JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2007**

Vol. 6, No. 11, Special Issue on Advances in Quality of Service Management, December 2007

Quality of Service Contract Specification, Establishment, and Monitoring for Service Level Management

Changzhou Wang, Boeing Phantom Works, Seattle, WA, USA
Guijun Wang, Boeing Phantom Works, Seattle, WA, USA
Haiqin Wang, Boeing Phantom Works, Seattle, WA, USA
Alice Chen, Boeing Phantom Works, Seattle, WA, USA
Rodolfo Santiago, Boeing Phantom Works, Seattle, WA, USA

Abstract

This paper describes a Quality of Service (QoS) management approach and architecture as well as a case study for Service Level Management (SLM). Our approach brings in a new perspective to the SLM probem by using QoS management and QoS Contract specification, establishment, and monitoring. In SLM, the service consumer side and the service provider side must share a common understanding of QoS characteristics and use a common language for specifying desired QoS parameters in the form of QoS contracts. A service consumer must negotiate with the service provider to establish mutually agreed QoS contracts for an interaction session. When establising a new QoS contract, the service provider must consider both QoS contracts already agreed upon with existing consumers and system resource conditions. Similarly, a service consumer must be prepared in revising its contract with the service provider as conditions change over time. Once a QoS contract is established, SLM must monitor QoS status to make sure that the service quality is provided at the agreed range. If necessary, SLM must activate adaptation mechanisms to bring the service quality to the desired level. A case study is presented in this paper to validate the QoS contract management design approach and architecture for SLM.

1 INTRODUCTION

Service level management (SLM) is a process that involves the creation of service level agreements (SLAs), provisioning of system resources, and management of system performance to meet the demands in the SLAs.

A SLA typically includes description of involving parties (both service consumers and providers), services, Quality of Service (QoS) contracts, and obligations [12]. Description of involved parties identifies service consumers and providers and their relevant properties. Service consumers could be end-users,

Cite this article as follows: Changzhou Wang, Guijun Wang, Haiqin Wang, Alice Chen and Rodolfo Santiago: "Quality of Service Contract Specification, Establishment, and Monitoring for Service Level Management", in *Journal of Object Technology*, vol. 6, no. 11, Special Issue December 2007, pp. 25-44 http://www.jot.fm/issues/issue_2007_12/article2/

applications, or components of an application. Properties may include addresses, security information, accounting information, etc. Description of services includes the capabilities of the provided services and their QoS characteristics. QoS contracts define the QoS parameters agreed upon between service consumers and providers. Obligations define the guarantees, constraints, and penalties based on measured actual QoS parameters and those in the QoS contracts.

A traditional approach to SLM is a monitoring based approach where SLA is negotiated offline and key performance parameters are monitored at runtime. Our approach is based on a QoS management. QoS management is critical for SLM because it brings a comprehensive set of services in a QoS management architecture and provides automated policy management, contract establishment, resource management, prediction, monitoring, diagnosis, and adaptation towards an autonomic computing paradigm. In practice, different service consumers often have different QoS requirements. In addition, both customer satisfaction rate and business operation cost largely depend on how these QoS requirements are met during runtime. As a result, effective management of the QoS is a key requirement for the success of the SLM.

A fundamental issue to the QoS management is the specification, establishment, and monitoring of QoS contracts. Advanced SLM features like diagnostics and prognostics, autonomic and dynamic resource management, as well as adaptation are built on top of QoS contract management. An effective enterprise SLM requires an integration of these fundamental concepts and advanced features in a QoS framework and architecture.

In this paper, we discuss the QoS framework and architecture, and in particular an approach for QoS contract specification, establishment, and monitoring.

The rest of the paper is organized as follows. Section 2 introduces some background knowledge and overviews the related work. Section 3 presents our QoS specification framework. Section 4 describes our QoS management architecture and focuses on QoS contract establishment and resource management to support QoS management. Section 5 discusses QoS contract monitoring, diagnostics, and adaptation. Section 6 reports a case study of the proposed QoS contract specification, establishment and monitoring approach in a publish-and-subscribe based messaging system. Section 7 concludes the paper and points out some future work.

2 BACKGROUND AND RELATED WORK

Past research in three areas, namely QoS, SLA/SLM, and Service Modeling and Analysis, provided background concepts and basic frameworks for enterprise SLM.

ISO/IEC QoS Framework [1] [2] defines general QoS management concepts and guidelines. RM-ODP [3] further defines QoS management concepts for distributed object-oriented systems in terms of objects and their interactions. Concepts and guidelines from these standards have provided a conceptual framework for enterprise QoS management. W3C [4] further specifies reliability characteristics for web services. Our previous work [5] [6] [7] extended these conceptual frameworks with architecture and implementation of QoS management for enterprise distributed computing systems. QoS research in the communication networks has focused on message delivery QoS issues at the packet level through labeling, scheduling, routing, and switching mechanisms [8]. [9] [10] extended traditional QoS research from the network communication area to the end systems (e.g., OS and devices) and multimedia applications. In particular, [11] proposed a QoS specification language for multimedia applications to describe the QoS at different levels. In comparison, this paper focuses on the application level QoS specifications and considers the transformation from application level QoS to lower level QoS as part of the QoS management, especially the resource management component.

