Computing and Informatics, Vol. 21, 2002, 363–382

# G-QOSM: GRID SERVICE DISCOVERY USING QOS PROPERTIES

Rashid J. Al-Ali, Omer F. Rana, David W. Walker

*Department of Computer Science*
*Cardiff University, Wales, UK*
*e-mail:* {rashid, o.f.rana}@cs.cf.ac.uk

Sanjay Jha, Shaleeza Sohail

*Department of Computer Science and Engineering*
*UNSW, Sydney, Australia*

Revised manuscript received 13 November 2002

**Abstract.** We extend the service abstraction in the Open Grid Services Architecture [12] for Quality of Service (QoS) properties. The realization of QoS often requires mechanisms such as advance or on-demand reservation of resources, varying in type and implementation, and independently controlled and monitored. Foster et al. propose the GARA [10] architecture. The GARA library provides a restricted representation scheme for encoding resource properties and the associated monitoring of Service Level Agreements (SLAs). Our focus is on the application layer, whereby a given service may indicate the QoS properties it can offer, or where a service may search for other services based on particular QoS properties.

**Keywords:** QoS, service-oriented Grids, differentiated services, web services

## 1 INTRODUCTION AND RELATED WORK

Service Oriented Computing is often seen as a natural progression from component-based software development [20], and as a means of integrating different component development frameworks. A service in this context may be defined as a behaviour

that is provided by a component for use by any other component based on a network-addressable interface contract (generally identifying some capability provided by the service). A service stresses interoperability and may be dynamically discovered and used. According to [12], the service abstraction may be used to specify access to computational resources, storage resources, and networks, in a unified way. How the actual service is implemented is hidden from the user through the service interface — hence, a compute service may be implemented on a single or multi-processor machine — however, these details may not be directly exposed in the service contract. The granularity of a service can vary, and a service can be hosted on a single machine, or it may be distributed. The focus of this work is to allow Quality of Service (QoS) criteria to be specified as part of a service interface — to enable service selection based on QoS.

The system closest to our approach is Darwin [7] — a service-oriented resource management system capable of supporting/managing requests for complex network services with QoS support. A request is initiated by the user in the form of a task graph, and the resource manager subsequently locates suitable resources to perform the requested tasks to the optionally specified QoS requirements. The resource manager is responsible for creating a "hierarchical grouping" — a structure of the flows with their QoS specifications and the node IP addresses. The Globus Architecture for Reservation and Allocation (GARA) [10] addresses QoS at the level of facilitating and providing basic mechanisms for QoS support, namely resource configuration, discovery, selection, and allocation. This architecture is particularly aimed at using Globus services to support allocation of resources, and utilises specialised resource managers (such as a Diffserv manager) to support admission control and application adaptation at network edges. Current emphasis has been on supporting request authentication and authorisation. The hierarchical service discovery approach adopted by [13, 26] is also particularly relevant in this context — as the common attributes of service providers and users can be determined across service domains. QoS feedback and the dynamic caching of service advertisements can also be supported through such a hierarchical organisation, and used by an application to adapt its behaviour. This enables the identification of 'QoS-similar' domains — to support advert propagation over time. GARA does not support these request-caching and inter-domain similarity derivation mechanisms. Specifying QoS requirements in terms of a 'policy' involves provision of an object-oriented, declarative language for specifying management and security policies [8], for instance. Policy specification allows various entities in the network to specify constraints on flows. Existing work on such policy languages has been targeted to abstract network components — and of these, few mappings to real network components exist. COPS (Common Open Policy Service) [18] provides a a simple query and response protocol that can be used to exchange policy information between a policy server (Policy Decision Point) and its clients (Policy Enforcement Points). This protocol, however, necessitates the existence of a single policy server in each administrative domain — and therefore it can be encapsulated into the NRM module in our system. External components — such as RSVP capable routers — can then query this server

to verify a policy. Our approach also explores the description of policy (based on constraints) in the context of software services. Czajkowski et al. [6] define a model and protocol for negotiating access to resources in distributed systems such as Grids. An SLA mechanism is introduced that may be used to describe application activities ("tasks"), resources with QoS information and a SLA to bind applications with resources. In [11] Gu et al. propose a model for QoS-aware service aggregation in Peer-to-Peer (P2P) systems. Tuecke et al. [22] introduce the concept of `service data`, which refers to descriptive information about a Grid service instance. Our extensions to the Grid service description (with `service-QoS` properties) builds on these `service data` extensions.

