Published in the proceedings of WORDS '97, February 5-7, 1997 in Newport Beach, California.

Taxonomy for QoS Specifications*

Bikash Sabata Saurav Chatterjee Michael Davis Jaroslaw J. Sydir Thomas F. Lawrence

SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
{sabata, saurav, mdavis, sydir}@erg.sri.com

Rome Laboratory
Griffiss Air Force Base
NY 13441-4505
lawrence@cliff.cs.rl.af.mil

Abstract

It is becoming increasingly commonplace for multiple applications with different quality of service (QoS) requirements to share the resources of a distributed system. Within this environment, the resource management algorithms must take into account the QoS desired by applications and the ability of the system resources to provide it. In this paper we present a taxonomy for specifying QoS for the different components of a distributed system, from the applications down to the resources. We specify QoS as a combination of metrics and policies. QoS metrics are used to specify performance parameters, security requirements and the relative importance of the work in the system. We define three types of QoS performance parameters: Timeliness, Precision, and Accuracy. QoS policies capture application-specific policies that govern how an application is treated by the resource manager. Examples of such policies are management policies and the levels of service. In this paper we explore each of these components of the QoS taxonomy is detail.

1 Introduction

The explosive growth of the internet and intranets has caused a dramatic increase in the number of users that compete for the shared resources of distributed system environments. It is becoming increasingly important for distributed systems to be able to handle application demands for resources more intelligently. Since resources are shared by distributed applications with varying quality of service (QoS) requirements, these requirements must be taken into account by the resource allocation and scheduling algorithms.

QoS parameters are expressed in terms of different units at different layers of the system. Application QoS parameters are a function of the application's goals and design. Resource-level QoS parameters depend on the design of the resources and on their control parameters. Applications running in the system need to be able to specify their QoS requirements and preferably be able to operate over a range of QoS values. The system, in turn, needs to be aware of application QoS parameters and be able to translate them to resource-level QoS parameters. For example, the communication resources understand QoS parameters such as the packet delay and packet jitter, while a video on demand application understands QoS parameters such as frame delay and frame jitter. Although the translation between the two delay parameters is fairly simple, the translation between the two jitter parameters is not.

A distributed system that provides QoS guarantees to general applications, sharing general system resources, must be based on a QoS framework. Such a framework consists of a QoS specification taxonomy, and a QoS architecture that integrates the different components in the various layers of the system. A general taxonomy leads to a better understanding of the QoS parameters and their inter-relationships. The final goal is to have generic translation schemes that can be the QoS interfaces between the various system components. The QoS based framework will help in identifying the functional requirements for the management and monitoring mechanisms of the distributed system.

Our current research is directed towards developing an integrated QoS framework for managing distributed systems resources in order to provide application-level QoS guarantees. We specify QoS as a combination of ***metrics*** and ***policies***. Metrics measure specific quantifiable attributes of the system components. Policies dictate the behavior of the system components. The system components here range from application modules to middleware objects to resources. The different types of metrics are the

*This work is funded by Rome Laboratory under contract number F30602-95-C-0299

performance metrics, the security levels, and the cost metrics. Timeliness, precision and accuracy define a classification of the QoS performance parameters. The timeliness metrics measure the specifications that are related to the timing constraints of the work. Precision metrics measure the volume of work, and the accuracy metrics measure the amount of errors in the completion of the work. Depending on the point of view i.e., the application/user point of view, or system point of view, or resource point of view, the QoS specification is either requested by the application or provided to the application.

In this paper we define and classify the different QoS attributes and place them into a general QoS taxonomy. In section 2 we present a brief overview of past work. In section 3 we give a brief overview of the QoS-driven resource management architecture of which the taxonomy is a part. This is followed by a description of the taxonomy in section 4 and finally some conclusions and a discussion of future work in section 5.

