QUALITY OF SERVICE BASED GRID COMMUNITIES

Omer F. Rana, Asif Akram, Rashid Al-Ali, David W. Walker
School of Computer Science, Cardiff University, POBox 916, Cardiff CF24 3XF, UK
{o.f.rana, a.akram, rashid, david.w.walker}@cs.cf.ac.uk

Gregor von Laszewski, Kaizar Amin
Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439, USA
{gregor, kaizar}@mcs.anl.gov

Abstract

Locating suitable services within a dynamic distributed system is a computationally intensive process, with no guarantee of quality and suitability of the discovered services. This is especially true for transient services, i.e. services which are likely to exist over short time frames. A significant effort has already been spent on developing distributed registry systems – such as the UDDI registry in Web services – to enable services to be published, and subsequently discovered. Regardless of the type of registry being used, it is nevertheless important to categorise services based on their particular properties – a process that should also aid the subsequent discovery of the service. A mechanism for grouping services based on a particular set of properties is investigated here – leading to the formation of service communities. Each such community is based on the existence of common parameter values being shared by members of the community. The structure of such a community is described, and a particular type of community established on the basis of Quality of Service properties is subsequently developed.

1. Introduction

There has been an increase in interest recently within the Grid community [8] towards "Service Oriented" Computing. Services are often seen as a natural progression from component based software development [13], and as a means to integrate 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 interface stresses interoperability, and may be dynamically discovered and used. According to Foster et al. [7], 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 “TeraGrid” project [14] provides an example of the use of services for managing access to computational and data resources. In this project, a computational cluster of Intel IA-64 machines may be viewed as a compute service, for instance – hiding details of the underlying operating system and network. A developer would interact with such a system using the Open Grid Services Architecture (OGSA) toolkit [7], derived from the Globus system [2], and consisting of a collection of services and software libraries.

A key aspect of many existing Grid computing applications is the distributed sharing of services – often organised into a “Virtual Organisation" (VO) [7]. Based on this idea, services from different providers may be dynamically combined based on demand (although the location of a service is often pre-specified), to enable the composition of these services for solving a single large problem. Each service may be shared concurrently between a number of VOs. The discovery of suitable services plays a significant role in organising and managing such organisations.

In cases where the location of services is not pre-defined, the notion of discovery becomes a critical and often time-consuming process. Service discovery imposes an overhead on network access, as the time to undertake discovery increases as the number of service providers/users (peers) increase. Grouping services and restricting interactions to be between a set of peers is a key factor to scale the resource discovery problem. Any initial cost used in categorising peers can provide benefits for discovering preferable peers without a large discovery cost subsequently – thereby leading to the development of “communities". A similar problem in Grid Computing is the “connection problem" [5], where peers need to find other suitable peers to co-operate with, assist, or interact with. “Focused Addressing" [12]is one solution to the connection problem where requests are sent to particular subset of peers, believed to assist the requesting peer. Individual peers, although selfish, are expected to interact with each other in some way. If this was not the case, it would be impossible to establish VOs – or allow sharing of services between different applications. Co-operation of one form or another therefore becomes essential. Each peer prefers to be in an environment where it may be easily discovered by a suitable service user, and can locate other peers with minimum effort/cost. To assist

discovery, peers providing services may be grouped together based on common attributes such as: type of service, resources owned, and domain of operation, for instance. We assume that each community has a manager entity or a "Service Peer" responsible for sustaining the community, and interacting with Service Peers in other communities. Such a hierarchy (whereby individual peers are not allowed to interact directly with peers in other communities), is also useful to restrict the number of messages exchanged between communities.

Existing work in supporting service discovery using registry services, such as UDDI, differs from our work. Current work in Web Services does not specify how UDDI registries must interact with each other, or how they may be organised into a hierarchy. Although there have been some efforts toward the establishment of "Business Registries" – essentially involving a collection of UDDI registries, the exact interaction between such UDDI registries is not enforced. Our focus on communities also allows the existence of registry services within each community – to record services available within a community. Such a registry may be managed by a Service Peer, and support service discovery within the community. Another related work is the referral mechanism for service discovery found in systems such as Chord and Tapestry [4]. These rely on the use of overlay networks to minimise the number of search messages that need to be propagated to discover a service. Messages are often broadcast with a limited hop count or Time To Live (TTL) setting, to ensure that message traffic is constrained to the vicinity of the requesting peer. Once again, we may establish an overlay network over our Service Peers to support service discovery between communities. Our approach does not preclude the use of such message forwarding techniques.

2. Community Formation

Service based communities can be of many different types, with considerable research already having been undertaken in this area in the context of Multi-Agent Systems (MAS). The notion of what constitutes a *community* differs – with an emphasis ranging in scope from "functional" communities (those based on a particular application or problem domain) to those based on community "characteristics" (such as performance, trust/reputation, security policy etc). Another distinguish feature is whether the community structure is centered on the capability of individual participants, or the overall objectives/goal of the community in its entirety. Work in Holonic MAS shares many similarities with the formation of communities [6], and involves the establishment of a community as a *self-similar* structure that can be repeated multiple times. Each peer in such a community undertakes a similar activity, although peers can exist at different levels of a hierarchy. Other descriptions of communities are based on the types of activities undertaken by its participants. When partici-