## 2 THE ARCHITECTURE OF THE G-QOSM FRAMEWORK AND ITS SCOPE

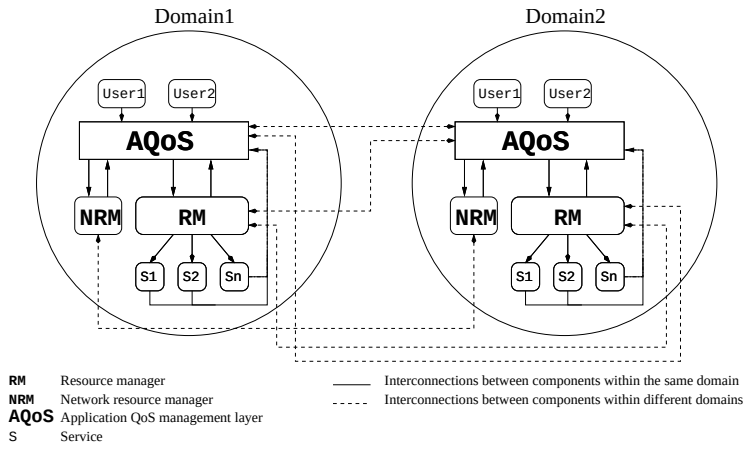



Fig. 1. The G-QoSM architecture

In the context of our framework, QoS may be characterised at three levels: (1) application level QoS, (2) middleware level QoS, and (3) network level QoS. For application QoS it is important to view both (1) user perception, and (2) application perception. The first is relevant when a user is accessing a remote service and the response that the user is likely to get. The second is related to when a service is being offered to multiple users, and the response (access times) that the service is likely to see from the users. This is particularly relevant in the context of applications which require data to be obtained from a user in order to proceed — such as in interactive visualisation applications, where users must send data to a remote service in order to alter the simulation control parameters. Application QoS also relates to the number of concurrent users that can be allowed to access a service, and therefore also relates to application security issues such as authentication mechanisms. The middleware

QoS is of particular importance as our work is based on existing middleware, namely Globus, and may use third party software which does not have middleware QoS awareness. This QoS model addresses large scale distributed computing — within which single or multiple services interact to exchange control parameters and data for computational steering applications, for instance, and therefore the responses have to be delivered within limited time frame to users/services — hence the need for network QoS.

The framework uses a hierarchical discovery scheme, and is therefore similar to work in [13, 26]. The hierarchy in our framework, however, does not represent the replication of similar services at high levels, but the provision of specialised services which are shared between domains. Different parameters need to be specified and monitored at each of the three levels above. The parameters at the network level are generally better understood than at the application level — and work undertaken at the application level has generally been in the context of very specialised services — such as a multimedia service altering its transmission quality based on network metrics such as latency and bandwidth [16]. The primary activity supported through the framework is the discovery of suitable services, managed locally or remotely, based on QoS criteria. Service execution and management, once a suitable service has been discovered, are not supported, and rely on the existence of systems such as Globus. The results of executing a service are recorded and used to aid subsequent service discovery operations.

Hence, the Grid QoS Management (G-QoSM) framework is aimed at supporting service discovery foremost, and monitoring services to ensure that the QoS requirements for these services are being met in the context of a distributed computing environment such as the computational Grid — as illustrated in Figure 1. It consists of the following modules:

- Resource Manager (RM): A single RM exists within a given administrative domain. A domain can be defined through an IP mask or a Globus domain, for instance, and contains a set of services over which the RM has administrative and configuration control. A RM is considered in this context as a combination of Globus Resource Allocation Manager (GRAM) and a Universal Description and Discovery Integration (UDDI) registry. Globus is used to host the service and to create an execution environment with specific QoS specifications; UDDI on the other hand is used as a registry service, to record service capability and QoS provisions.
- Network Resource Manager (NRM): The NRM is conceptually a Bandwidth Broker (BB) — the concept of a BB can be found in [21]. NRM manages the QoS resources within a given domain based on the SLA that have been agreed upon in that domain. The NRM is also responsible for managing interdomain communication, with the NRMs in neighbouring domains, with a view to coordinate SLAs across the domain boundaries. It will also communicate with local enforcers to determine the state of the network as well as configure the network. The NRM also gathers and monitors the state of resources within its domain

and on the edges of the domain (edge routers connected to and from adjacent domains). The NRM accounts for the ability of the entire network to deliver a particular policy request.

In our framework a single NRM exists within an administrative domain, and is responsible for monitoring network parameters, and supports admission control. For monitoring purposes, the measurements taken by the NRM are based on those outlined by Lowekamp and Tierney [15].

- Application QoS (AQoS) manager: The AQoS manager interacts with all the other components in the G-QoSM to find services with specific QoS requirements, coordinates with the resource managers to allocate resources for subsequent service execution and finally monitors the service execution. Further, the AQoS is also responsible for aggregating metrics across a series of operations needed to run an application. Once a service to run has been discovered, the next step involves initiating execution of the service by sending the control parameters, locating and transferring the data, invoking the executable corresponding to the service, and subsequently transferring the results (achieved via Globus). The response time for each of these operations is measured and aggregated by the AQoS manager. This is recorded by the AQoS manager as a three-tuple, $(RM_j, S_i, m)$, where $RM_j$ corresponds to the broker within the $j^{th}$ remote domain, $S_i$ represents a service within that domain, and $m$ is a real number corresponding to the total time required to transfer data and initiate execution of the remote service. The AQoS manager may either assign the same value for $m$ to all services within a particular domain, or it may distinguish between services.
- Service: The IETF has proposed two major architectures to support QoS: (1) Intserv [4] provides QoS guarantees on per-flow basis using RSVP [5], (2) Diffserv enables scalability across large networks but may not be able to support per-flow QoS guarantees [3]. A hybrid architecture that takes Intserv at the edge of the network (as state explosion is unlikely to happen at the edge), and Diffserv at the core, called "Intserv over Diffserv" (IS/DS), can address the scalability problem associated with either of these approaches. Because the hosts are still connected to Intserv at the edge, IS/DS can support end-to-end per-flow QoS. As IS over DS is emerging as an architecture of choice, the G-QoSM utilises this service model. Hosts connected to Intserv at the edge use one of the three classes: (1) Guaranteed-service [17], (2) Controlled-load [25]; or (3) Best-effort-service. These services are invoked by individual flows. The Guaranteed-service delivers QoS based on pre-defined constraints identified by a user, and agreed upon by the provider in a SLA (delay bounds are stringent). In this type of service, the QoS parameters are enforced, monitored, and the service provider is committed to deliver the service with such specifications. In the Controlled-load, the user states the QoS requirements, the brokering service finds the most suitable resources to execute the service, and the service is eventually executed. However, the main difference is that the QoS requirements are a less stringent