Traditional SLA/SLM focused on enterprise performance, reliability, and availability issues in a client-server architecture. Recently, work in SLA/SLM for service-oriented architecture showed the extended scale and complexity of performance, reliability, and availability management issues. [12] proposed an XML-based language for expressing quality properties in the web service level agreement (WSLA). [13] used WSLA and provided an overview of the management elements of WSLA in a utility computing framework. This framework consists of a WSLA language, resource provisioning mechanisms, a workload management system that prioritizes requests according to SLAs, and a system to monitor compliance with SLAs. [14] described a metering service in utility computing. The metering service is used to measure performance parameters and compute utility metrics such as resource utilization and rate. While we believe standards like WSLA are important and should be used eventually, research and experiments on what essential elements to SLA/SLM are needed before such standards can be effectively specified and utilized. We believe a QoS management perspective for various type of applications (e.g., task-based, message-oriented, or streaming multimedia) is needed.

In the enterprise service modeling and analysis, QoS issues have become critical aspects of services along with their functional aspects. [15] described an approach to model non-functional aspects such as security and QoS along with the modeling of functional properties in a model-driven development. In this approach, two modeling spaces, the design space and the analysis space, are used for functional design and quality design, respectively. The two spaces are integrated by means of model transformations. Non-functional design is to identify suitable QoS metrics and define the confidence that system designs meet targets expressed in terms of these metrics. [16] proposed the use of a Probabilistic Computational Tree Logic (PCTL) to express quality constraints involving time and probabilities, associate constraints with a software components at design time, and verify these constraints over the implementation at runtime.

Until recently, research in these areas has been in three separate thrusts. Our work described in this paper is a step forward in their convergence. We aim to integrate enterprise service modeling and analysis, QoS, and SLA/SLM in a comprehensive QoS management architecture and technology for enterprise SLM.

3 QOS SPECIFICATION FRAMEWORK

Our specification framework focuses on providing a specification language and an associated software tool to define QoS requirements, offers and contracts. The foundation of this framework relies on the common understanding of various QoS characteristics and their relationship. Standard bodies such as ISO and OASIS have defined many commonly-used QoS characteristics [1][2][3][4]. In our framework, we consider three commonly used types of services: (1) Task: task-oriented services perform operations on demand (e.g., Web Service); (2) Message: message-oriented services deliver pieces of information from a source to a destination on demand; and (3) Streaming Media: streaming media services deliver stream of information continuously. As a result, we identified important and common QoS characteristics in these types of services as following:

- Accountability: the correct identification of the service consumer, provider and involved actions of each party.
- Availability: the fraction of time that the service is available.
- Confidentiality: the secrecy of information, i.e., the message content or the request parameter cannot be leaked to unauthorized parties.
- Criticality: the importance or value of the request. For example, when contract violation is inevitable, less critical requests will be sacrificed in order to meet more critical requests.
- Deadline: the urgency of the service request. Deadline can be hard or soft depending on the value of the service provided after the deadline.
- Information Accuracy: information content may be compressed or approximated to certain degree. For example, images can be compressed or reduced to a lower resolution; movie frames can be selectively dropped.
- Information Throughput: the amount of information transported in a unit of time.
- Integrity: the correctness of information, i.e., the message content or the request parameter is not changed during transmission.
- Message Delay: the end-to-end delay in delivering the message from the source to the destination.
- Message Delivery Guarantee or Loss Ratio: whether the message must be delivered, if not, an upper bound may be given on the failure probability.
- Message Duplication Elimination: whether a single message can be duplicated and multiple copies delivered.
- Message Ordering: whether multiple messages shall be delivered to the destination in the same order as they are received by the service provider. The order may be imposed on messages from a single message source or a group of sources.
- Priority: the preference to handle the request in comparison with other requests.

- Retry Limit: the maximum number of times for the service provider to retry to deliver messages to or perform tasks for the service consumer when the initial operation fails.
- Streaming Media Jitter: the variance of the inter-arrival time for consecutive frames.
- Task Response Time: the time between request submission and response reception.

A key observation on these QoS characteristics is that they are inter-related. Many QoS characteristics are independent from each other and any combination is possible. For example, Delivery Guarantee, Duplication Elimination, Confidentiality and Throughput are four orthogonal dimensions. Notice that these dimensions are independent only from the service consumer's point of view. There may be implementation constraints such as resource limitations that prevent the service provider from supporting some combinations in these dimensions. For example, high Throughput requires quick handling of communication messages and hence may not allow sophisticated encryption methods for high Confidentiality. Indeed, almost all QoS characteristics supported by a single service provider are related in this sense.

More interestingly, some QoS characteristics are closely related to each other. For example:

- When the maximum Loss Ratio becomes 0, it is equivalent to Guaranteed Delivery.
- Response Time and Throughput are often inversely correlated, when the volume of information per request/response is given.
- Time to Live and Deadline might be positively correlated in message delivering systems.
- Retry Limit seems to be incompatible with Delivery Guarantee. Theoretically, the service provider should retry unlimited number of times, if necessary, to guarantee message deliveries. In practice, 100% delivery guarantee is sometimes impossible to achieve. For example, if a consumer fails for an extended period of time, the pending messages cannot be held within the predefined main/secondary memory. In this case, the Guaranteed Delivery might be interpreted as the fact that the service provider will try a limited number of times to deliver the message.
- Security dimensions are often positively correlated. Especially, high Confidentiality and Integrity often require high Accountability.

