, reflective configuration of component server and container, and the strategies to reconfigure the component server. *Lusceta* [3] is a component model environment based on formal techniques, which can be simulated and analyzed. *Lusceta* provides support for the specification of QoS management, which can be used to synthesize (dynamic) QoS management components. The execution framework is a general QoS-aware reflective middleware. Another component modeling environment is presented in [18]. It proposes solutions for the description of component architectures and for evaluation of response times. This is an architectural environment not supported by execution environments. [7] introduces a solution for the integration of QoS basic services, such as resource reservation and negotiation, in EJB (Enterprise Java Beans).

The component infrastructures introduced use two techniques for the specification of QoS: application interfaces that are part of the infrastructure, and component descriptors that are used for the automatic generation of managers and containers that support the QoS aspects. In some solutions the component descriptors are XML files with data type structures for the specification of QoS attributes. Nevertheless, they do not provide support for the description of user guided QoS attributes.

2.4. QoS Analysis Methods

Analytical models for QoS management provide support for the application of metric evaluations and resource allocation optimization. [25] proposes a general QoS analytical model for the optimization 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 the resources allocation, utility functions and weighted utility functions evaluate the application and system quality. The optimization of these functions provides the optimal resource distribution.

Other approaches are domain specific. [26][23] are analytical models to support the QoS metrics of video and multimedia applications. They identify the QoS 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.

The RFC (Request For Comments) of IETF “Specification of Guaranteed Quality of Service“ [22] introduces the basic parameters and theorems for the analysis of deadline and jitters in QoS Guaranteed mode of Integrated Services in the Internet architecture.

3. Constructors of QoS Modeling Languages

QoS specification languages are based on a set of constructors that provide support to describe the main QoS elements of the problem. Nevertheless, the model requires a general reference architecture. We are going to consider the QoS specification for two different abstraction levels: QoS application analysis and QoS application architecture. In the first case we analyze the QoS problems of the systems that is going to be developed and in the second case we study the QoS problems of solutions. The general model for the QoS application architecture is based on ISO QoS general architecture [11].

The basic functional elements of the QoS model we will be considering is the resource-consuming component (RCC) for the QoS application architectures and the QoS-aware specification functions (QASF) for the QoS application analysis. QASF are significant services and functions of the new systems (specified from an analysis point of view) that have associated QoS requirements. These functions support the functional behavior of the system. However, the execution of each function will take time, require system resources and be subject to occasional system errors or failure. These and other similar features are non-functional behavior of the system. RCC is a processing entity that includes a group of concurrent units of execution, which cooperates in the execution of a certain activity and share common budgets. The budget is an assigned and guaranteed share of certain resources. An RCC has the following associated: i) facets (interfaces provided and synchronously used by RCC clients), ii) receptacles (interfaces synchronously used by this RCC), iii) event sinks (event queues supported by this RCC and asynchronously used by RCC clients), and iv) event sources (event queues asynchronously used by this RCC). UML can model RCC in different ways; in general, classes, component and interfaces are modeling elements that model the RCCs. At this point what we want to address is the identification of the main concepts that a QoS model includes.

QASFs and RCCs have non-functional characteristics associated, which can be general purpose or domain specific. In both cases QoS characteristics make reference to quantifiable non-functional attributes. The quantification with one or multiple dimensions is fundamental for the expression of QoS supported-provided, the monitoring of the characteristic, and evaluation of fulfillment and level of satisfaction.

A **quality characteristic** includes a set of quality attributes that are the dimensions to express a quality satisfaction. An example of quality characteristic to express latency constraints could include the following attributes: i) arrival patterns, the values of this enumerated quality value are: periodic, irregular, bounded, busty, unbounded, ii) minimum period, iii) maximum period, iv) jitter, v) burst interval, vi) burst size, vii) requirement type, the values of this enumerated quality value are: hard, soft, firm, viii) deadline hard, ix) deadline soft, and x) output jitter.

The facets, receptacles, event sinks and event sources interconnect the RCC group, which collaborate to provide support of QASF. They support QASF transforming input data and events into output data and events. The QASF are the external QoS system operations, which have a quality utility associated that express the degree of satisfaction of the operation, from the user or external system point of view. The quality utility is expressed in terms of quality types and quality constraints. The grouped RCC are not quality independent in the sense that their configuration and quality provided in their facets and event sink may limit the quality behavior of another RCC. The end-to-end quality of a qualified functionality depends on the sequence of transformations developed along the RCC sequence. For example, the end-to-end latency of a video signal transformation depends on the latency of all RCC involved in the transformation operation.