(i.e. the service provider is not committed to deliver the service as per the user specifications, and the QoS parameters are not enforced). There are two occasions where the Controlled-load service might be appropriate; one is when the brokering service fails to find reliable resources to undertake the execution of the service and the user still wants to attempt executing the service with whatever resources are available. Another occasion is when the user and the service provider could not reach a mutual agreement on the SLA contract (all constraints identified by the user, or the service provider, cannot be satisfied). The third type of QoS level is Best-effort-service, and occurs when there is neither a SLA involved nor the user specifies QoS requirements. The brokering service now has full control to configure the execution environment based on the available resources. This type of service might be suitable for users who do not know the exact configuration or the QoS requirements needed for their desired service.

### 2.1 Interaction with the NRM

The AQoS interacts with a network resource manager (NRM), where the NRM is responsible for provisioning of network resources, monitoring and supporting admission control (based on a domain specific policy). Once an appropriate service has been identified, the AQoS communicates with the NRM to determine a communication path between the source and the destination. This utilises the QoS range in the service advert from the provider. Typically the AQoS will work out the traffic characteristics (such as RSVP TSpec [5] parameters described using token bucket). As RSVP is a receiver-oriented protocol, it must be customised to make it sender-oriented whereby the NRM will act as a proxy sender on behalf of the user, and the service (or NRM of Service domain) will act as receiver of the RSVP messages. This assumption is based on the fact that RM has already performed match making (resource requirements for a service and user capability).

We describe how this communication takes place in an IS over DS architecture below. For simplicity we take a single point-to-point connection. The DS domain is RSVP unaware. We describe the process of establishing end-to-end QoS through the following steps:

- The NRM generates RSVP PATH message describing the user's traffic profile;
- The PATH message is processed (path state is installed) by all routers in the IS domain and it is forwarded to the DS domain towards the Service;
- Routers in the DS domain ignore the PATH message (no path state processing) and forward it to the IS domain towards the Service;
- Finally, Service (or NRM in the service domain) receives the PATH message and generates a RESV message (subject to authentication/authorisation for charging etc);

- IS domain carries the RESV message toward the NRM of service user; if any router has insufficient resources to support this reservation, it will generate an error message and will reject the request;
- When the RESV message reaches an edge router, the NRM of the domain checks the service level specification (SLA) of the DS interface to see if there is enough unused resources in the agreement to support this reservation request. If not, the request is rejected, otherwise the RESV is forwarded toward the AQoS. In order to support IS over DS model, a new Diffserv code point (DSCP) is also allocated for the session;
- Receipt of the RESV message by AQoS's RSVP process indicates that the resource reservation has been successful.

Furthermore, NRM is the management entity that has a complete and up-to-date topology of the domain, and maintains a record of network resources within it. In order to provide strict QoS guarantees the NRM monitors network resources periodically and then performs off-line analysis to calculate different attributes of Per Domain Behaviour (PDB). PDB is the edge-to-edge treatment that traffic receives in a Diffserv domain. PDB depends upon the service type as well as the load conditions and some domain specific parameters like domain topology, links used to transfer traffic etc. The sum of same type of PDB parameters of all the domains from which the flow will pass gives the end-to-end QoS parameters for the particular flow. The attributes that can be part of the PDB include delay, packet loss and throughput.

### 2.2 Interaction with the AQoS Manager

The RM interacts with the AQoS manager for aggregating metrics across the multiple sessions required to execute a remote service. The AQoS manager also maintains aggregate metrics corresponding to a particular remote RM and service, and uses this as a means to subsequently select a remote service. The service requester also interacts with the AQoS for requesting services with QoS properties and negotiating SLAs. Figure 2 shows a sequence diagram between the AQoS and other G-QoSM components — the legends on the figure indicate the various stages involved.

### 2.3 Interaction with the Resource Manager

In the context of our G-QoSM framework, the RM is meant to integrate Globus GRAM and the UDDI registry service. The AQoS interacts with the RM to (i) pass requests for services to be discovered based on QoS properties and (ii) create a service execution environment with regard to the middleware QoS as described in Section 3.2. The RM can further interact with a remote RM to find services with similar capacities or to allocate resources for service execution as both RMs are part of the service-oriented Grid.

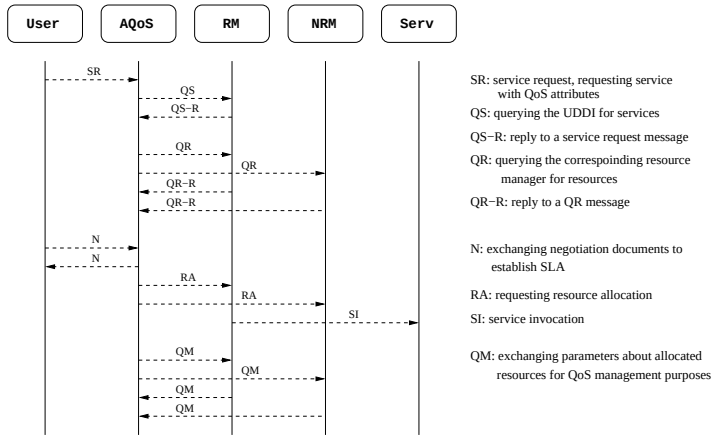



Fig. 2. Sequence diagram between the AQoS and other G-QoSM components

## 3 SPECIFICATION OF QOS CRITERIA

QoS parameters are defined with respect to the three levels defined in Section 2. QoS parameters also differ depending on whether they are measured and administered by the NRM, the AQoS manager or the middleware such as Globus.