pants are cooperative, the community can be a “congregation”, a “coalition”, or a “team”. A congregation generally consists of a meeting place, and the agents that assemble there (taken from the analogy of a club, a marketplace, a university department etc. in the context of human societies). Generally, members of such a congregation have expended some initial effort to organise and describe themselves so that they are considered to be useful partners with whom others can interact. Hence, within a community of this kind, agents somewhat know about the capabilities of others, and take ‘for granted’ some of the attributes that the other agents may possess. Brooks and Durfee [3] outline how such congregations may form, and various other infrastructure services that need to be made available (such as a MatchMaking service, the location of a congregation meeting place etc). The usefulness of a congregation-based community is the limited effort each agent within such a community needs to expend once it has established itself into a congregation. Such communities are therefore likely to involve repeated interactions and may generally exist over long time frames – as the whole point of developing such congregations is to allow an agent to have a greater level of trust in other participants (and therefore devote less resources to finding suitable partners for interaction). Panzarasa, Jennings and Norman [11] explore a formal model for specifying collaborative decision making, in which agents coordinate their mental models to achieve a common group objective. They indicate that such a decision making will be impacted by the social nature of agents, which also motivates their particular behaviour. The formation of such a community involves agents which provide some commitment to the group in which they belong.

For the work presented here, a community may be defined as a “collection of agents/peers working towards some common objectives, or sharing some set of common beliefs". An agent may simultaneously belong to one or more communities, and must make a commitment to remain within a community for a particular duration. Each community may therefore be governed by a set of policy rules that all participants within the community must adhere to. These policy rules are managed by the Service Peer in the community – which is also responsible for ensuring that these rules are being adhered to. A policy may be presented to each joining peer during the community formation phase, and must be accepted prior to the peer being allowed to operate within the community. The Service Peer may also mediate interaction between peers within a community, and may support a number of common services (such as an “Event Service", a “Name Service", a “Resolver Service" etc). Groups may be defined, as illustrated in Figure 1 – the C5 community includes C1–C4. Similarly, a community manager may share common services from a community at higher levels of the hierarchy (for C1,C2 for instance). In such a hierarchy, the community managers also act as gateways between multiple communities, and may facilitate inter-group query forwarding. Community

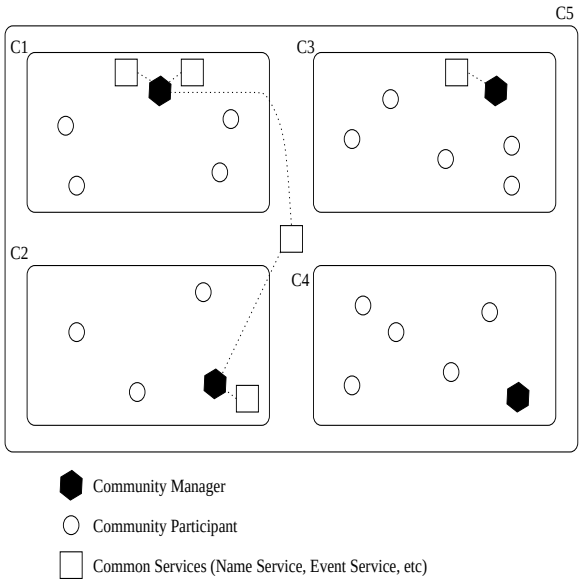

Figure 1. Hierarchical Communities

managers at a higher layer can interact and delegate activities to community managers at lower layers, and also register the address of lower layer community managers. Service and participant information is not replicated at higher layer community managers – simply the names of community managers who may then be able to provide additional details about participants within a community. The structure of such a community is significant, as it impacts the search time for locating particular services and peers. It is assumed that once an agent has agreed to participate within a community by agreeing to a policy, the agent is given a higher level of trust by other agents, and the community manager. However, enforcement of policy rules is not considered here – and therefore the case where agents are malicious and violate policy rules is not considered. Hierarchical communities provide a useful way to structure interactions between peers, with each Service Peer acting as message relay to other communities – constrained by the policy rules operating in the community.

The existence of a Service Peer is the minimum requirement to establish a community – as it allows the registration of new services, and keeps track of which peer is present within the community. We identify two concepts to illustrate how peers may be admitted to a newly formed community. The first of these is *Expertise* or *Capability* – and refers to one or more services that a particular peer offers. The fact that a peer has the ability to provide such services to others, allows us to associate such Expertise with the peer. The second concept relates to the *Interests* of peers within a community – and

represents particular activities that a peer would like accomplished, but does not directly possess the Expertise to carry these out. Collaboration between peers is therefore also necessary to enable the Interests of a peer to be carried out (provided that there are other peers which provide the Expertise to do so). A new peer wishing to join a community must similarly meet the policy requirements (consisting of Expertise and Interests) identified by the Service Peer. If the Interests of a Service Peer are different from the Expertise being offered by a peer requesting membership, then the new peer is referred to other Service Peer/s, or the new peer tries to locate alternative Service Peer/s with compatible Interests.

A Service Peer manages all peers within the community and communicates with neighbouring Service Peers on the behalf of its members. A Service Peer is also essential for the bootstrapping of a new peer, as it provides support for a new peer to discover enough network and compute resources to sustain itself within a community. A Service Peer may interact with a monitoring service within a community to achieve this. More generally, a Service Peer may therefore make use of other infrastructure services (such as monitoring, directory, security/certificate authority, etc) within each community. Such common services may be supported within a community, or may be external to a community and shared. Figure 1 illustrates how these services are organised both within and outside a community.

2.1 Types of Communities

Individual autonomous peers have Expertise and Interests in specific resource/s. Based on these Expertise and Interests, peers may be grouped together within a single community. It is therefore possible to divide communities into the following types:

- Competing Community: In such a community each peer has the same Expertise – although some service attributes may vary. Similarity in services may develop competition amongst member peers, leading to a competition amongst peers to be selected for offering a service by the Service Peer. Competing communities enable the grouping of common services, and require a service provider to identify ways in which its own Expertise may be distinguished from those being offered by others in the same community. Each peer within such a community may be self-contained (i.e. be able to execute its own services).