QoS Specifications in the requirements, offers and contracts are often defined using these QoS characteristics and their allowed or desired values. The specification of the allowed and desired values depends on the type of the value domains:

- Nominal (categorical values with no order among them). In this case, individual values are directly listed, such as TRUE for Delivery Guarantee.
- Ordinal (categorical values with a full order among them). In this case, a range of values can be given by the lower and/or upper bound. For example, Confidentiality is MEDIUM or above, or Criticality is between Green and Orange (assuming Criticality can be Green, Yellow, Orange and Red in the importance order).

- Numeric. In this case, common statistics such as minimum, maximum, and average may be applied on the values over a specified window of time. The time window can be either consecutive or sliding.

In addition, for the same QoS characteristics, multiple sets of the allowed values may be specified, for example, one for the average situation, and another for some limited peak period.

In our framework, the QoS specification is defined by an XML Schema due to its flexibility, expressiveness, and the wide acceptance in the industry. In other words, our QoS specification language is an XML-based language. The language defines the appropriate QoS characteristics in the application domain, and the allowed/desired values for each QoS characteristic. In addition, the language includes the constraints on QoS characteristics to represent their relationship.

Our framework includes a tool for end users to generate, modify and validate QoS specifications in the given language. The tool facilitates easy generation of the QoS specifications without requiring the user to remember the supported set of QoS characteristics and their value domains. More importantly, it guides the user to create correct specifications and validates the generated specifications to ensure that the constraints among different QoS dimensions are met.

In practice, different service consumers often have different QoS requirements (e.g., various data downloading bandwidths). On the other hand, the service provider usually supports different QoS offers (e.g., gold or silver service). Before a consumer subscribes to the provider for actual services, it needs to establish a QoS contract as a mutual agreement with the provider on the guarantee level of various QoS characteristics. Once the contract is created, both sides shall stick to the contract. For example, the consumer shall not issue excessive requests and the provider shall meet the agreed level of performance. Finally, the contracts may be revised at some later time due to the dynamic changes in the business and technical environment.

4 QOS CONTRACT ESTABLISHMENT AND RESOURCE MANAGEMENT

In [5][6][7], we introduced an integrated QoS management architecture to support the QoS contract negotiation, establishment, revision and maintenance. To facilitate this support, it also includes functionalities to support admission control, resource management, prediction, monitoring, and adaptation. Figure 1 illustrates our QoS management architecture, which consists of component services, their interactions, and interfaces with external services such as real-time host and network condition monitoring through Commercial Off-The Shelf (COTS) monitoring tools.

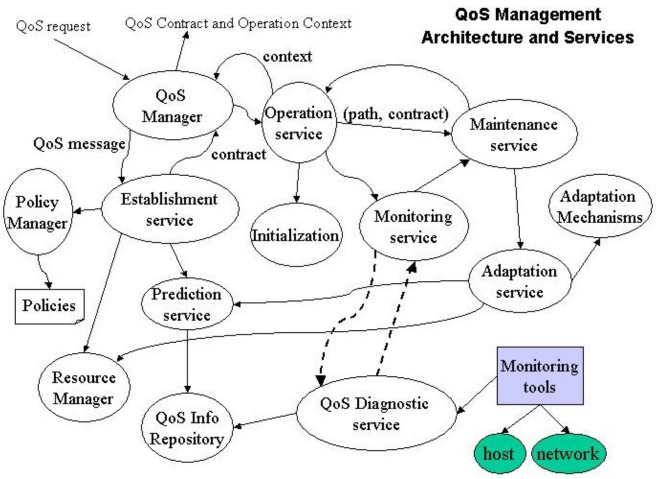


Figure 1. QoS Management Component Services

This architecture includes the following component services.

- *QoS Manager*. Provide an interface to the client for QoS contract negotiation, and orchestrate the establishment and maintenance of QoS contracts.
- *Establishment Service*. Establish QoS contracts based on requirements.
- *Policy Manager*. Provide admission control, resource management, monitoring and adaptation strategies as specified in policies.
- *Resource Manager*. Manage resource lifecycle: reservation, allocation, and release.
- *Prediction Service*. Predict future resource usage, for some resources or the whole system, with or without perturbation.
- *Operation Service*. Coordinate the services during the execution of a QoS contract.
- *Maintenance Service*. Maintain the QoS guarantee level for each contract.
- *Adaptation Service*. Change resource settings to maintain key QoS parameters within normal ranges, or provide graceful degradation for contract violations.
- *Monitoring Service*. Monitor contract health, and system conditions given by Diagnostic Service.
- *Diagnostic Service.* Aggregate real-time inputs from external system monitoring tools to generate high-level system condition information.