### 3.1 Application QoS

We characterise application QoS (as managed by the AQoS manager) as comprising the following operations: (1) Send configuration (control) information to a remote service, (2) transfer data to the remote service, (3) execute the remote service, and (4) transfer data from the remote service to the requesting service. These operations may need to be replicated, in the context of computational steering applications (for instance), for a set of interactions between two services. Associated with these operations are times which can be measured and recorded by the AQoS manager, and primarily measure the interaction between two services.

### 3.2 Middleware QoS

We assume that middleware such as Globus will manage the creation of an execution environment for a service, along with subsequent management and control. Moreover, we provide here QoS metrics for describing the execution environment and we do not guarantee QoS enforcement; it is totally up to the middleware to provide such a guarantee. The Libra project [19] aims to explore middleware QoS in particular developing a QoS-based scheduler for resource management on a homogeneous cluster.

### 3.3 Network QoS

The network QoS in our framework is primarily concerned with managing bandwidth. The network resource manager (NRM) takes raw network measurements and generates statistics about the network link. The measurements taken by the NRM are based on those outlined in [15] and consist of the following metrics: (1) Capacity: the maximum bandwidth a path can provide to an application when there is no competing traffic load (cross traffic), (2) Availability: the maximum throughput that the path can provide to an application, given the path's current cross traffic load, and (3) Utilisation: the aggregate bandwidth currently used by all applications on that path.

## 4 SPECIFICATION OF SERVICE LEVEL AGREEMENTS

Service Level Agreements (SLAs) are used as a means of managing and monitoring QoS attributes at the three levels (outlined in Section 2, and to enforce contracts. Once an AQoS has discovered a RM or a remote RM (or RM-group) which contains the required service, a SLA needs to be established between the service provider and the service requester. If the requesting service requires "Guaranteed-service" QoS, then the SLA consists of exact constraints which must be met by the AQoS, RM and NRM. If a "Controlled-load" QoS is being requested, then the QoS requirements are more relaxed and the parameters are specified as a range constraint — allowing the AQoS and the corresponding RM and NRM to find bounds on QoS parameters which can match the required constraints. All attributes of the SLA are measurable and quantifiable — and measured by the corresponding resource manager or the AQoS manager, Figure 3 shows a sample SLA specification document. It is important to remember that the AQoS maps these SLA parameters to the corresponding resource managers; for example in the case of network QoS specifications, TSpec parameters such as token rate, token bucket size and peak rate will be generated based on the specified SLA parameters and passed to the network resource manager (NRM). We believe that in order to establish an effective SLA mechanism, three essential requirements should exist in the context of the G-QoSM framework:

1. A management entity should have a comprehensive view of the existing infrastructure and all the capability that can be provided by the resource managers.
2. A contract-based agreement should include the QoS parameters and the ability to negotiate its parameters.
3. A mechanism is needed to verify and monitor the agreement contract and to enforce its elements to meet the overall contract requirements.

Our AQoS has the ability to provide these three requirements. It consults the NRM for network QoS, and the RM to gather information about the existing infrastructure and its capability. It has the capability to initiate negotiation between two services.

```
<Service_SLA>
    <Service_Information>
        <Service_Name> MathCompService </Service_Name>
        <Service_Description>
         Scalar and Matrices Comp.
        </Service_Description>
        <Service_URL_Address> http://localhost/Services/MyMathService
        </Service_URL_Address>
        <Service_Cost> 52 </Service_Cost>
        </Service_Information>
    <User_Information>
        <Service_Requester_Name> Application DffEQ
        </Service_Requester_Name>
        <Service_Req_IP_Address> 192.200.168.33
        </Service_Req_IP_Address>
    <User_Information/>
    <QoS_Class> Guaranteed-Service QoS </QoS_Class>
    <Temporal_QoS>
        <Start_Time> 20/6/2002 11:05 </Start_Time>
        <End_Time> 20/6/2002 13:18 </End_Time>
    <Temporal_QoS/>
    <Appl_QoS>
        <Availability> > 99.5% </Availability>
        <Accessibility> > 90% </Accessibility>
        <Reliability>  High </Reliability>
        <Security> Normal </Security>
    </Appl_QoS>
    <Middleware_QoS>
        <Node_IP_Address> 135.200.50.101 </Node_IP_Address>
        <CPU_Count> 4 </CPU_Count>
        <Real_Storage> 240 MB </Real_Storage>
    <Disk_Storage> 40 MB </Disk_Storage>
    </Middleware_QoS>
    <Network_QoS>
        <Source_IP> 192.200.168.33 </Source_IP>
        <Dest_IP> 135.200.50.101 </Dest_IP>
        <Throughput> 256 Kbps </Throughput>
        <Packet_Loss> 10% </Packet_Loss>
    </Network_QoS>
</Service_SLA>
```

Fig. 3. SLA specification

It also monitors and enforces the QoS parameters as per the agreed contract with regard to the QoS class.