Quality levels express the quantifiable level of satisfaction of a non-functional property. An RCC can associate quality levels to its facets and event sinks. These quality levels are the RCC's quality provided contracts. To support the quality provided contracts, the RCC can require some minimum budgets and quality levels in its receptacles and event sources, and in the system resources. These quality levels are expressed in the **quality-required contracts**. Quality contracts are expressed in terms of the values associated to quality characteristics.

3.1. QoS Modeling Elements

A general QoS modeling language must provide support for the specification of:

- *Definition of QoS Characteristics*: *QoS Characteristic* is a quantifiable aspect of QoS, which is defined independently of the means by which it is represented or controlled [11]. *QoS Characteristics* are quantified with some specific parameters and methods, and with other characteristics with a lower abstraction level. QoS Characteristics can be grouped into categories that group characteristics of a common subject. Different enterprises and organizations use the same QoS Characteristics, but they use different evaluation methods or establish different hierarchies. An example of divergence is that standards like [12] do not identify specific QoS characteristics for performance, but other proposals like [5] do. Another specific example is what we can do to measure *availability* characteristics. We can do it with different levels of abstraction: i) a simple probability, and in a hypothetical domain, this is enough, and we do not enter in more details, ii) in other domains we require more details, and we use two arguments, the mean-time-to-repair (MTTR) and the mean-time-to-failure (MTTF), and the *availability* is the probability: MTTF / (MTTF + MTTR). iii) When the operations are transactions, this probability is not enough, and we need to introduce the availability period (the period that a client will be able to access times arbitrarily) and the availability makes reference to a *continius availability* [5][13]. Some authors would classify the last case such as a *reliability* quality.

Different domains require different levels of abstraction. We need enough flexibility to make the description of particular characteristics of specific domain environments possible. Examples of specific domains are consumer terminal and digital television, and video applications in workstation. Both domains are similar but they have their own specific quality parameters; screen resolution, frame size, compression quality, filter coefficients and type of processing, are not used in the same way in both domains.

- *QoS Force*: The *QoS Forces* define any kind of restriction that QASF and RCC impose on *QoS characteristics*. 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 *QoS Forces* are constraints that describe maximum response times, or the minimum number of errors supported. Sometimes the *QoS Characteristics* have associated interdependencies, for example, 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. The Figure 2 represents the dependencies of qualities qx, qy and qz for a hypothetical implementation function. The Figure represents 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 optimization of these functions and these functions can be restricted for specific analysis methods.

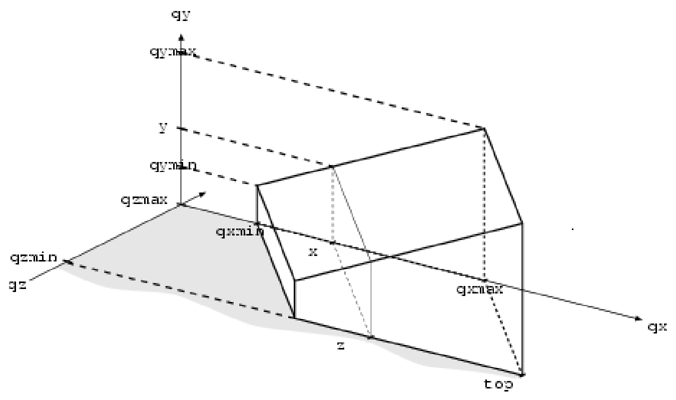

Figure 2. Relationship of qualities.

- *QoS Execution Modes*: Sometimes, QASFs and RCCs are designed to support different modes of executions. Each mode has different *QoS Forces* associated and their functionality can be different. Often, the execution modes for specific QASF or RCC are discrete (there is an enumeration of the different modes), for example, the levels of quality of a window in a digital television, the quality levels of this window are High, Medium and Low resolution, and the television system uses different implementation algorithms for each mode. But in some cases, the mode is defined with continuous values, for example, the maximum speed of the target in radar, and the quality level supported is specified with the values of the speed. In this case, the level is based on the real number that describes the speed. In general, the design of the functional architecture must take into account these modes, and there are different functions and components for each mode.
- *QoS Adaptation and Monitoring*: The transition from one execution mode to another requires some actions in the application execution, and some types of transitions are not allowed (we cannot change the quality level arbitrarily). Another common activity in some applications is monitoring QoS characteristics for the detection of errors and robustness. The monitors detect non-achievement of some QoS constraints, but we must specify the actions to be taken when the system does not achieve the quality levels.

Specific reservation protocols, admission control and analysis methods must specialize these general elements and restrict allowed values. They address solutions for platform independent models. Specific QoS frameworks that provide specific supports to the negotiation process require the specialization of some elements.