Service providers often publish their service offers in some registry. Service consumers usually initiate the QoS contract negotiation process with the service providers based on its knowledge of the service offers. The QoS Manager in the service provider provides public interface to facilitate the QoS contract negotiation and other contract management functionalities. The follow code snippet highlights such an interface defined in our QoS management architecture.

```
interface QosManager {
  QosContract establish(qosRequirement);
  QosContract revise(qosContract, qosRequirement);
  void agree(qosContract);
  void abort(qosContract);
  void release(qosContract);
}
```

To establish a QoS contract, a service consumer first call the `establish` method with its QoS requirements. In the service provider, the QoS Manager will forward the request to the Establishment Service which will consult the Resource Manager, the Prediction Service and the Policy Manager to create an initial QoS contract. This contract may not meet all the required QoS due to policy or resource limitation. It will be returned to the consumer, which will decide whether it is satisfactory. If so, the consumer will notify the provider by calling `agree`. Otherwise, the consumer can modify its requirements and send the request again by calling `revise`. This contract establishment process goes on until both sides agree on the same contract. In this process, the consumer can choose to abort the negotiation (by calling `abort`).

After the contract is established (i.e., the consumer's call of agree succeeds), the consumer will use the service for a period of time, and eventually decides to end the service. The consumer should release the QoS contract at the end of the service. Otherwise, the provider needs to detect service termination and release the QoS contract appropriately. In addition, during the service period, the consumer may also decide to modify the QoS levels due to the dynamic business and technical environmental changes. This can be done by a sequence of calls of `revise` followed by a single call of `agree`, similar to the initial establishment phase (except the initial call of `establish`).

In order to meet the QoS contract, the service provider must carefully manage the resources, since many different consumers (using the service at the same time) compete for the limited resources. This is often the case because service providers want to maximize their resource utilization and profit or benefit.

Resource can be managed through static or dynamic provisioning. In the static resource provisioning approach, fixed amount of resources are allocated for each consumer based on the QoS contract with it. In order to minimize contract violations, resources are allocated according to the worst case scenario when the service load of the consumer's requests is maximized. This usually results in waste of large fraction of resources. In addition, this approach also requires a clear understanding of the exact relationship between the service requests (with the QoS contract) and the required resources, often through extensive modeling and simulation.

In the dynamic resource provisioning approach, resources are initially allocated to meet average requirements on the services with each given QoS contract. After that, the service load for each QoS contract is monitored to detect change of resource demand for meeting the QoS requirements. Whenever the service load for a consumer reaches some threshold level, adaptation mechanisms will be triggered in order to maintain the QoS level. The details of the monitoring and adaptation will be discussed in the next section.

A clear advantage of the dynamic provisioning approach is its effective use of resources. Usually, the service loads for different consumers will not reach the peak at the same time. The dynamic provisioning approach enables reuse of certain resources for different consumers at different time: whenever a consumer's service requests reach the peak load, the reusable resources will be allocated to serve that consumer. When multiple consumers do reach their peak service request loads at the same time, the provider may be willing to take some penalty by degrading some QoS contracts for the moment. The trade-off between the resource utilization ratio and the contract degradation ratio is a key tuning factor of the dynamic provisioning approach.

5 QOS CONTRACT MONITORING, DIAGNOSTICS AND ADAPTATION

To determine how well the service provider and the service consumer perform in compliance to the QoS parameters agreed in a contract, a monitoring service is used to collect and sort performance data pertaining to a contract and aggregate the data into metrics that can be used to evaluate the compliance. Should a contract be violated, the corrective actions are taken on behalf of the management environment.

Monitoring performance parameters in the context of contracts involves monitoring real-time computational resource usage condition on the service provider side. It also involves monitoring service consumer's actual usage of the services and comparing it against the defined threshold in contract. For example in a publish-and-subscribe based messaging system, the messages sent by a publisher may exceed the publishing rate defined in the contract, or the end-to-end delay may be longer than what is agreed.

Some of these QoS parameters, such as task response time and message publishing rates, especially the resource utilization parameters, can be measured from inside the service provider. Others like the throughput and the end-to-end delay require probing the service consumers.

There are various COTS tools for monitoring the system performance. However, the traditional monitoring tools are not sufficient in QoS management system, as many concurrent contracts and shared system resources could fluctuate over time, and system health conditions could change. There is a need for a more comprehensive monitoring approach that is integrated with diagnostics and adaptations.

As shown in Figure 1, our approach uses monitoring, diagnostic and adaptation services as an integral part of end-to-end QoS management. The role of Monitoring Service is to sample and aggregate QoS parameter values. It registers condition predicates with the Diagnostics Service, which returns with notifications when the predicates become true due to changes in system conditions. The Diagnostics Service is a vital service that uses formal reasoning models like causal networks or Bayesian networks to aggregate low-level system signals into attributes on system conditions. It takes real-time inputs from monitoring tools, aggregates data on the fly, and stores the data in a repository. It may also evaluate any predicates on the attributes upon value changes and trigger notifications to interested parties such as Monitoring Service. When Monitoring Service receives the notifications of the conditions of interest, it updates the corresponding data in Maintenance Service, which in turn activates some adaptation mechanisms, defined in the policy, to take care of the situation. Figure 2 depicts the interactions between Monitoring Service and Diagnostics Service.