### 4.1 Contract-Based Agreement

A contract-based agreement forms the main constituent of the SLA. The content of the contract covers mainly the three levels of the QoS. Furthermore, our contract is similar to the work in [2]; however, their contract is in the context of multiple control domains providing services such as E-commerce, web hosting, etc. The contract elements are measurable and quantifiable, and have interdependencies among them, meaning that the contract is atomic in its nature and all elements must hold in order for the desired service to function properly. Furthermore, the set of contract elements must be consistent and verifiable, meaning that there must not be dependencies between the contract elements in such a way that if one is TRUE another one is forced to be FALSE. Verifiability requires a programming tool to evaluate the contract elements at any given time. Our G-QoSM framework supports mechanisms to examine contract consistency and contract verification through its "AQoS" component.

We define a contract $C$ in terms of assertions $A$, where $A$ is a set of QoS attributes that are required for service delivery and agreed on with the service requester. Each service requester must specify its QoS requirements for the AQoS manager. Therefore, we define $v$ as the QoS vector consisting of QoS properties, so that:

$$v = v_{(Application-QoS)} \cup v_{(middleware-QoS)} \cup v_{(network-QoS)}.$$

Similarly, depending on the QoS class being requested (Guaranteed or Controlled-load), we define the assertion $A$ to be a set of relationships that exist over $v$, as $R(v)$. The set of relations $R$ primarily specifies the contract agreement between the service requester and provider. The relations are expressed in statements which contain logical predicates, which are measurable, and the predicates are composed further of variables and logical operators. Moreover, the assertion $A = R(v)$ must be evaluated at any given time to be either TRUE or FALSE depending on the attributes described in the relation. In the case of a Guaranteed QoS, for instance, it consists of assertions which must evaluate to TRUE. In the case of "Controlled-load" QoS it consists of assertion constraints which must hold within the specified bounds.

For example; a Guaranteed-QoS request may arrive for a Grid service with the following QoS attributes: service accessibility of more than 99 %, 64 MB of main memory, with 4 parallel CPUs, and 2Mbps bandwidth. These QoS specifications can be expressed as an assertion $A$ of relation $R(v)$ as follows:

$$A = \{ServiceAccess > 99\,\%,$$

$$A = Memory = 64\,MB \wedge CPU = 4, Bandwidth = 2\,Mbps\}\,.$$

When a service request is received by a remote RM, it needs to verify if the required QoS criteria can be met. This is achieved by confirming the request with

parameters recorded by the NRM, and with summary statistics maintained by the AQoS manager for the remote service. If the request for a given service has not been obtained before, parameters from the NRM corresponding to the entire traffic for the domain are used. As additional requests to a service are obtained, the response time, execution time and data transfer time are recorded. These values constitute the aggregate times for all requests which have been received by the service — and are maintained by the AQoS manager.

### 4.2 Service Budget Estimation

We introduce a service cost factor as a QoS criteria; the user is expected to specify the maximum budget that s/he is willing to spend, or the budget range to be associated with the service QoS specification. The user is also expected to specify the budget level of tolerance with respect to other QoS criteria; in this way we can utilise this tolerance information to make a number of tradeoffs such as service cost versus service accuracy. For instance, a user might request a graphics service that is network-QoS sensitive, with a certain budget tolerance level that indicates the user is willing to accept a lower level QoS of network resource but not to exceed the specified budget.

Service software cost is associated with the extended service interface description in the Web Services Description Language (WSDL) document along with other QoS attributes that the service is sensitive to, and the minimum set of QoS attributes that are associated with the service. The total service cost is the aggregation of all costs of the various resources associated with the service plus the service license/usage cost. The resource costs include the network-QoS cost, computation-QoS cost, and the data transfer cost. We estimate the total cost of a service as follows; every service has QoS properties $P(s)$, so that:

$$P(s) = P_{appl-QoS}(s) \cup P_{comp-QoS}(s) \cup P_{storage-QoS}(s) \cup P_{net-QoS}(s).$$

Every set of QoS properties such as $P_{appl-QoS}(s)$, $P_{net-QoS}(s)$, etc. has resource QoS and usage costs associated with it. The AQoS consults the corresponding resource manager, such as the NRM, for the usage cost in the case of network resources, whereas in the case of middleware resources the AQoS has its own mechanism to compute the resources budget that works in coordination with Globus. Therefore, the total service cost $C(s)$ is:

$$C(s) = s_{software-cost} + \sum_{i=1}^{n} C(Ri_{QoS}),$$

where $C(Ri_{QoS})$ is the cost of resource QoS $(i)$, and $(i)$ is the $i$th resource associated with the given service $s$.

## 5 EXTENDING WSDL & UDDI TO INCLUDE QOS PROPERTIES

Having defined the QoS matrix for Grid services, we now need a mechanism for users to express their requirements with regard to QoS criteria, for the system to recognise these requirements, and also for the service providers to advertise their services with QoS capabilities. Further, the system should be able to search for services on the basis of QoS requirements. Therefore, to facilitate this feature, firstly we need to provide the users with an environment enabling them to state their QoS requirements so the system can capture them; secondly we need to make sure that the matchmaking broker can indeed discover services based on such QoS attributes.

We associate QoS criteria with the service interface description, and include service software usage cost and all the QoS attributes that the corresponding service is sensitive to. The service software usage cost is the cost of the actual software provided by the service provider during service interface creation. It is worth mentioning that this cost is distinct from the total service cost and represented by the variable $s_{software-cost}$ in the cost equation derived in Section 4.2. The other QoS attributes that should be included in the service interface are the QoS that the service is sensitive to, with the minimum/recommended QoS levels necessary to run the service in an acceptable form; similarly, this information should be supplied by the service provider. This extension enhances the matchmaking process and subsequently leads to better matches. We have therefore provided means for the service requesters, who are concerned about the QoS and budget associations with particular services.