2 Background

Although there has been work in QoS-driven resource management, most of this work has focused on either the network or operating system layer [2][6][8][10][13][14]. Resource management at each of these layers has been done separately, with very little understanding of how the confluence of resource management at the different layers can provide end-to-end QoS support to applications. As a result, these disjoint approaches to resource management are not sufficient for providing end-to-end application QoS support. Refer to [12] for a thorough description of previous work in each of these layers.

Campbell et. al. [1] have been among the first to recognize the need for an integrated approach to resource management. They presented an integrated framework that deals with end-to-end application QoS requirements. The notion of flow is introduced as an important abstraction within the framework. Flow is defined to characterize the production, transmission and consumption of the data associated with a single media. Flows are either unicast or multicast and generally require end-to-end admission control.

Based on the notion of flow, Campbell et al. define QoS to include specifications for *flow synchronization*, *flow performance*, *level of service*, *QoS management policy*, and *cost of service*. This taxonomy is the best we have seen in the literature. However, it fails to include important concepts such as precision of the data produced, and application security and fault handling requirements. The focus of all work in the area of QoS has been on the timing and error aspects (timeliness and accuracy) of systems. The volume of work needed to perform a service has not been considered to be a dynamic adjustable parameter. A QoS based system should be able to dynamically adjust the amount of work performed (e.g., by using hierarchical encoding of video data, or resizing the video frame). This would allow the system to make trade-offs between the various QoS parameters, when sufficient resources are not available.

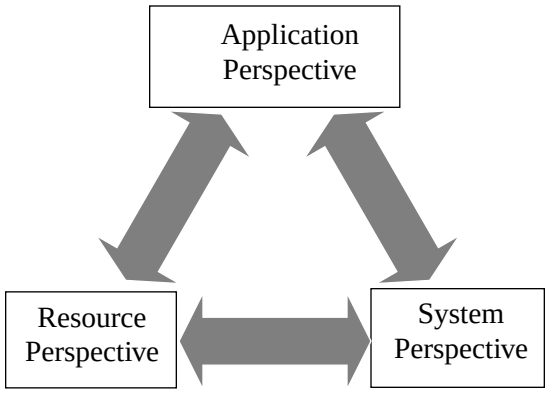

FIGURE 1 Three Resource Management Perspectives

3 QoS Driven Resource Management

We define QoS-driven resource management as the end-to-end allocation and scheduling of resources to applications, based on the QoS requirements of the applications. When faced with inadequate resources, the system is able to make trade-offs between the various aspects of application QoS because it understands each application's resource usage requirements as a function of the QoS that is provided to the user. There are three different perspectives from which the resource management problem can be viewed: application, resource, and the system (Figure 1).

Applications want access to enough system resources to achieve the desired level of QoS; they are not concerned about how this is done or how it affects other applications.We call this the ***application perspective***. We model application information using two abstractions. The application model captures the structure of the application and the load that it places on system resources. The benefit function abstraction captures the application's QoS requirements and its specifications regarding the relative importance of the various QoS metrics, in case the optimal QoS level cannot be provided by the resource manager.

The individual system resources (such as processors, disks or communication networks) also have a perspective on resource management. We call this the ***resource perspective***. Each resource is concerned only about managing access to itself and not about other resources or applications running on other resources. We model this perspec-

tive using the resource model, which captures resource-specific attributes such as local scheduling policies and the execution/concurrency behavior properties.

The system is composed of resources and supports applications. We denote this as the ***system perspective***. The system perspective captures all system policies. Examples of such policies include: end-to-end scheduling policies, policies to decide which application's QoS to degrade when there are not enough resources to provide the desired QoS to all applications, admissions control policies, policies governing the amount of effort and time that should be expended in attempting to find the optimal resource allocation. The system perspective also includes all the resource management algorithms that are found at the middleware level.

The objectives of the individual applications, the individual resources, and the system are likely to be in conflict; thus, the role of a resource manager is to resolve these conflicts. QoS-driven resource management is a particularly interesting problem because it must account for all three perspectives. Further detail about our QoS-driven resource management system architecture is presented in [12].