- Cooperative Community: In such a community peers either provide different Expertise, or may offer a non-overlapping set of services. In such communities, each peer requires interaction with another local or external peer to execute services it manages. Hence, when one peer is selected, then the possibility of selecting another member peer is increased. There

may thus be a mutual dependence between peers within such a community.

- Goal Oriented Community: Such a community is similar to a Cooperative community, and peers work together to achieve a particular goal. Membership in such a community is only allowed if the perceived Expertise offered by the joining peer allows the community to accomplish its goal. Such communities may be important in self-organising systems, where interactions between member peers are not pre-defined, but the services required are. In such instances, member peers may interact with each other in arbitrary ways to achieve a given end result. A goal oriented community differs from a cooperative community in that the goal being pursued may change over time – whereas the Expertise possessed by a cooperative community are expected to be more static.

- Ad Hoc Community: Here peers can be in a co-operative or competing community, but need to work together as a team. In ad hoc communities, peers interact directly with each other without interference and involvement of a Service Peer. Peers belonging to different communities providing different but supporting services form the basis of an ad hoc community, as long as both concerned communities have agreed to use each other's service.

- Domain-Oriented Community: Such a community is formed by linking together similar-minded organisations and institutions, instead of the services they provide, such as a music sharing community, a research community, and an open-source community for a particular type of software. Hence these communities are domain-oriented rather than service-oriented.

Describing communities in terms of their Expertise and Interests in this way, allows us to map other related concepts to these. Hence, a Virtual Organisation (VO) may be mapped to a Cooperative or Goal oriented community – depending on how often the environment within which such a community exists changes. Similarly, a "market place" may be mapped to a competing community, where different sellers and buyers operate to reach market equilibrium.

The effectiveness of a community may be characterised via its "rating" by other Service Peers. Rating can be inter-community (i.e. the rating of one Service Peer by another Service Peer), or may be intra-community (i.e. the rating of the Expertise of a peer belonging to a particular community by the Service Peer). We base our rating measure for a community on the number of requests sent to a peer by others. Hence, the greater the number of requests forwarded to a peer, the greater will be its rating. Such a metric provides a useful means to evaluate and compare communities offering similar Expertise.

The use of a “Ratings" index is therefore a subjective measure of effectiveness – seen from the perspective of a service user.

3. QoS-enabled Communities

As mentioned in section 1.2, a community is based on the Expertise and Interests of the Service Peer (acting as a community manager). Expertise and Interests can be “functional" – i.e. based on the names and parameter types of the services being provided or requested. If Web Service technologies are being used, these names and parameters are generally encoded in the Web Services Description Language (WSDL), and associated with particular `port types`. An alternative set of Expertise and Interests relate to the “non-functional" properties, such as performance, trust/reputation and cost (for instance). Often, such non-functional properties are ignored, even though they may be significant in choosing and discovering a peer. We utilise particular types of non-functional properties related to the Quality of Service (QoS) offered by a peer. We may now extend the Expertise and Interests of peers with QoS attributes; thereby impacting all the different types of communities discussed previously. Hence, in a cooperative community, peers may search for services which enable them to complete the execution of their service within pre-defined time constraints (thereby necessitating the need to evaluate the Expertise of other peers with reference to their QoS attributes). Similarly, in competing communities peers may be able to distinguish themselves based on the QoS attributes they offer. QoS attributes in this context relate to network parameters associated with accessing a service, and execution parameters in running a service managed by a peer. For instance, network parameters include:

- Delay: this represents the time taken to deliver a message from a source to a destination peer
- Jitter: this represents the variation in the delay of messages being sent, along the same route, from a source to a destination peer
- Throughput: this represents the number of messages that are received by a destination peer within a particular time frame
- Loss rate: this represents the number of messages that are lost or become corrupted within a particular time frame

Each of these parameters can be monitored by the Service Peer. Similarly, parameters associated with service execution are related to the percentage of CPU time being allocated to a given service [1]. We assume that by default communities are not QoS-enabled – as providing QoS support within a community is an overhead for the Service Peer. In this context, a QoS guarantee is essentially an agreement to provide a service according to an initially negotiated contract

between a community manager (the Service Peer) and an external peer. An additional set of management activities are now required within a community to enable a Service Peer to offer such guarantees.

A structure of a community is not static, and the Service Peer will try to adopt the structure based on changes in the environment in which the community exists. This is achieved by controlling the membership of peers that belong to the community. Communities which fail to adapt themselves will end up reducing their overall rating. As our rating measure is based on the number of forwarded requests to a community, we can make use of QoS attributes to measure how effectively these additional requests are being handled. As a community has a limited number of physical resources (network and CPU capacity), increasing the number of requests will also degrade the community QoS. Communities which are QoS enabled are better suited to providing such assurances – as the change in QoS due to increasing workload can be monitored. A Service Peer, for instance, may refuse to accept any additional requests if its QoS is likely to degrade.

As not all communities are likely to be QoS enabled, it is necessary that Service Peers of only QoS enabled communities rate each other. A particular community may decide to become QoS enabled when it has enough resources to work effectively even at high intra-community loads. If a community cannot cope with this extra overhead, then it may badly affect the overall performance of the community. Furthermore, it is important that a community should have enough resources to provide the same service through different mechanisms, so that its QoS cannot be compromised due to the failure of a single peer.