The OGSA specification for service interface definition documents [22], which defines the elements tags and their grammars in WSDL, does not include any tags for QoS provisions. Therefore, we suggest the incorporation into the main *definitions* element tag a new sub-element tag called *QoS* with various QoS attributes, as in the following WSDL service interface definition document:

In this way, service providers will be able to define their services using service interface definition documents along with their QoS capabilities, and subsequently services would be registered with a registry service such as the UDDI. Having considered the technologies used by OGSA, namely WSDL, Simple Object Access Protocol (SOAP) [24] and Universal Description Discovery and Integration (UDDI) [23], it can be said that the UDDI is the primary registry and service discovery engine, although it does not provide the support to discover Web services based on QoS criteria. Nevertheless, it supports service discovery based on other criteria, which include the following five XML data-structures, each of which contains a number of data fields that serve either a business or technical descriptive purpose: `businessEntity`, `tModel`, `businessService`, `bindingTemplate`, and `publishAssertion`. These XML data structures are generated on the basis of both WSDL documents, the service interface definition and the service implementation; therefore, in order for our proposed element tag *QoS* to be recognised, and hence searched, QoS attributes need to be associated via a new category

```
<?xml version="1.0" encoding="UTF-8"?> <wsdl:definitions
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    ...
targetNamespace="http://MyService-Interface">
    <wsdl:messagename="printNameResponse">
    </wsdl:message>
        ...
    <QoS>
        <service_cost> 5 </service_cost>
        <network_bandwidth> 256K </network_bandwith>
        <memory> 48MB </memory>
        ...
    </QoS>
    ...
</wsdl:definitions>
```

called *serviceQoS*, along with the QoS-related data fields. Furthermore, this allows search of the *serviceQoS* category, which would result in finding all the services with these specified QoS properties. With these extensions to the service interface definition (in the WSDL-based document and the UDDI categorization), service providers will be able to advertise their services with QoS attributes, users will be able to state their QoS requirements for services, and the matchmaking broker will be able to match queries and services based on QoS properties.

We utilise the UDDI4J APIs and the UDDI-M APIs [9] to demonstrate this idea in a prototype system using the IBM Web Services Toolkit (WSTK) [14]. This is discussed in more detail in Section 6 (Implementation). UDDI-M is an extension to the core registry functions of UDDI — and provides additional interfaces for handling service leasing, service search on keyword attributes, and provides additional extensions for searching the registry based on range based attributes. This is achieved by providing an additional query management layer over the core UDDI implementation.

## 6 IMPLEMENTATION

A prototype of the G-QoSM framework has been implemented to demonstrate the following ideas; i) extending WSDL and UDDI to include QoS provisions in the service interface definition and showing how this extension leads to the enhancement of the service discovery process; ii) handling the network QoS specifications, by showing how the AQoS manager requests, allocates and monitors network QoS-aware resources through interaction with the network NRM to provide network QoS guarantees.

### 6.1 UDDI and WSDL Extension

We use the IBM Web Services Toolkit (WSTK) version 3.1 that includes UDDI, Axis server, and several other tools and development products to create, publish, find and bind web services. We also use the JDK version 1.3 as the development technology for our implementation. We use the `Java2WSDL` tool in the IBM WSTK to create WSDL service interface and implementation from Java classes. This interface definition is subsequently extended with the QoS properties defined previously, using a Java based tool. The QoS information is assumed to be specified by the service provider. These attributes are: i) the service software cost; ii) the QoS attributes that the service is sensitive to with their minimum and recommended values and any other constraints the service provider may think it necessary to advertise.

After associating the service provider-supplied QoS attributes with the service interface definition, the service interface and implementation documents are published and the service then becomes registered with UDDI; hence our tool publishes the service and associates its QoS properties with the proposed *serviceQoS* category. The UDDI APIs [23] are utilised to develop a search mechanism that searches UDDI for services based on QoS attributes, such as the cost of service; furthermore this search mechanism also reveals to the service requester information about the minimum and recommended QoS levels required in order to execute the service in an acceptable form based on the service provider-supplied QoS specifications and constraints.

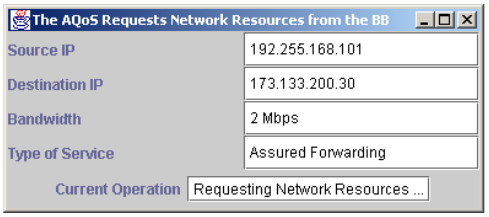



Fig. 4. The AQoS requests network resources

We demonstrate this idea by populating the UDDI with a number of services and various QoS attributes; our prototype provides tools that takes in two WSDL-based documents: i) service interface definition, and ii) service implementation and publishes the service in the UDDI. Furthermore, the tool helps service providers include QoS attributes in the service interface definition document and associates the attributes with the newly defined `<QoS>` element tag. On execution, the tool takes requests from users and searches for services with particular QoS specifications. The result is a list of services with similar QoS properties. Figures 4, 5 and 6 show screen shots of our prototype that illustrate the process whereby the AQoS requests network resources with specific QoS levels. In Figure 4 the AQoS queries for avail-