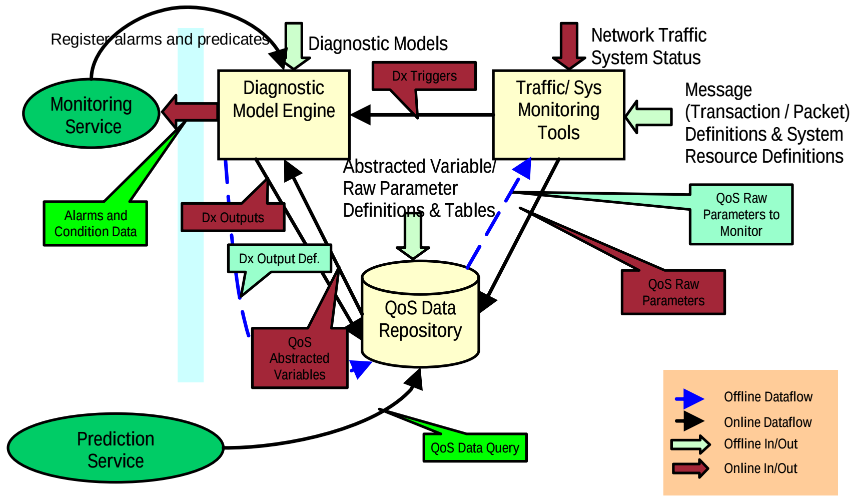


Figure 2 Interactions between Monitoring Service and Diagnostics Service.

Our Adaptation Service focuses on the service provider side resource management since QoS Manager does not have control over the client side resources. Therefore, the adaptation mechanisms are defined in the QoS manager's resource management policies using an XML-based language and are transparent to service consumers. In systems that some kind of resources may be negotiated or priced, the adaptation mechanism can also be defined in QoS contract. For example, if a publisher violates contract by sending messages at a more than agreed publishing rate, it needs to pay more on the service, and the unit price for the extra messages are often higher than normal.

The knowledge from monitoring and diagnostics services enables our system to support contract reuse, a feature particularly useful for mobile ad-hoc environments. As mobile ad hoc and wireless networks become more popular, integrated monitoring, diagnostics and adaptation services become more important for QoS management system. In the ad hoc network environment, clients can join and leave the network at any time. Similarly in wireless networks, clients may lose network connection accidentally when communication signal fades. As a result, the network topology changes frequently. In such environments, monitoring and diagnostics services can help detect the abrupt drop-off of the clients. After the drop-off is detected, the existing contract can be held for a certain period of time and be reused without renegotiation when the clients return. The major benefit of this contract reuse is that it reduces the load of unnecessary resource reallocation and improves the efficiency of resource management.

6 CASE STUDY

To verify and validate our QoS contract specification, establishment and monitoring framework, we created a few prototypes and conducted many experiments. This section describes a case study in details to illustrate the application of the proposed framework.

We selected a publish-and-subscribe based messaging system. In this system, there are two types of service consumers, namely the publishers and the subscribers.

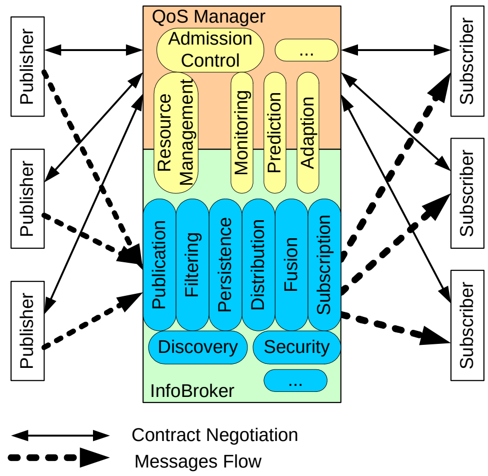


Figure 3 Integrate QoS Manager with a Publish Subscribe System

Publishers send messages to the service provider, namely InfoBroker, in certain channels provided by the InfoBroker. Subscribers subscribe to certain channels and receive all messages published to those channels. The publishers are totally decoupled from the subscribers through channels in the InfoBroker. The selected system is very powerful as it also supports message filtering, transformation, fusion and persistency and other functionalities. Unfortunately, this system as well as other similar commercial publish-and-subscribe based messaging systems (e.g., those based on Java Message Service) do not provide service differentiation among consumers. Indeed, they do not support the concept of service level management at all.

We integrated a QoS management prototype into this system to provide service level management, as illustrated in Figure 3. The QoS management prototype provides essential service level management functionalities including QoS contract specification, negotiation, establishment, operation, monitoring, diagnostics and adaptation. The integration efforts mainly include:

Customize QoS Specification Language

Identify applicable and appropriate QoS characterisitics for the publish-and-subscribe based messaging system. These include the message reliabilities, securities and transportation performance. An important observation is that Criticality is a high level QoS characteristic commonly used in mission-critical applications. Other QoS characteristics may be derived using Criticality based on the consumers' roles and domains. These QoS characteristics are used in specifying the QoS specification XML schema.