3.1 Properties of QoS-enabled Communities

QoS Enabled Communities only grant membership to peers which support QoS assurance. When a peer wishes to join a QoS enabled community, it must provide its initial QoS properties, and the associated performance information about its services. This information is stored in the Service Profile Repository (SPR), and activates a membership protocol. Figure 2 illustrates a QoS enabled community – offering additional management services to enable the Service Peer to monitor and enforce QoS attributes. In such a community:

1 Peers are created to provide specialist service/s (to enable the measurement of QoS attributes) along with basic/core services.

2 Peers are physical hardware resources attached to the network. Two peers cannot represent a single hardware resource and vice versa.

3 Peers which do not provide support for measuring and monitoring QoS attributes may not be allowed to participate in a QoS enabled community.

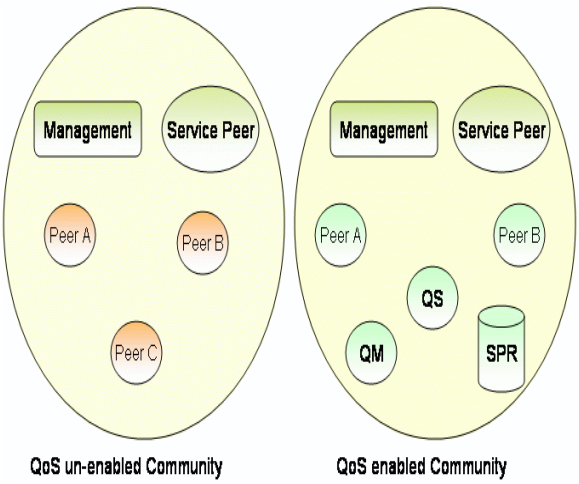

Figure 2. QoS enabled Community

4 Peers normally decide to become QoS enabled based on the type of community(ies) to which they belong.

5 A QoS enabled peer may participate in any community.

6 A community may decide to become QoS enabled to improve its rating. The rating of a community is related to the number of requests received by a community over a particular time frame.

As illustrated in Figure 2, QoS enabled communities have four additional components:

- The Service Profile Repository (SPR): can be a database or service registry. Its main role is to hold QoS related properties about services offered by a designated peer. For example, the service duration and service resource requirements are QoS properties. These properties are recorded/updated in the SPR in two forms: (a) On accepting membership of a new peer into the QoS enabled community – the peer must provide QoS properties of services offered, and these properties are entered in the SPR in the form of profiles for services. (b) On completion of a service by a client – in this case the QoS Monitor (QM) updates the profile in the SPR for this particular service. The update mechanism uses a proportional approach of number(s) of service usage(s) vs. resources consumed in every usage, and the duration of service execution.
- The QoS Scheduler (QS): in a QoS enabled community, the QS works together with the service peer and the QoS Support module of a desig-

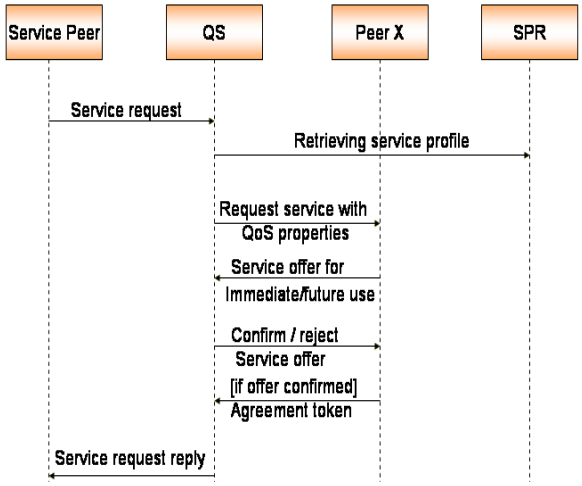

Figure 3. QoS Negotiation

nated peer to provide services with QoS guarantees. Its main role is to receive requests from the service peer, for services providing QoS assurance. The QS also indicates to a peer whether the service requested can be immediately honoured, or when it is likely to be honoured depending on the existing schedule that it maintains. The QS undertakes a QoS negotiation process with the designated peer, with QoS properties obtained from the Service Profile Repository (SPR) for the requested service. The result of the negotiation process is the reply to the service peer, which the service peer can then use to admit, or reject, a client's QoS-based service request. Figure 3 illustrates the negotiation protocol.

- QoS Monitor (QM): this is used to track the execution pattern of each service and gather performance characteristics for QoS requirements. It uses this information to update profiles in the SPR, for use by the QS when needed.

- The QoS Module: this resides in the peer node and it is, in essence, a QoS management system which supports QoS negotiation, provides advance/immediate resource reservation, supports resource allocation/release and an agreement protocol to confirm resource reservation. The QoS Module provides a guaranteed execution environment for services, described in terms of resource requirements, the starting time of the execution, and the likely duration for which a particular resource may be needed. The resource requirements can be expressed in terms of 'compute', or CPU specifications – such a CPU specification is supported by: (a) the total compute power, or (b) a specific compute slot offered by

the peer. The four QoS management subsystems providing the required assurances are:

1 QoS Manager: this is the interface for the QoS management system. The QoS Scheduler (QS) interacts with the QoS system through this interface. It accepts requests for QoS services and orchestrates the request between the components of the system, and then generates a reply to the QS.

2 Policy Manager: this provides the required dynamic information about resource characteristics offered by the peer, and policy information of when, what and who is authorised to use the community resources. This information may be updated by the peer owner or the Service Peer.

3 Reservation Manager: an abstract data structure that holds reservation entries for a particular resource(s). It does not interact directly with the resources, but obtains resource(s) characteristics from the Policy Manager. When a reservation request is received from the QoS manager, the reservation manager undertakes an admission control test to check the possibility of a reservation promise to the requester. Similarly, for service execution, the QoS manager checks the reservation data structure for the reservation, and, subsequently, instructs the allocation manager to allocate the reserved resource(s).