3.1 QoS Definition

In order to design a system where multiple applications coexist within a QoS management framework, it is necessary for them to either have a common understanding of how QoS should be specified, or to be able to map their individual specifications into a common one. To this end we define in a generic fashion the various facets of QoS. We believe that QoS parameters are grouped into ***metrics*** and ***policies***. The metrics measure quantifiable QoS attributes in the applications, system, and the resources. The metrics are further divided into performance metrics, security levels, and the relative importance. The policies describe the system behavior specifications. The primary policies are the management policies, and the levels of service. This taxonomy is defined in Section 4.

3.2 Application Model

We model an application using a directed graph, where graph nodes represent ***units of work*** (UoW) and graph edges represent ***data flow*** between these units of work; the data flow implicitly specifies the order in which the work must be done. This is shown in Figure 2, where

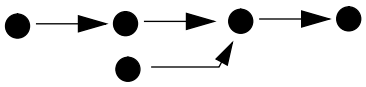

FIGURE 2 Application Model

the circles represent the work and the edges represent the data flow. The UoW represents the smallest granularity of work for which resources are allocated. A UoW is scheduled to be performed by a single resource. QoS requirements are defined for each UoW, along with the resource demand model, which describes the resource usage as a function of the QoS.

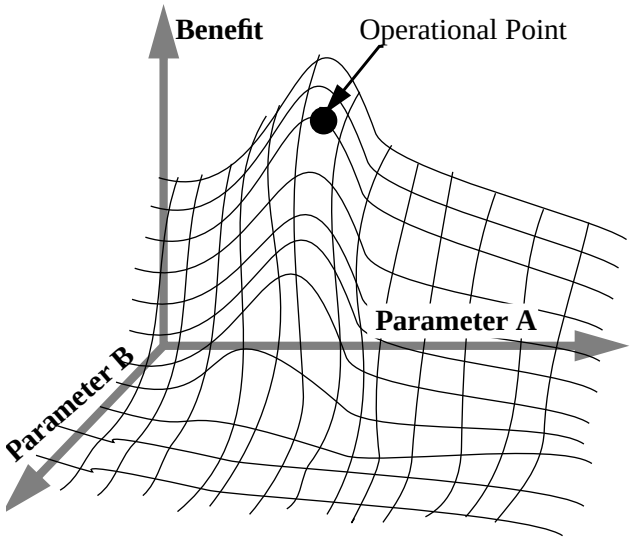

FIGURE 3 Example of a 2-parameter Benefit Function

A ***service*** is a collection of one or more units of work that may span multiple resources. A single QoS specification is provided for the entire service. QoS parameters are defined for each service, so that the invoker of a service negotiates the QoS of the entire service, without having to understand the UoWs that make up the service. A service may use other services to complete a task. Each instantiation of the service is defined to be the ***realization of service (RoS)***. A service may be realized in many different ways. The application model contains the information that the resource manager needs in order to allocate and schedule resources to that a given end-to-end QoS can be achieved. Details of the application model are presented in [5].

3.3 Benefit Functions

The application demands a certain level of QoS from the system. We have developed the ***benefit function abstraction*** to model an application's QoS requirements and preferences. The benefit function is a multidimensional graph specifying the benefit that the user receives if the system provides a certain level of QoS. The system will therefore attempt to provide QoS in such a manner as to maximize the application's benefit; this will be the operational point. An example benefit function is shown in Figure 3. The dimensions of the benefit function correspond to QoS metrics of interest to the application.

The benefit function is especially useful for facilitating a graceful degradation of the application QoS. If resources fail or are diverted towards higher-priority applications, the system may not be able to continue to provide the desired levels of QoS of all applications. The benefit function can then be used to make intelligent decisions regarding which applications' QoS to degrade, which QoS metrics to degrade, and by how much. The details of benefit functions are presented in [4].