4. Conclusions

The main elements for the specification of QoS are the *QoS characteristics* that provide the vocabulary of QoS expressions and reward functions, the *QoS Forces* provide support to express the QoS constraints and functions that represent the interdependencies of QoS characteristics and parameters. QoS systems operate with different *QoS modes*, which depends on resource available or the quality the other component provide. Depending on resources available or the dynamic user requirements, the *QoS* systems *adapt* their behavior and change of quality level. These four concepts define the basic elements for the specification of QoS systems. Specification of QoS requirements and architectures can be done with these four basic types of elements.

These basic concepts provide support for the specification of QoS metamodels, profiles and specification languages in general. The models that they express can be used for the generation of code of QoS infrastructure frameworks (e.g., middleware and component), for the specification of QoS architectures and for the application of QoS analysis methods.

5. References

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

[2] 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).

[3] L. Blair, G. Blair, A. Andersen and T. Jones. "Formal Support for Dynamic QoS Management in the Development of Open Component-based Distributed Systems". *IEE Proceedings Software.* Vol. 148 No. 3. (June 2001).

[4] 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).

[5] M. Barbacci, T. Longstaff, M. Klein and C. Weinstock., *Quality Attributes,* CMU/SEI Technical Report No. CMU/SEI-95-TR-021 ESC-TR-95-021, (December 1995).

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

[7] M. de Miguel, J. Ruiz and M. García, "QoS-Aware Component Frameworks", Proc. International Workshop on Quality of Service, (May 2002).

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

[9] K. Geibs. "Middleware Challenges Ahead" Computer IEEE. (June 2001).

[10] C. Hou, C. Han, and Y. Min. "Communication Middleware and Software for QoS Control in Distributed Real-Time Environments". *In Proceedings Computer Software and Applications Conference. COMPSAC'97.* IEEE (1997).

[11] International Organization for Standardization, CD15935 Information Technology: Open Distributed Processing - Reference Model - Quality of Service, ISO document ISO/IEC JTC1/SC7 N1996 (October 1998).

[12] International Organization for Standardization, *Quality of Service: Framework,* ISO document ISO/IEC JTC1/SC 6 ISO/IEC 13236:1998 (December 1998).

[13] J. Koistinen, "Dimensions for Reliability Contracts in Distributed Object Systems", Hewlett Packard Technical Report, HPL-97-119 (October 1997).

[14] J. Loyall, R. Schantz, J. Zinky and D. Bakken, "Specifying and Measuring Quality of Service in Distributed Object Systems", Proc. 5th International Symposium on Object-Oriented Real-Time Distributed Computing, (April 1998).

[15] MAQS

[16] K. Nahrstedt, D. Xu, D. Wichadakul and B. Li. “QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments“. IEEE Communications Magazine. Vol. 39, No. 11. (November 2001).

[17] Object Management Group, *UML Profile for S*cheduling, Performance, and Time, Draft Adopted Specification, OMG document number ptc/2002-01-20 (January 2002).

[18] U. Rastofer and F. Bellosa. “An Approach to Component-based Software Engineering for Distributed Real-Time Systems“. *In Proceedings SCI 2000 Invited Session on Generative and Component-based Software Engineering.* IIIS (2000).

[19] D. Schmidt, V. Kachroo, Y. Krishnamurthy and F. Kuhns. “Developing Next-generation Distributed Applications with QoS-enabled DPE Middleware”. *IEEE Communications Magazine.* Vol. 17, No. 10. (October 2000).

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

[21] M. Shankar, M. de Miguel, and J. Liu. “An End-to-End QoS Management Architecture”. *In Proceedings of Real-Time Application Symposium. RTAS’99.* IEEE (1999).

[22] S. Shenker, C. Partridge and R. Guerin. “Specification of Guaranteed Quality of Service”, Internet RFC 2212 (September 1997). http://www.ietf.org/rfc/rfc2212.txt?number=2212

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

[24] A. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks,* PhD Thesis, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, (February 1992).

[25] R. Rajkumar, C. Lee, J. Lechoczky and D. Siewiorek, “A Resource Allocation Model for QoS Management”, Proc. Real-Time Systems Symposium. IEEE Computer Society (December 1997).

[26] N. Venkatasubramanian and K. Nahrstedt, “An Integrated Metric for Video QoS”, Proc. ACM Multimedia 97, (November 1997).

[27] N. Wang, D. Schmidt, M. Kircher, and. K. Parameswaran. “Adaptative and Reflective Middleware for QoS-Enabled CCM Applications”. *IEEE Distributed Systems Online* Vol 2 No. 5. (July 2001).

[28] J. Zinky, D. Bakken, and R. Schantz. “Architecture Support for Quality of Service for CORBA Objects“. *Theory and Practice of Object Systems*. Vol. 3 No. 1. (January 1997).