4 Allocation Manager: this module is used to interact with resource managers for allocation and de-allocation of resources, based on requests from the QoS manager. A variety of resource managers can be used; for example, a Dynamic Soft Real Time Scheduler (DSRT) [10] for allocating compute or CPU resource(s).

3.2 QoS-enabled Peers

As previously discussed, peers are not QoS enabled by default, but may decide to become QoS enabled when the communities to which they belong start to support QoS constraints. This is achieved by the Service Peer within the community deciding to support additional services mentioned in section 1.3.1 above. QoS enabled communities can only have QoS enabled peers, and if any peer decides to not support this functionality, its membership may be terminated by the Service Peer. Each peer that belongs to a QoS enabled community must agree to provide performance data to one or more services managed by the Service Peer in the same community. This performance data includes parameters related to execution time/resource usage, and network parameters (such as bandwidth/latency. A peer may be a member of two communities in two situations, as follows:

Case 1: In this case, a peer is a member of both a QoS enabled and an un-enabled community. As a member of two communities, service requests are expected from two different types of Service Peers. A service request received from a Service Peer of the un-enabled community will be treated on a 'best-effort' basis, with the service executed when resources become available – in this instance, no assurance on service execution is provided. The following scenarios could occur:

- A request is received while no other services are running on the peer, and the 'best effort' mechanism will make use of the all the available resources to execute the service.
- A request is received while a 'guaranteed' service(s) is/are running on the peer belonging to the QoS enabled community. The new request is now scheduled to run on the remaining resources – if any are available after responding to the requests from the QoS enabled community. The implication is that unpredictable execution patterns may occur for services that do not provide QoS data; for example, the service might take a long time because of insufficient resources, or might run normally as the available resource is sufficient – no prior guarantee as to which mode of execution is used can be made.
- A request is received while a guaranteed service(s) is utilising close on 100% of the compute resource. Here the 'best effort' service will not be immediately executed, and will have to wait until the resource is released by the 'guaranteed' service.

Case 2: In this case, a peer may be a member of two QoS enabled communities, and requests are expected from two different QoS Schedulers (QSs). The QoS management system operates on the concept of a 'First In - First Out (FIFO)' scheduling mechanism. When either of the QSs attempts to negotiate for QoS requirements, the QoS manager passes the request to the reservation manager, which in turn runs an admission control test to check the feasibility of granting this request. If the request cannot be admitted, a reject message is sent to the QS with a proposal for the next available time, and it is up to the QS to accept or discard. On the other hand, if the request is granted, an accept message is returned to the QS for confirmation or rejection.

Although QoS guarantee is a desirable feature it has associated drawbacks – viewed as trade-offs for the promised assurance. The QoS overhead can be classified into:

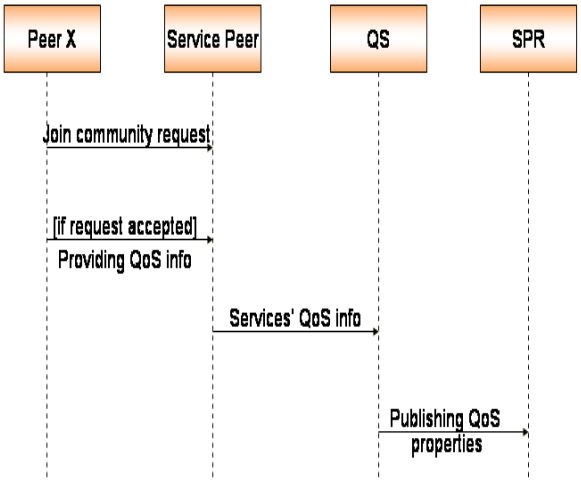

Figure 4. Protocol for joining a QoS enabled community

- Request protocol overhead: The request protocol overhead is when the Service Peer receives a request for service but the request does not propagate directly to the peer who provides the service; it goes to the QS. The QS then goes through the negotiation process as outlined above. The negotiation process may succeed or fail. This negotiation process is a request protocol overhead.

- Execution protocol overhead: Similarly, during service execution, when the QoS manager receives a service execution request, it does not start the service directly, but goes though an execution admission test, which checks if the required resources have been reserved. This is viewed as an execution protocol overhead.

It can be argued that this protocol overhead is a compromise to the assurance promised, otherwise if the 'best effort' execution is chosen, no guarantee on execution times can be provided. We envision associating a cost model with the 'guaranteed' services, so this cost factor will play a major role in assisting clients to select their best option – and is used to influence the service rating mentioned in section 1.2.1. Figure 4 illustrates the `Join` protocol.

4. Implementation Status

The central idea of a community is implemented using the JXTA library [15] from Sun Microsystems – which enables the development of Peer-2-Peer systems in Java. In our implementation, each community is therefore essentially a JXTA group, consisting of a collection of peers and a Rendezvous peer (which pro-

Figure 5. A JXTA Community Manager: User Interface

vides some of the functionality of the Service Peer), along with a collection of common services (such as a registry service). Each peer on joining the community registers its expertise – as a collection of service signatures specified in XML with the Service Peer. At any point in time, the Service Peer has a list of all the services that are available within the particular group that it manages – therefore each peer registration within the group leads to an update of the `PeerGroup` advertisement managed by the Service Peer. The peer joining the JXTA Group is also allocated a unique identity by the Service Peer. Each JXTA Group has a sorted list of its member peers, and each peer present within this group has a sorted list of Groups to which it belongs.