The QoS specification XML Schema needs to support the specificaiton of each identified QoS characteristics and potential constraints among them. A snippet of the schema is shown below.

```
<xs:schema …>
 <xs:element name="qos-requirement">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="criticality" …/>
    <xs:element ref="performance" …/>
    <xs:element ref="reliability" …/>
    <xs:element ref="security" …/> …
    <xs:element ref="constraints" …/>
    <xs:element ref="monitoring" …/>
   </xs:sequence> …
  </xs:complexType>
 </xs:element>
 <xs:element name="reliability">
  <xs:complexType>
   <xs:attribute name="guaranteed-delivery" …/>
   <xs:attribute name="duplication-elimination"…/>
   <xs:attribute name="message-ordering" …/>
  </xs:complexType>
 </xs:element>
…
</xs:schema>
```

Implement the Application Dependent Resource Management Code

Modify the resource management code in the existing publish-and-subscribe based messaging system so that critical resources are managed in accordance to QoS contracts. For example, the existing unlimited-sized single channel FIFO queue is replaced by a new multi-channel FIFO queue whose size is configurable and modifiable by the Resource Manager (see Figure 1).

Our QoS management prototype already supports generic management of resources in the abstract Resource Management service. However, it does not understand the actual resources used in the application domain and hence cannot create the resources. The concrete implementation needs to provide resource allocation and release mechanisms. On the other hand, our QoS management prototype does provide a library of common resources such as FIFO queues. Hence, the concrete implementation can reuse these resources for different purposes, e.g., a FIFO queue to hold messages for each channel.

Develop Monitoring, Diagnostics and Adaptation Strategies

Identify critical attributes of the QoS contracts that need to be monitored, and determine possible adaptation strategies for detected changes in these attributes. Sometimes, adaptaions may be triggered not because of the changes of directly monitored attributes, but due to changes in the interaction among the monitored attributes. The latter changes are detected by the Diagnostics Service based on configured diagnosis models. For example, in this case study, we also created diagnosis models to determine whether a client is slow based on the monitored network condition and other clients running on the same host.

Adaptations can be triggered when the changes result in a violation of the QoS contract, or is likely to result in a violation in the near future (e.g., reaching a warning threshold). The violation can be either on the InfoBroker side or the clients (publishers and subscribers) side. If the violation is on the InfoBroker side, adaptations are triggered to bring the attributes back to normal range (in the QoS Contracts). If the violation is on the client side, adaptations are triggered to degrade services according to SLAs.

For example, when the publishing speed is greater than the agreed QoS contract, the service provider may reduce the priority of the publisher and hence decrease the serving speed if there is resource contention. For another example, if a message payload is larger than the agreed QoS contract, the service provider may drop the message even though the message is guaranteed to be delivered according to the contract.

In addition to configuring the monitoring points and creating the diagnosis models, we also need to develop adaptation code according to the determined strategies. Our QoS management prototype provides generic code to register adaptation mechanims as plugins and trigger adaptations according to policy. To intergrate with the publish-and-subscribe messaging system, we need to implement the adaptation strategies as plugins. Unlike the generic Adaptation Service code, these plugins understands the application domain and can modify the internal logic of InfoBroker. For demonstration purpose, adaptation actions are displayed in an Admin window. Figure 4 gives an example.

Figure 4. An example screen dump of the adaptation messages.

Develop Policies

As shown in Figure 1 and discussed in Section 4, our QoS management architecture uses policies to support admission controls, resource management as well as monitoring and adaptations. In this case study, we created two types of policies in the XML-based policy languages interpreted by the Policy Manager in the QoS management architecture.

The first type of policies is for admission control. These policies determine the allowed QoS requirements at the application level using the QoS specification language. The following is a snippet of an example policy which assigns `1 second` response time for any `red` (criticality) publisher and agree on the delivery guarantee requirement.

```
<?xml version="1.0"?>
<qos-policy name="…" version="1"
            target="qos-requirement">
 <variable name="role"><path>…/@role</path></variable>
 …
 <node description="Publishing critical (red) message">
  <condition><function name="and">
   <function name="is">
    <varref name="role"/><constant>publisher</constant>
   </function>
   <function name="is">
    <varref name="criticality"/><constant>red</constant>
   </function>
  </function></condition>
  <copy source="/qos-message/profile"/>
  <create name="performance">
   <quote><response-time period="1" unit="second"/>…</quote>
   <copy source="/qos-message/performance/volume-rate"/>
  …</create>
  <create name="reliability">
   <copy source="/qos-message/reliability/delivery"/>
  …</create>
 …</node>
 <node description="Publishing yellow message">…</node>
…</qos-policy>
```