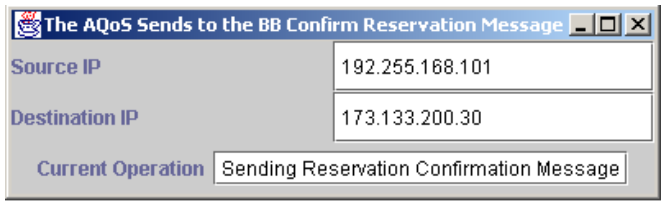



Fig. 5. The AQoS confirms resources reservation

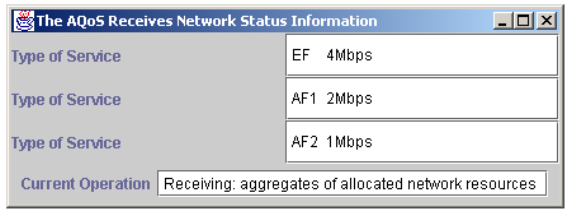



Fig. 6. A state information about allocated network resources

able resources, in Figure 5 the AQoS confirms the reservation for previously queried resources and in Figure 6 the AQoS displays the result of the requested state information about the allocated resources for QoS monitoring purposes. The result is aggregate of allocated network resources including assured forwarding (AF) services and expedited forwarding (EF) services.

## 7 CONCLUSION

A framework for QoS-based service management in service-oriented Grids is described. The main focus of this framework is to provide a means for the service requesters to search for services based on QoS criteria in computational Grids, to provide QoS guarantees for service execution and to enforce these guarantees by executing service level agreements. We provide extensions to UDDI to incorporate QoS definitions in WSDL-based documents, in the context of computational Grids — and based on `service data` extensions, and we show how our framework treats requests for resources with QoS specifications.

This work builds on the Open Grid Services Infrastructure (OGSI) being developed within the Global Grid Forum (GGF) — and will involve close interaction with the Grid Resource Allocation Agreement Protocol (GRAAP) working group. The framework outlined supports service discovery with QoS properties. The framework also enables a user to specify their QoS requirements, which are then checked against both network and computation/data resources using specialised management modu-

les. These modules interact by either reserving resources or undertaking allocation based on a budget.

**Acknowledgement**

We would like to acknowledge the implementations work undertaken by Ali Shaikh-Ali as part of his honours project — using UDDI and WSDL provided with the Web Services Toolkit (from IBM). We also would like to acknowledge Jonathan Giddy's (from the Welsh E-Science Centre) comments on utilising Globus in the G-QoSM framework.

## REFERENCES

[1] Argonne National Laboratory The Globus Project. See Web site at: `http://www.globus.org/`, last visited: April 2002.

[2] Bhoj, P.—Singhal, S.—Chutani, S.: SLA Management in Federated Environments. Technical Report HPL-98-203, Internet Systems and Applications Laboratory, HP Labs, Palo-Alto, CA 94034, USA.

[3] Blake S. et al.: An Architecture for Differentiated Service. Internet RFC 2475, 1998.

[4] Braden, R.—Clark, D.—Shenker, S.: Integrated Services in the Internet Architecture: an Overview. Request for Comments (Informational) RFC 1633, Internet Engineering Task Force, June 1994.

[5] Braden, R.—Zhang, L.—Berson, S.—Herzog, S.—Jamin, S.: Resource Reservation Protocol (RSVP) — Version 1 Functional Specification. RFC 2205, Internet Engineering Task Force, November 1997.

[6] Czajkowski, K.—Foster, I.—Kesselman, C.—Sander, V.—Tuecke, S.: SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. 8th Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh, Scotland, July 2002.

[7] Chandra, P.—Chu, Y.-H.—Fisher, A.—Gao, J.—Kosak, C.—Eugene Ng, T. S.—Steenkiste, P.—Takahashi, E.—Zhang, H.: Darwin: Customisable Resource Management for Value-Added Network Services. IEEE Network, Vol. 15, 2001, No. 1.

[8] Damianou, N.—Dulay, N.—Lupu, E.—Sloman, M.: The Ponder Policy Specification Language. Proceedings of workshop on Policies for Distributed Systems and Networks, Bristol, UK, January 2001.

[9] Dialani, V.: UDDI-M Version 1.0 API Specification. University of Southampton, UK, 2002.

[10] Foster, I.—Kesselman, C.—Lee, C.—Lindell, B.—Nahrstedt, K.—Roy, A.: A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation. Proceedings of the International Workshop on QoS, pp. 27–36, 1999.

[11] Gu, X.—Nahrstedt, K.: A Scalable QoS-Aware Service Aggregation Model for Peer-to-Peer Computing Grids. Proceeding of the HPDC11, 2002.

[12] Foster, I.—Kesselman, C.—Nick, J. F.—Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Downloadable as: http://www.globus.org/research/papers/ogsa.pdf, 2002.

[13] Haas, R.—Droz, P.—Stiller, B.: A Hierarchical Mechanism for the Scalable Deployment of Services over Large Programmable and Heterogeneous Networks. In Proceedings of ICC 2001, Helsinki, June 2001.

[14] IBM: Alpha Works Web Site, http://www.alphaworks.ibm.com/tech/webservicestoolkit/.

[15] Lowekamp, B.—Tierney, B.: Network Metrics for Grid Applications and Services. GWD-C, Network Measurements Working Group at GGF. See Web site at: http://www-didc.lbl.gov/NMWG/, 2002.

[16] Nakao, A.—Peterson, L.—Bavier, A.: Constructing End-to-End Paths for Playing Media Objects. Proceedings of IEEE Open Architectures and Network Programming, 2001.

[17] Shenker, S.—Partridge, C.—Guerin, R.: Specification of Guaranteed Quality of Service. Request for Comments (Standard Track) RFC 2212, Internet Engineering Task Force, September 1997.