External peers apply for membership by sending their service advertisement to the Service Peer (this is done when a peer is first created – or when a peer wishes to change its current group), and selection is then based on the type of community that the Service Peer is managing (competitive, cooperative, etc – as outlined in section 1.2.1). The rate at which advertisements are being sent from a peer may be on a periodic basis, or may be stimulated by events such as a new peer joining/leaving the peer Group, or geographical information indicating the other services that are offered at a particular location, etc. Each peer has its own thread of control, and decisions regarding when an advertisement should be published is left to the peer owner/developer. Each peer must also provide a listener interface that can be implemented and registered with the Service Peer. Using the listener interface, a peer can register interest in other peers that are also registered with the Service Peer (within the same community) – and the identity of the registering peer is used to propagate subsequent event information to it.

A Service Peer may terminate the membership of a peer at any time – and a peer may decide to leave a group at any time. We assume in our implementation

that the identity of a peer (and a group) does not change over time. To support group management, we provide an interface that enables a developer to view the current state of the group at any time – this is illustrated in figure 5. Each Peer advertisement (provided in Code Segment 1) contains both optional and mandatory elements. The mandatory elements are `PID` (peer identity), `GID` (group identity), `Name` (peer name), and `Desc` (summary of peer expertise). The other elements, such as `Dbg` (debug) and `Svc` (service description) are optional – hence, it is possible in our scheme to have peers which provide no service. A peer may not provide any service when it first joins a community, for instance – and therefore the `Svc` elements may be blank. Subsequently, once a peer has been part of a community for a particular duration, it may issue a new peer advertisement (using the same `PID` as before) and advertise its new services. The use of blank `Svc` elements within a peer advertisement may be useful when bootstrapping a community. The peer advertisement is seen by all the peers within the same community, and also by the Service Peer – restricted by the `GID`. Each peer advertisement can contain multiple `Svc` elements. The Service section may also optionally contain an element `isOff` meaning that this service is disabled. This element is used to convey a configuration choice made by the owner of the peer. The `Svc` element may contain details about a particular expertise held by the peer, or may be a reference to a service maintained in some repository. For instance, such a reference may point to a service maintained in a UDDI registry. Each of these `Svc` elements describe the association between a group service denoted by its `MCID` (Module Class ID), and arbitrary parameters encapsulated in a `Param` element. The `MCID` is essentially a service identity that uniquely identifies a given service – even though the service may be provided by different peers. The `MCID` may also acts as a service name – and is generated by a running a factory service with a URL reference to the service location – which returns a unique identity for the service at the specified location. The `Param` elements should specify the exact set of parameters (and their ordering) that need to be sent to a service – and must also specify the types of return parameters that are sent in response. There is no mechanism in JXTA to specify which parameters are inputs or outputs in the Peer advertisement (Code Segment 1), and it is therefore necessary for a service provider to specify this information in the Module Specification advertisement (Code Segment 3). A Module Specification advertisement may also specify how to invoke and use a service.

Code Segment 2 and 3 identify a way to describe Module Class IDs. Each such ID must have a name and unique description, and must specify a way to invoke the service – through a JXTA “pipe" (equivalent to a named communication channel). There are also other ways in which the service could be invoked (other than the use of a JXTA pipe – although this is assumed to be the default mechanism), such as use of Java RMI or via a SOAP message. In the case of

invocation using SOAP, a URL to the location of the service needs to be provided. Each Module Specification Advertisement (Code Segment 3) can have multiple Module Implementations (Code Segment 4). It is therefore possible to have multiple implementations for a service (with a particular `MCID` to exist. Whereas the Module Specification (Code Segment 3) only provides a higher level specification of the service, the Module Implementation (Code Segment 4) needs to provide details of the exact bindings for invoking the service. Each service implementation for a particular `MCID` must have a unique `MSID` (Module Service ID) – as illustrated in Code Segment 4. In this way, it is possible to uniquely identify an implementation for a particular service specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PA xmlns:jxta="http://jxta.org">
<PID>...</PID> <GID>...</GID>
<Name>...</Name> <Dbg>...</Dbg> <Desc>...</Desc> <Svc>
<MCID>...</MCID> <Param>... </Param> </Svc>
</jxta:PA>
```

Code Segment 1: Peer Advertisement

```
<?xml version="1.0" encoding="UTF-8" ?>
<jxta:MCA>
<MCID>...</MCID> <Name>...</Name> <Desc>...</Desc>
</jxta:MCA>
```

Code Segment 2: Module Class Advertisement

```
<?xml version="1.0" encoding="UTF-8" ?>
<jxta:MSA>
<MSID>...</MSID> <Name>...</Name> <Crtr>...</Crtr>
<SURI>...</SURI> <Vers>...</Vers> <Desc>...</Desc>
<Param>...</Param>
<jxta:PipeAdvertisement>...</jxta:PipeAdvertisement>
<Proxy>...</Proxy> <Auth>...</Auth>
</jxta:MSA>
```

Code Segment 3: Module Specification Advertisement

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:MIA>
<MSID>...</MSID> <Comp>...</Comp> <Code>...</Code>
<PURI>...</PURI> <Prov>...</Prov> <Desc>...</Desc>
<Param>...</Param>
</jxta:MIA>
```

Code Segment 4: Module Implementation Advertisement

As indicated in code segments 1–4, each peer belongs to one or more JXTA groups – and supports one or more services. A service may be directly provided by a peer and specified in its service advertisement, or alternatively the properties of a service may be registered within a UDDI registry service also provided within the same community. In the latter case, the advertisement of a peer contains a reference to the UDDI registry containing elements of the WSDL document for the service.