3.4 Resource Model

The resource model captures information about the system resources which is required by the allocation algorithms. Examples of resource attributes that appear in the model are: resource types, performance characteristics, and scheduling policies. We also associate a cost function with the resource that describes the performance and the cost of each resource at the different operational points (QoS parameter values).

3.5 System Model

The system perspective encompasses both the application and resource perspectives, as well as end-to-end system policies. Because the various application and resource perspectives may have conflicting goals, the system perspective contains policies for reconciling these conflicting goals.

We model distributed systems by using a subsystem-resource hierarchical structure. Computing, communication, and storage resources form the bottom layer. A set of resources governed by a single resource management scheme form a *subsystem*. A set of subsystems governed by a single resource management scheme form a higher-level subsystem. This hierarchical structure continues until the complete system is defined.

Each subsystem is represented by a graph, where the graph nodes represent resources and other (child) subsystems, and graph edges represent the connectivity (topology) of the resources and child subsystems. The parent subsystem sees all of the resources and child subsystems within it as black boxes whose internal composition is hidden. Each subsystem has a *QoS interface* that enables the parent subsystem manager to communicate QoS information with each child subsystem manager. Parent subsystem managers convey QoS requirements via the QoS interface, and assume that resource and child subsystem managers will meet their QoS obligations; how a child subsystem does so is of no concern to its parent. This hierarchical representation enables us to model heterogeneous systems running different network protocols, operating systems, and resource management schemes. The details of the system model are presented in [9].

For the management of heterogeneous resources and different types of applications, the system defines a set of system policies. There are a number of different types of system policies. For example, system policies define the action that should be taken when a new application is started and there are not enough resources to perform it with the desired QoS. Should its QoS be degraded? Should the QoS of a less important application that is already running be degraded to free up some resources? There types of system-wide questions are answered in the system policies.

3.6 Management Algorithms

The purpose of the application, resource, and system models is to provide a structure within which the QoS-driven resource management algorithms can be defined. The basic problem is to provide appropriate QoS support to each application. The structure of the application, as described in the application model, defines the different combinations of UoWs that can be performed to complete the task. The management algorithms must choose the best method for performing the task, determine the resources on which the individual UoWs should run, reserve the required amount of these resources and schedule their use. If the desired resources are not available, the QoS is degraded in a negotiation with the application.

4 QoS Taxonomy

We classify QoS parameters using the taxonomy shown in Figure 4. The primary categories are metrics and policies. Metrics specify quantifiable QoS parameters. Metrics can be further grouped into the following classifications: *performance specifications, security levels, and relative importance.* Policies are divided into *level of service,* and *management policies.*

Performance QoS is defined in terms of *timeliness, precision,* and *accuracy.* Performance metrics specify the parameters related to the performance of a task. For example end-to-end delay, total volume of computations, and bit error rate are performance measures. For each performance parameter there are absolute specifications and consistency specifications. Consistency specifications define the relations between the different task flows and the different instances of a task flow. For example the jitter and synchronization metrics measure the consistency of the delay parameter between different instances of the task, and between different task flows, respectively.

Relative importance represents the price (cost) that the user is willing to pay for a service of a given quality (in a system where the users compete for resources) or a measure of the importance of the work (in a system of cooperating users). *Security levels* define the data security level that need to be provided to the applications [11].