[18] Durham, D. et al.: The COPS (Common Open Policy Service) Protocol. Internet RFC 2748, 2000.

[19] Sherwani, J.—Ali, N.—Lotia, N.—Hayat, Z.—Buyya, R.: Libra: An Economy driven Job Scheduling System for Clusters. Technical Report, The University of Melbourne, July 2002. Available from: http://www.gridbus.org/.

[20] Stevens, M.: Service-Oriented Architecture Introduction, Part 1. See Web site at: http://softwaredev.earthweb.com/microsoft/article/0,,10720_1010451_1,00.html.

[21] Teitelbaum, B.—Hares, S.—Dunn, L.—Neilson, R.—Narayan, R. V.—Reichmeyer, F.: Internet2 QBone: Building a Testbed for Differentiated Services. IEEE Network, Vol. 5, 1999, No. 13, pp. 8–16.

[22] Tuecke, S.—Czajkowski, K.—Foster, I.—Frey, J.—Graham, S.—Kesselman, C.: Grid Service Specification. Argonne National Laboratory, Argonne, IL. Draft3 (7/17/2002).

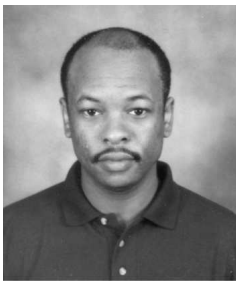
[23] Universal Description, Discovery and Integration of Bunsiness for the Web — Specifications. http://www.uddi.org/specification.html.

[24] W3C: Simple Object Access Protocol (SOAP). See Web site at: http://www.w3.org/TR/SOAP/.

[25] Wroclawski, J.: Specification of the Controlled-Load Network Element Service. Request for Comments (Standard Track) RFC 2211, Internet Engineering Task Force, September 1997.

[26] Xu, D.—Nahrstedt, K.—Wichadakul, D.: QoS-Aware Discovery of Wide-Area Distributed services. Department of Computer Science, University of Illinois at Urbana-Champaign, {d-xu,klara,wichadak}@cs.uiuc.edu, Technical Report UIUCDCS-R-2000-2189, November 2000.

[27] Yemini, Y.—Goldszmidt, G.—Yemini, S.: Network Management by Delegation. Proceedings of Intl. Symposium on Integrated Network Management, 1991.



**Rashid J. Al-Ali** is a doctoral candidate at the Computer Science Department, in the Parallel and Scientific Computation Research Group at Cardiff University, UK. His research interests are in QoS and resource management in Grid computing, and Grid services. He received his BS degree from the University of the Pacific, Stockton, CA, USA, in 1992, and his MS degree from the George Washington University, Washington DC, USA, in 1997. His work experience covers foremost mainframe computers and UNIX systems, network administration and IT project management.

**Omer F. Rana** is a Senior Lecturer in computer science at Cardiff University, and the Deputy Director of the Welsh E-Science/Grid Computing Centre. He also acts as advisor to Grid Technology Partners — a US based company specialising in Grid technology transfer to industry. He holds a PhD in Computer Science from Imperial College, London University in parallel architectures and neural algorithms, an MSc in microelectronics from Southampton University, and a BEng in information systems engineering from Imperial College, London. His research interests are in the areas of high performance distributed computing, multi-agent systems and data mining. Prior to joining Cardiff University, he worked for over 5 years in the use of neural algorithms in a number of fields, including biotechnology, instrumentation and control.

**David W. Walker** is the Director for the Welsh E-Science/Grid Computing Centre at Cardiff University. He received a B.A. degree in mathematics from Jesus College, University of Cambridge, in 1976. His M.Sc. degree in astrophysics was obtained from Queen Mary College, University of London, in 1979, and his Ph.D. from the same institution in 1983. He has held appointments at the University of London, the Jet Propulsion Laboratory, California Institute of Technology, University of South Carolina, and at Oak Ridge National Laboratory. His research interests focus on software, algorithms, and environments for computational science on high performance computers. He has been closely involved in the development of the ScaLAPACK parallel software library, and the MPI message passing standard. In recent years he has also been active in problem solving environments and agent based computing.

**Sanjay Jha** is an Associate Professor (Networks) at the School of Computer Science and Engineering (CSE) at the University of New South Wales. He has a PhD degree from the University of Technology, Sydney, Australia. He is the founder of the Network Research Laboratory at CSE, UNSW. His research activities cover a wide range of topics in networking including quality of service (QoS), mobile/wireless Internet, and active/programmable network. He is the principal author of the book Engineering Internet QoS (Artech House, 2002). He has been working as an industry consultant for major organizations such as Canon Research Lab (CISRA), Lucent and Fujitsu. In his previous job, he was a Lecturer at the School of Computing Sciences, University of Technology (UTS), Sydney. He also worked as Systems Engineer for the National Informatics Centre, New Delhi. He was a visiting scholar at the Distributed Computing and Communications Laboratory, Computer Science Department, Columbia University, New York, and Transmission Systems Development Department of Fujitsu Australia Ltd, Sydney.

**Shaleeza Sohail** is a PhD student in the School of Computer Science and Engineering at the University of New South Wales, Sydney, Australia. She received her Masters degree from UNSW in 2000. She is a member of the Network Research Laboratory of UNSW and currently is supervised by Dr Sanjay Jha. Her research focuses on providing extensions to the Bandwidth Broker, for supporting network level QoS for next generation applications.