The second type of policies is for resource allocation, monitoring and adaptation. These policies determine how to allocate resources for the agree QoS contracts, which monitoring points need to be installed, and which adapatation mechanisms needed to be triggered in response to monitored changes. This type of policies depends on the exact resource types, supported monitoring points and adaptation mechanisms supported in the application. The following is a snippet of an example policy which assigns create a message queue resource (and others which are not shown here) whose length depends on the message payload size. It also installs an adaptation mechanism to be triggered when the payload size exceeds an upper bound and another adapation mechanism to be triggered when the payload size returns normal.

```
<?xml version="1.0"?>
<qos-policy    name="…"    version="1"    target="resource-
management">
  …
 <node description="Contract for the Publisher">
  <condition><function name="and">
   <function name="is">
    <varref name="role"/><constant>publisher</constant>
   </function>
   <function name="is">
    <varref name="criticality"/><constant>red</constant>
   </function>
  </function></condition>
  <create name="resources">
   <create name="message-queue">
    <create name="queue-size">
     <create name="target">
      <node>
       <condition><function name="lt">
        <varref name="pay-load"/><constant>5120</constant>
       </function></condition>
       <constant>100</constant>
      </node>
      <node>
       <condition><function name="lt">
        <varref name="pay-load"/><constant>10240</constant>
       </function></condition>
       <constant>50</constant>
      </node>
      … <!-- other conditions-->
      <node><constant>3</constant></node>
     </create> <!-- queue-size -->
    </create> <!—message-queue -->
    … <!-- other resources -->
   </create>  <!—resources -->
   <create name="monitoring-points">
    <create name="monitoring-point">
     <create name=”source”>
      <constant>receiver</constant>
     </create>
     <create name=”name”>
      <constant>payload-size</constant>
     </create>
     <create name=”facet”>
      <create name=”target”><varref name=”target”/></create>
      <create name="threshold">
       <create                  name=”name”><constant>upper-
bound</…></create>
       <create name=”value”><function name=”multiply”>
        <varref name=”target”/><constant>1.2</constant>
       </function></create>
       <create name="cross-up”>
        <create name=”action”><create name=”class”>
         <constant>….AdaptationChangeDeliveryGuarantee</…>
       </create></create></create>
```

```
         <create name="cross-down”>
          <create name=”action”><create name=”class”>
           <constant>….AdaptationRestoreDeliveryGuarantee</…>
         </create></create></create>
       …</create> <!-- other facet such as average -->
      …</create> <!-- other monitoring point -->
    </node> <!-- Publisher Contract -->
    … <!-- other clients -->
  </qos-policy>
```

Case Study Summary

The integrated system now offers service differentiation for different service consumers. Publishers and subscribers should first establish a QoS contract with the InfoBroker, and then send or receive messages according to the QoS contract. To accommodate existing legacy applications, a default QoS contract will be created by the modified InfoBroker if a publisher or subscriber does not explicitly negotiate a QoS contract with the InfoBroker. These default QoS contracts depend on the service consumers' identities and domains. This feature promotes customer acceptation and smoothes transition from existing system to the enhanced system.

A snippet of an example QoS contract including QoS parameters as a part of a consumer's SLA is shown below.

```
  <qos-requirement …>
    <performance>
      <volume-rate unit='second'>100</volume-rate>
      <pay-load volume='32' unit='kilobyte'/>
    </performance>
    <reliability>
      <guaranteed-delivery>yes</guaranteed-delivery>
    </reliability>
    <criticality>green</criticality>
  </qos-requirement>
```

The service level agreement in this study is relatively simple. It includes the consumer's profile (include identity and domain), the message channel, the message profile (including size and rate), and the QoS contract. Nevertheless, this study verified and demonstrated key components of the service level management including QoS contract specifications, negotiation, establishment, maintenance, revision, monitoring, diagnostics and adaptation. It also helped us gain insights in enhancing existing legacy systems to support service level agreements. Using a generic QoS management implementation, this enhancement effort requires some additional work that is specific to the legacy system. The additional work includes both modifying existing implementation for resource management and developing new components such as QoS characteristics, policies and adaptation mechanisms.

7 CONCLUSION

In this paper, we discussed QoS management to support service level management and described a QoS contract specification, establishment and monitoring framework. Our work focuses on a common understanding of QoS characteristics and their relationships between service providers and service consumers. Our QoS management architecture provides clean and reusable concepts and processes to facilitate QoS contract establishment and monitoring through contract negotiation, resource management, diagnostics and adaptation.

Our future work will focus on two areas. One is the derivation of service QoS characteristics from enterprise service modeling and analysis, in particular, the QoS aspects of services. The other area is the research and development of dynamic QoS-driven resource management algorithms for SLM.

8 ACKNOWLEDGMENT

The authors acknowledge Stephen Uczekaj for his tremendous support, and thank Casey K. Fung, Yichi C Pierce and Paul Z. Thunemann for their invaluable help. The authors also thank Klara Nahrstedt and Chui Sian Ong for their insights in this work. This paper is an extended version of a paper presented to the IEEE EDOC 2006 Advances in Quality of Service Management (AQuSerm) workshop.

REFERENCES

[1] International Organization for Standardization. *ISO/IEC JTC1/SC21 Working Draft for Open Distributed Processing - Reference Model - Quality of Service*. July 1997.

[2] International Organization for Standardization. *ISO/IEC International Standard 13236: Information Technology - Quality of Service: Framework.* First edition, Dec 1998.

[3] International Organization for Standardization. *ISO/IEC Technical Report 13243: Information Technology - Quality of Service: Guide to Methods and Mechanisms.* First edition, Nov 1999.

[4] OASIS Web Services Reliable Message TC. *WS-Reliability 1.1. OASIS Standard*, available at http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf. November 2004.