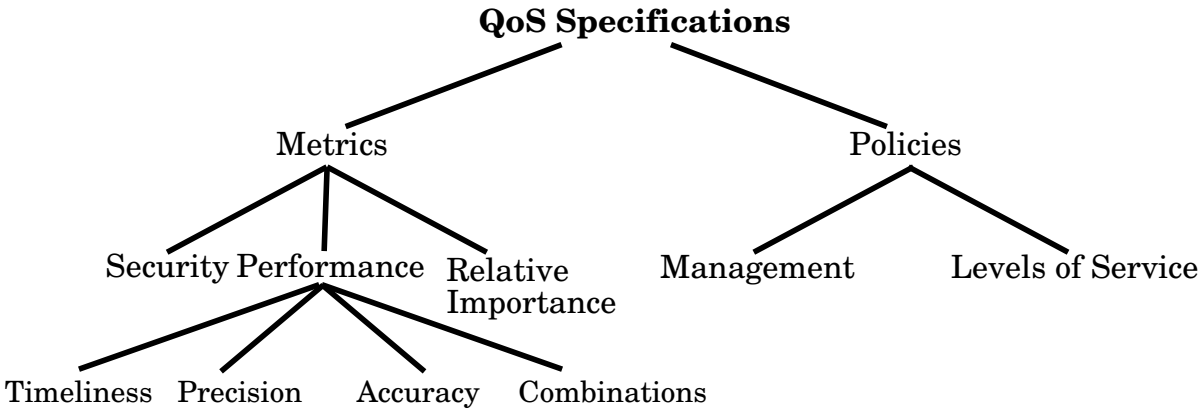

FIGURE 4 A Taxonomy for QoS Specifications

Level of service is defined as the type of QoS commitment given to the application. QoS *management policies* define the actions to be taken by the system under different situations. For example, in case of an unforeseen scarcity of resource, the application may be willing to go through a renegotiation and accept a lower quality of service instead of being denied the service. Management policies also describe the nature of the interaction between the applications and the system.

4.1 Performance Metrics

4.1.1 Timeliness

Timeliness parameters define a class of metrics that measure time related entities. Timeliness is expressed in units of time. Its definition is straightforward, because time is understood in the same way by humans and computers, and has roughly the same meaning in all layers of the system, from the applications down to the network. We have defined timeliness as a representation of the timing requirements for performing a given piece of work.

Timeliness parameters are metrics that measure

1. The total time taken to complete (begin to end) a task (UoW or service). This is measured as the delay, or latency, or time to complete.
2. The start time (earliest/latest) for a task.
3. The deadline (earliest/latest) for the task to complete.
4. The variability in time to complete a task (jitter). This measures the internal consistency of the timeliness parameters.
5. The relationship between the deadlines and the start times of the different tasks (synchronization). This measures the mutual consistency of the timeliness parameters.
6. The statistical distribution of each one of the above parameters.

The first three are absolute specifications, 4 and 5 are consistency specifications and finally 6 applies to all the parameters. The absolute metrics are used by schedulers to schedule the task at the appropriate time. The consistency specifications require the introduction of mechanisms that ensure that the time relationships between the different task flows and within a task flow are maintained. The statistical distribution of each parameter considers it to be a random variable and describes the distribution of the variable. This allows the tolerance associated with each parameter to be specified. Also, in the case of repeating tasks the statistical distribution describes how the parameters change over the different instances of the task.

4.1.2 Precision

Precision parameters specify the volume related quantities. Precision and accuracy require more detailed definition, since the same data can be viewed in different ways (and understood in different ways) by different components of the system. Since precision and accuracy are attributes of the data that flows through the application, it is important to distinguish between the content of the data in and the series of bits that represent the data. We define ***data content*** to be the meaning of the data. For example, the fact that a floating point number represents a median of some number of data points is its data content. Since a piece of data that is manipulated by a computer system must be represented as a series of bits, we define ***data representation*** to be the computer representation of a given piece of data. Thus, a given piece of data that is understood in terms of its data content can be represented by one or more data representations that differ in format and size. Since a data representation is a tangible construct, it has a spatial characteristic, which we refer to as the ***volume of data***. Volume of data is an amount of data in terms of a number of bits or bytes. It is important to note that the

volume of data in a data representation of a given data content depends on the algorithm used to encode the data. The same data content can have different representations whose volumes are different. For example, a floating point number can be represented in a C program as a float variable or as an ASCII character string.