Consider a peer within a community providing a `Stock Quote` service. This service should have an XML based advertisement so that other peers (within the same community) or the Service Peer can discover it, and forward requests to it. Code Segment 5 provides a user defined Stock Quote Advertisement. This description of a service differs from Code Segment 2, as only one instance of such a user defined service can exist within a community. Hence, the format for service specification in Code Segment 2 is the default format expected by JXTA – however Code Segment 5 illustrates how a user defined format can be used instead (with constraints on the number of implementations that can co-exist – in the case of Code Segment 5 this is restricted to one).

```
<?xml version="1.0 "encoding="UTF-8" ?>
<StockQuoteAdvertisement> <Name>...</Name> <Desc>...</Desc>
<Status>...</Status>
</StockQuoteAdvertisement>
```

Code Segment 5: Stock Quote Service Advertisement

The `Name` element is used to encode a developer defined name for the service, the `Desc` element a (human readable) text string indicating what the service does, and the `Status` a developer defined element encoding the current state of the service. A peer creates a service advertisement by using an abstract class, mainly providing `set` and `get` methods to access the elements defined in the advertisement. Using these methods, it is possible to assign and retrieve values for particular elements within the advertisement. The abstract class must make use of a parser to analyse the structure of the advertisement in XML (i.e. retrieve the order in which XML elements are to be parsed, and subsequently processed).

The abstract class and the parser are mainly used to process the advertisement. Additional code is needed to be able to publish and respond to discovery requests/queries from other peers. This is achieved by defining the `publishService`, and the `discoveryEvent` methods. Finally, an implementation is associated with the service (and is similar to the description in Code Segement 4). Only a single implementation can now be associated with the service advertisement.

Figure 6(a) illustrates how a request to discover a service is propagated across different communities. Each peer within a community (including a

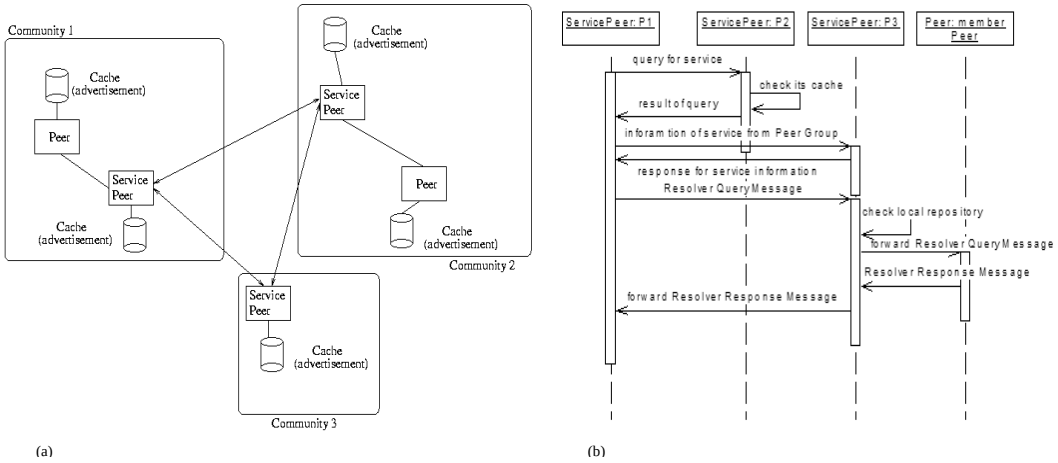

Figure 6. (a) Advert propagation between different Service Peers, (b) Sequence diagram indicating message exchanges between Service Peers

Service Peer) can support an advert cache – which is searched before a request to discover a service is propagated to the Service Peer (intra-community) or to another community. As indicated in figure 6(b), once a service has been discovered in another community (via Service Peer 3), it is now necessary to locate the peer providing the matched service – this is achieved by the `Resolver Query` message. The response message contains the `PID` of the requesting peer and the `HandlerName` – the `HandlerName` is an aggregation consisting of a `PID`, the `MCID` and the `MSID`, and uniquely identifies a particular service implementation being offered by a particular peer. Code Segment 6 illustrates the structure of the `Resolver Query` message.

```
<?xml version="1.0 "encoding="UTF-8"?>
<jxta:ResolverQuery xmlns:jxta="http://jxta.org">
<HandlerName>...</HandlerName> <Credential>...</Credential>
<QueryID>...</QueryID> <SrcPeerID>...</SrcPeerID>
<Query>...</Query>
</jxta:ResolverQuery>
```

Code Segment 6: Resolver Query message

In order to now make use of a service, a peer sends a query either directly to the peer providing the service (in an Ad-Hoc community) or to the Service Peer (in the other types of communities). The query should be wrapped in the `Resolver Query` message (Code Segment 6), in our implementation we have java classes to allow query messages to contain both optional and mandatory parameters. This message should now contain the exact set of parameters needed to initiate this request – for the `Stock Quote` service, this is illustrated in Code Segment

7 – where the `Min` and `Max` elements refer to the minimum and maximum price for the stock on a particular `Date`. Subsequently, a `Query Response` is sent to the requesting peer – illustrated in Code Segment 8 for the `Stock Quote` service.

```
<?xml version="1.0 "encoding="UTF-8" ?>
<InitiateStockRequest>
<StockName>...</StockName>
<Date>...</Date> <Min>...<Min>
<Max><Max>
</InitiateStockRequest>
```

Code Segment 7: Stock Request Query

```
<?xml version="1.0 "encoding="UTF-8" ?>
<jxta:ResolverResponse xmlns:jxta="http://jxta.org">
<HandlerName>...</HandlerName>
<Credential>...</Credential> <QueryID>...</QueryID>
<Response>
 <StockData><StockName>...</StockName>
 <StockSymbol>... </StockSymbol>
 <link>http://quote.yahoo.com/q?s=SUNW</link>
 <price currency = "pounds">20.59</price>
 <detail>...<detail>
 </StockData>
</Response>
</jxta:ResolverResponse>
```

Code Segment 8: Query Response message

There are therefore three kinds of services that exist within our implementation of a community:

1 Services that are implemented as Web Services: in this instance, it is necessary to provide a description of the service in WSDL, and requires each peer to provide a Tomcat server (and a SOAP client for accessing services on other peers). In this case, each peer is a service provider that is capable of managing one or more services. Service execution is now undertaken by the particular Web Services hosting environment being provided by the peer. The inclusion of such a peer exists within a JXTA group is primarily so that it can be combined with other peers – depending on the type of community it belongs to. Although such services may have QoS constraints specified, there is no guarantee that these constraints are likely to be observed.