[5] Guijun Wang, Alice Chen, Changzhou Wang, Casey Fung and Stephen Uczekaj. "Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures". *Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, 2004.

[6] Changzhou Wang, Guijun Wang, Alice Chen and Haiqin Wang. "A Policy-Based Approach for QoS Specification and Enforcement in Distributed Service-Oriented Architecture". *The IEEE International Conference on Services Computing (SCC)*, 2005.

[7] Guijun Wang, Changzhou Wang, Alice Chen, Haiqin Wang, Casey Fung, Stephen Uczekaj, Yi-Liang Chen, Wayne Guthmiller and Joseph Lee. "Service level management using QoS monitoring, diagnostics, and adaptation for networked enterprise systems". *Ninth IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, 2005.

[8] Zheng Wang. *Internet QoS: Architectures & Mechanisms for Quality of Service*. Morgan Kaufmann Publishers, ISBN 1-55860-608-4, 2001.

[9] Klara Nahrstedt and Jonathan Smith. "The QoS Broker". *IEEE Multimedia Magazine*, Vol. 2, No 1, 1995.

[10] Jingwen Jin and Klara Nahrstedt. *Classification and Comparison of QoS Specification Languages for Distributed Multimedia Applications*. Technical Report UIUCDCS-R-2002-2302/UILU-ENG-2002-1745, Department of Computer Science, University of Illinois at Urbana-Champaign, November, 2002.

[11] Xiaohui Gu, Klara Nahrstedt, Wanghong Yuan, Duangdao Wichadakul, Dongyan Xu. *An XML-based Quality of Service Enabling Language for the Web*. Journal of Visual Language and Computing (JVLC), special issue on Multimedia Languages for the Web, vol. 13, num. 1, pp. 61-95, Academic Press, February, 2002.

[12] A. Keller and H. Ludwig. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". *Journal of Network and System Management*, Special Issue on E-Business Management, Vol. 11, No. 1, March 2003.

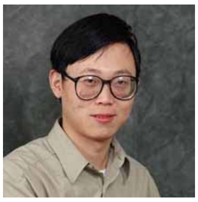
[13] A. Dan, D. Davis, etc. "Web Services on Demand: WSLA-driven Automated Management". *IBM Systems Journal*, Vol. 43, No. 1, 2004, pp. 136-158.

[14] V. Albaugh and H. Madduri. "The Utility Metering Service of the Universal Management Infrastructure". *IBM Systems Journal*, Vol. 43, No. 1, 2004, pp. 179-189.

[15] H. Jonkers, M. Iacob, M. Lankhorst and P. Straiting. "Integration and Analysis of Functional and Non-Functional Aspects in Model-Driven E-Service Development". *IEEE International Enterprise Distributed Object Computing (EDOC) conference*, Enschede, Netherlands, September 19-23, 2005.

[16] I. Poernomo, J. Jayaputera and H. Schmidt. "Timed Probabilistic Constraints over the Distributed Management Taskforce Common Information Model". *IEEE International Enterprise Distributed Object Computing (EDOC) conference*, Enschede, Netherlands, September 19-23, 2005.

About the authors



Changzhou Wang is an Advanced Computing Technologist in the Boeing Company. He received his Ph.D. in Information Technology from George Mason University, Fairfax, Virgina in 2000. His current research interest includes information management, data mining, and software quality of services. He can be reach by email under changzhou.wang@boeing.com

Guijun Wang is a Technical Fellow of the Boeing Company. He received his Ph.D. in Computer Science from the University of Kansas. His current research interests include service-oriented architecture for system of systems, quality of service management for network centric operations, enterprise distributed computing technologies and systems, systems and software engineering. In addition to research work, he has been an architect for several large scale distributed systems. He has been on the steering committee of the IEEE Enterprise Distributed Object Computing (EDOC) conferences since 2002. He was the Program Co-Chair, General Chair, and Panel Chair for EDOC 2000, 2001, 2002, respectively.

Haiqin Wang holds Ph.D and MS degrees from Intelligent Systems Program at the University of Pittsburgh. She also received MS degree in Pattern Recognition & Artificial Intelligence from the Institute of Automation at Chinese Academy of Sciences (1996) and BS degree in computer science from the University of Science & Technology of China (1992). She is currently working in the Adaptive Systems group of Boeing Phantom Works, the central research and development organization of The Boeing Company. Her research interests include Bayesian belief networks, uncertainty reasoning, sensitivity analysis, machine learning, data mining, and user interfaces to decision support systems.

Alice Chen is a Technical Fellow of the Boeing Company in the areas of Communications/Network and Information Assurance (IA) technologies. She holds a M.S. degree in Computer Science from Memphis University in 1975. Her recent research interests are the OSI upper layer issues such as embedding IA in Service Oriented Architecture, Middleware QoS, and policy-based processes flow control/collaboration. Coaching junior researchers and transferring research technologies to enterprise production environments are also her interests.

Rodolfo Santiago is an advanced computing technologist at Boeing Phantom Works' Mathematics and Computing Technology division. His research interests has been on distributed computing, networking and embedded systems. He has an MS degree in Computer Science from the George Washington University and a BS degree in Electronics and Communications Engineering from De La Salle

University.