We recognize that the amount of work that a computer system has to do to transfer or store data is proportional to the size of the data. Thus, the volume of data is a measure of the amount of work required to transfer or store it. Analogously, we define the volume of computational work as the total amount of computations (e.g. FLOPS) to complete the task. "Work" is used here in a very broad sense, to mean the usage of all involved hardware resources. Similar to the notion of the data content, a given computation executes a function that can be implemented in one of many possible ways. Each implementation does a different volume of computation to do the same amount of "work content".

We can now define precision as it applies to content and representation. In general, the ***precision of representation*** describes the amount of data or work. The ***precision of content*** is defined in terms of the specific data content or the functional transform. For example, the precision of the median value for some set of data points may be defined in terms of the number of decimal places to which it is calculated. The precision (or volume) of the representation is defined in terms of its volume (the physical amount of data in bits or bytes). The precision of content in one layer translates into a precision of representation in the next layer, e.g., a data packet in the transport layer is composed of the payload and the header; this content of the data structure is understood by the transport system. But the lower network layer recognizes the data only as a collection of data bytes and represents that as a series of bits. The units of volume may change for the different components of the system, e.g., the number of frames of video in the application translates to the number of bytes of data in the middleware and the network layers.

As we did with timeliness, we can define the jitter and synchronization of precision parameters, to capture consistency requirements. The variation in the volume in successive instances of tasks (UoWs or services) measures the consistency of the precision metric. For example, when video is compressed the application can choose to have the same number of bytes for each frame or may vary the total amount of data. This variation in the total volume of data between frames is captured in the internal consistency of the precision metric. The relationship between the precision of different flows is measured in terms of the consistency between the precision parameters of each flow. This may be important because the results from two different computations can be combined and the precision of the data generated in both flows have to be compatible for them to be combined. The precision parameters of interest are

1. The precision of content for input and output data.
2. The precision of representation for input and output data.
3. Internal consistency of precision over a flow (precision jitter).
4. Mutual consistency of precision between flows (precision synchronization).
5. The statistical distribution of the above parameters.

As in the case of timeliness, precision parameters 1 and 2 are absolute specifications; 3 and 4 define the consistency parameters; and 5 defines the statistical distribution of each precision parameter.

4.1.3 Accuracy

Accuracy measures the errors introduced into the data by UoWs and services. The accuracy of data content is also defined in terms of the specific data content. Using the median example, the precision of the data specifies the number of decimal places to which the median is calculated, while the accuracy of the data specifies how many of those decimal places actually contain correct (accurate) data. Finally, the accuracy of data representation is defined as the amount of data volume that is actually correct. (This amount is most naturally specified as a percentage.) In both the case of data content and that of data representation, accuracy is bounded by precision (e.g., a floating point number calculated to a precision of three decimal places can be accurate to at most three decimal places). The accuracy of computations is described in terms of the accuracy of the data generated by the computations. The accuracy parameters of interest are

1. The accuracy of content for input and output data.
2. The accuracy of representation for input and output data.
3. Statistical distribution of accuracy.

Notionally we can define an accuracy jitter and synchronization corresponding to the internal and mutual consistency parameters. However, practically there does not seem to be any application of such a concept.

4.1.4 Combination

The three classifications, timeliness, precision, and accuracy, do not represent independent (orthogonal) axes. Since the timeliness, precision, accuracy components of a QoS specification must be provided to the application "simultaneously", there are cases when QoS can be specified by a parameter that is a combination of them. Throughput, defined as precision over time, is one of these metrics. At this time, we are not aware of any other practical QoS metrics that are combinations.

4.2 Security

Security metrics deal specifically with policies and mechanisms related to data security that need to be provided to the applications [11]. We have defined two security parameters, level of confidentiality and level of integrity. Confidentiality is the problem of insuring that information doesn't get into the wrong hands. Integrity is the problem of making sure that information is and remains accurate. That is, the persons and processes that are allowed to modify a given piece of information are restricted to those that are trusted to do so. Integrity also applies to system components, making sure that they are not modified or replaced (presumably in order to violate one or more aspects of security policy). These two QoS parameters express the sensitivity of the data being handled with respect to confidentiality and integrity. The units in terms of which these parameters are specified are specific to the type of system. For example in military systems, they might be the military security levels (e.g., unclassified, secret).