2 Services that are implemented with Java CoG Core [9]: in this instance, each service interface must also be specified using a WSDL interface, but

the execution of each service is managed via the CoG Core API. Each service now generates one or more `Task` objects – each of which has a unique `TaskHandler`. Using such a `TaskHandler` it is now possible to schedule these tasks on the resources managed by a single peer, or a collection of peers within the same JXTA group. Each such `TaskHandler` may be mapped to an `MCID` (in Code Segment 2). To support QoS-enabled services, the Service Peer interacts with the Dynamic Soft Real Time Scheduler (DSRT) [10]. If QoS guarantees are to be provided, it is now necessary for *all* services within the same community to be managed by DSRT. The Service Peer, in this instance, relinquishes control to this scheduling system. A Service Peer may pre-reserve resources based on likely demand from services within the community. Such demands for reservation are forwarded to the scheduler – which may accept or deny them based on the available resources and other services that are scheduled to run over a particular time frame.

3 Services that are implemented with JXTA execution engine: in this instance, each service interface is implemented directly using the `Svc` elements, and each peer must provide its own execution environment. No guarantees are available that particular execution quality will be observed. Interaction between services take place using JXTA pipes – and service discovery is supported through the propagation of peer advertisement (within the community) or `PeerGroup` advertisements between communities.

5. Conclusion

The concept of a "community" is a useful abstraction for grouping services. Communities can be of different types, and may support a variety of hosting environments. Each community has a "manager" responsible for admitting and releasing participants within it. Each participant within the community has its own thread of control – referred to as a "peer" – and may belong to multiple communities simultaneously. A community manager is responsible for allocating resource present within it, and for managing interactions with other communities.

We implement this concept of a community as a JXTA group along with some common services made available to all participants within the group. In such a group, a Rendezvous peer is responsible for caching advertisements generated by peers that belong to the group. Each peer can either directly provide a service, or may utilise a third party execution engine. We illustrate how a particular type of hosting environment is necessary in order to support QoS guarantees within a community. Although, we emphasise that it is important that not all communities may be QoS-enabled.

References

[1] R. Al-Ali, K. Amin, G. von Laszewski, O. Rana and D. Walker, “An OGSA-based Quality of Service Framework", 2^{nd} International Workshop on Grid and Cooperative Computing (GCC), Shanghai, China, December 2003, Springer Verlag

[2] Argonne National Laboratory, Mathematics and Computer Science Division, “The Globus Project". See Web site at: `http://www.globus.org/`. Last visited: December 2003.

[3] C. H. Brooks and E. H. Durfee, “Congregation Formation in Multi-agent Systems", Journal of Autonomous Agents and Multi-Agent Systems, Volume 7, Issue 1-2 (Special Issue), 2003

[4] M. Castro, P. Druschel, Y. C. Hu and A. Rowstron, “Topology-aware routing in structured peer-to-peer overlay networks", Microsoft Research, Technical Report MSR-TR-2002-82, One Microsoft Way, Redmond, WA 98052, US, 2002. Downloadable as: `ftp://ftp.research.microsoft.com/pub/tr/tr-2002-82.ps`

[5] R. Davis and R. G. Smith, “Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence, **20**, pp 63–109, 1983

[6] K. Fischer, M. Schillo, and J. Siekmann, “Holonic Multiagent Systems: The Foundation for the Organisation of Multiagent Systems", Proceedings of the First International Conference on Applications of Holonic and Multiagent Systems (HoloMAS‘03), 2003

[7] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, “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

[8] The Global Grid Forum. See Web site at: `http://www.gridforum.org/`. Last visited: December 2003.

[9] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, “A Java Commodity Grid Kit", Concurrency and Computation: Practice and Experience, Vol. 13, No. 8-9, pp 643–662, 2001.

[10] K. Nahrstedt, H. Chu and S. Narayan, “QoS-Aware Resource Management for Distributed Multimedia Applications", Technical Report UIUCDCS-R-97-2030, University of Illinois at Urbana-Champagne, 1997. Also available at: `http://citeseer.nj.nec.com/nahrstedt98qosaware.html`

[11] P. Panzarasa, N. R. Jennings, and T. J. Norman, “Formalizing Collaborative Decision-making and Practical Reasoning in Multi-agent Systems", Journal of Logic and Computation, Vol. 11, No. 6, pp 1–63, 2001

[12] H. V. D. Parunak, “Manufacturing Experience with the Contract Net", in book “Distributed Artificial Intelligence", pp 285–310, Research Notes in AI series, Morgan Kaufman, 1987

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

[14] The TeraGrid Project. See Web site at: http://www.teragrid.org/. Last visited: December 2003.

[15] B. J. Wilson, “JXTA", New Riders, June 2002. A version of this book is also available from the authors Web site at: http://www.brendonwilson.com/projects/jxta/