Availability is commonly used as a third security parameter. Availability is the problem of insuring that there are sufficient computing resources to perform the required work at the desired time. Since the provision of an availability guarantee involves not only security, but also fault tolerance and resource management, we place the availability parameter in the levels of service category. It is discussed in Section 4.4.1.

4.3 Relative Importance

In order to allocate a resource to multiple, competing applications the resource management system requires a way to evaluate the relative importance of the different applications that are contending for the resource.

In the case of competing users (as in the case of commercial systems) the price (cost) that the user is willing to pay for a service of a given quality is an effective mechanism for determining application priority. While in the case of cooperating users (as in the case of military systems) the importance of the user and the application can be absolutely defined and used to gauge the relative importance of the work. We also define resource cost functions to express a resource's willingness to provide a given QoS setting. In the event of a scarcity of a given type of resource the cost of using that resource increases.

4.4 Policies

4.4.1 Levels of Service

Applications require an assurance about the system's level of commitment to providing their QoS needs. This commitment dictates the policy adopted by the resource manager to provide the service to the application. The specific policies can range from a best effort policy (no guarantees) to a policy of providing a very high level of assurance that application QoS will be maintained at all costs. We define ***level of service*** to be the level of commitment for a task. A service is either a ***guaranteed service*** or a ***best effort service***. The distinction between the two is that the system may provide no benefit to the user of a best effort service, while the user of a guaranteed service is promised a given level of benefit. Level of service is a meta-specification of the QoS parameters. It provides a policy statement about the way each performance parameter needs to be monitored and manipulated.

Different levels of guaranteed service can be provided by the system. For example, missing even one QoS requirement can lead to a catastrophic failure for many control and defense applications. On the other hand, it is generally not a catastrophe if the system occasionally misses an audio or video application's QoS requirements. We use the ***availability*** QoS metric to let the application convey what level of guaranteed service it requires. Availability is expressed as the probability that the QoS assurances will be met, thus it is a meta-attribute for the other QoS metrics.

4.4.2 Management Policies

QoS management policies define the application--specific actions to be taken by the resource manager under different situations. For example, in the case of an unforeseen scarcity of resources, the application may be willing to go through a renegotiation and accept a lower quality of service instead of being denied the service. There are no quantifiable metrics that describe these specifications, but in general, the policies can be classified into different classes of management functions. The user may specify a class as part of the application requirements. For example, the classes might be "renegotiation allowed" and "renegotiation not allowed". This again is a meta-level specification for the other QoS metrics.

5 Conclusions and Future Work

We have seen an increase in the utilization of distributed computing for a wide range of computing tasks. Distributed computing is not only attractive economically but also has desirable reliability and fault tolerance properties. However, since applications ranging from mission critical to leisure, all coexist in the same large system of systems, it is important that resources and the applications be managed intelligently to provide an acceptable level of service to all applications and guarantees to critical applications.

Applications need services from the system to complete their tasks. The needs of an application are best understood by the application and the mechanism used to convey the requirements to the system is the language of

QoS. QoS must tie the user's needs to the amount of resources required to provide them. Because distributed systems can be composed of heterogeneous components, it is important that QoS be defined in a generic fashion. Such a definition will allow the system designer to define a uniform QoS interface to all system components, enabling the resource manager to provide end-to-end QoS guarantees to applications running on these heterogeneous components.

In this paper we presented a taxonomy of QoS specifications that is the starting point for defining the QoS interfaces between the different system components. The key to understanding QoS based distributed systems is recognizing the relationships between the different ways that QoS concepts are defined by different system components. The taxonomy that we presented clearly identifies the different classes of QoS parameters. We see that the fundamental metrics are those of performance, cost and the security. The performance metrics have classifications of timeliness, precision and accuracy. The notion of having three primary classes of performance is important because most of the QoS specifications we have seen are just a statement of one or two of these classes. Within each class of the QoS specification there are requirements for absolute quantities and consistency measures. This gives a large dynamic range of parameter specifications. Finally, the notion of having a statistical distribution of each one of the parameters allows the specification of parameter tolerance and variability. The classification of QoS metrics into their most basic groupings provides insights into the common mechanisms for translating QoS specifications between system layers

We are developing a framework for defining QoS-driven system architectures, based on this QoS taxonomy. This framework will be useful in the design and management of large systems of systems. Generalizing the notions of applications, systems, resources, and QoS make it clear that the many different types of management mechanisms found in distributed systems are related to one another. Our unified framework ties together research in the areas of security and fault management to research in the area of QoS based resource management. In the next phase of our research we will take our QoS taxonomy and develop a complete system model and design management algorithms for each class of QoS parameter.

Acknowledgments

We would like to acknowledge the help of Dr. Ambatipudi Sastry and Elin Klaseen from SRI International, Charles Hammond, Hien Nguyen, and Pamela Clark from Stanford Telecom for numerous technical discussions about the ideas presented in this paper. Their continuous critique and feedback helped in the clarification of many concepts.

References

[1] A. Campbell, Aurrecoechea, & Hauw, "*A Review of QoS Architectures*," Proc. of the 4th IFIP International workshop on Quality of Service (IWQS '96), Paris, March 1996.

[2] A. Campbell, Coulson and Garcia, "*Integrated QoS for Multimedia Communications*," Proc. of IEEE InfoCom, Los Alamitos, CA, 1993, pp 732-739.

[3] A. Campbell, Coulson & Hutchison, "*A Quality of Service Architecture*," ACM SIGCOMM, Computer Communication Review, Vol. 24, pp 6-27, Apr 1994.

[4] S. Chatterjee, B. Sabata, J. Sydir, and T. Lawrence, *"Benefit functions for QoS trade-offs,"* SRI Technical Report, Forthcoming.

[5] S. Chatterjee, M. Davis, B. Sabata, J. Sydir, and T. Lawrence, *"Modeling Distributed Applications with QoS requirements,"* SRI Technical Report, Forthcoming

[6] D. Katcher, H. Arakawa, and J. Strosnider, "*Engineering and Analysis of Real-Time Microkernels*," Proceedings of the 9th IEEE Workshop on Real-Time Operating Systems and Software, 1(1):15-19, May 1992.

[7] K. Nahrstedt & Jonathan Smith, "*The QoS Broker*," IEEE Multimedia, pp 53-67, Spring 1995.

[8] K. Nahrstedt and Ralf Steinmetz, "*Resource Management in Networked Multimedia Systems*," IEEE Computer, May 1995, pp 52-63.

[9] B. Sabata, S. Chatterjee, J. Sydir, and T. Lawrence, *"Hierarchical Modeling of Systems for QoS based Distributed Resource Management,"* SRI Technical Report, Forthcoming.

[10] D. Stewart, D. Schmitz, and P. Khosla, "*Chimera II: A real-time multiprocessing environment for sensor-based control*," Proceedings of the IEEE International Symposium on Intelligent Control, 1989.

[11] J. Sydir, S. Chatterjee, M. Davis, B. Sabata, and T. Lawrence, *"Relationship between security and QoS driven resource management,"* SRI Technical report, Forthcoming.

[12] J. Sydir, S. Chatterjee, B. Sabata, and T. Lawrence, "*QUASAR: QUAlity of Service Architecture for Resource management,"* SRI Technical Report, Forthcoming.

[13] D. Verma, H. Zhang, and D. Ferrari, "*Guaranteeing Delay Jitter Bounds in Packet-switched Networks*," Proceedings of Tricomm 91, pp35-46, April 1991.

[14] Vogel, Kerherve, Bochmann, and Gecsei, "*Distributed Multimedia and QOS: A Survey*," IEEE Multimedia, Summer 1995, pp 10 - 19.