Service Level Agreement Requirements of an Accounting-Driven Computational Grid

Thomas Sandholm
Dept. of Numerical Analysis and Computer Science and PDC
Royal Institute of Technology
SE-100 44 Stockholm, Sweden
+46-8-7907811

Abstract
In this paper[1] we present the requirements of a national computing Grid. In particular we discuss the issues involved in managing complex policies of multiple stakeholders in such a large-scale, dynamic, and heterogeneous Grid. We also propose a Service Level Agreement (SLA) and agent-based architecture to address these issues. This work is a continuation of the work performed and experiences gained when we developed a Grid accounting system for the Swedish national Grid network, called SweGrid, which provides the foundation for the investigation presented here. We conclude that many SLA concepts fit very well within the SweGrid network to address some of the issues of the current system. Future work includes prototyping parts of the SLA framework and running simulations before eventually deploying it in the SweGrid production environment.

1 Introduction

In recent years, the Internet has had a virtually explosive growth in number of users, largely due to the fact that there has also been matching technological progress. Moore's law is often quoted when comparing CPU speed, but the advances in network performance have improved even more rapidly. As a result, we are now at a point where communication between computers distributed over large geographic areas almost matches the internal bus communication of PCs.

This trend, in conjunction with the ubiquity of the Internet and the price drop of high performance computing devices, prompted some researchers in the mid 90's to try to build virtual supercomputers operating across WANs like the Internet [1]. The initial results were very promising and this new architecture became known as the Grid to emphasize that computing power could be viewed as a utility akin to the electric power grid [2-4].

Today, the Grid has moved beyond the pure academic research projects, and the industrial involvement has gained momentum [5, 6]. For industrial innovators, the Grid offers a means to fulfill some long-sought dreams, such as charging for service usage, and outsourcing and automating management of high-end resources [7].

The main issue with the current Grid infrastructure is that, like the Internet, it operates on a best-effort basis. That is, there are no guarantees of delivered service quality. This mode of operation is sufficient when it comes to free information provisioning via, the World Wide Web, for example, but when it comes to delivering services or making computations to solve complex problems, for which a customer may have paid a large amount of money, new solutions are needed. Some low-level solutions already exist to provide differentiated services over Quality of Service (QoS)-enabled networks capable of guaranteeing offered bandwidth [8, 9] or CPU [10]. The biggest problem with these solutions is that the low-level infrastructure of the Grid is intrinsically very heterogeneous, and it

[1] This work was partly funded by The Swedish Research Council (VR) under contract 343-2003-953

thus requires designs of much higher abstraction and coarser granularity. The key to such designs is interoperability through standardization, which also serves one of the cornerstones of the Grid: ubiquity.

As a first step towards guaranteeing QoS, service usage needs to be tracked. This is typically done by Grid accounting systems [11-14] and it is elaborated on in our previous work [11]. The next step involves the ability to request and claim agreed upon QoS parameters, such as response time and price. This capability can be provided by setting up electronic contracts between the stakeholders embodying the agreed-upon service level. This kind of contract is referred to as a Service Level Agreement (SLA). It is a non-trivial task to make sure that SLA contracts in a Grid environment are adhered to, because they often span many software and hardware abstraction layers. Coordinated monitoring and management software is thus vital for such a system. Furthermore, policing contracts is not simply a matter of making sure that users will not exceed their QoS grants, and that resources fulfill their promises. Such a static view may in fact lead to worse resource utilization and less service offered to the user. This is where policies are introduced to allow the stakeholders to state their preferences. The inherent large scale of the Grid and the disparity of its users make policy management a complex task subject to automation.

Automation and reasoning about complex and even contradictory and partly unknown policies is an issue that has occupied the mobile agent and soft computing communities. Some preliminary efforts are now underway to merge their resulting technology with the Internet [15] and the Grid [16].

In this paper, we discuss the requirements of an accounting enabled national Grid: SweGrid, set up to serve the Swedish research community. We then propose an architecture based on SLA management and agent-driven reasoning to use as a starting point for addressing these requirements.

Section 2 discusses Service Level Agreement technology in more detail. In Section 3, the SweGrid system and its requirements are presented. Section 4 then describes how SLAs could be used in SweGrid. An SLA Management system is thereafter proposed in Section 5. Finally, we draw some conclusions and discuss future work in Section 6.

2 Service Level Agreements

A Service Level Agreement (SLA) is a contract between a user and a provider of a service specifying the conditions under which a service may be used. An SLA specifies the agreed-upon level of availability, serviceability, performance, and operation both in high-level business-value terms understood by end-users, and low-level technical terms that can be enforced to reserve resource capabilities. Typically, an SLA contains Quality-of-Service commitments (including penalties and rewards), pricing policies, authorization policies, and negotiation policies. To ensure the authenticity of an SLA it is digitally signed by all parties using a trusted-third-party (TTP)-based model such as X.509 certificate authorities, a.k.a. the Public Key Infrastructure (PKI) [17]. An SLA can be constructed either by the user or the provider of the service. In the former case it embodies a request for resources or capabilities, and in the latter case it represents an offer of available services and guarantees. Once an SLA has been mutually agreed upon and signed it has to be actively managed to ensure that all the commitments are attained.

The SLA is a mutual agreement among all stakeholders in a service interaction, and everyone must therefore also be able to subscribe to possible modifications triggered either by other stakeholders directly or by an automatic upgrade or downgrade by the agents representing them. Not all stakeholders may, however, have the same flexibility in changing various parts of the agreement. Typically, the service provider only allows a very limited set of parameters to be renegotiated and there might also be temporal restrictions, e.g. when and how frequently updates can be made. Monitoring systems must, hence, be tied directly to the agreement both for the users and the providers. The monitoring system must also report when violations arise due to either overuse or failure to provide the qualities committed to. From a provider's point of view it might also be interesting to

detect underutilization, in order to reallocate idle resources. In a fully automated management solution it is, furthermore, desirable to predict when violations are about to happen by analyzing usage history and patterns in order to take corrective actions, such as adjusting the number of resources in an active pool, before the agreement is broken. The management system is only aware of the conditions under which the agreement can be considered violated, and what possible penalties it may cause various parties, but it must be possible to specify and configure application specific actions to take when certain violation events occur. To determine whether the SLA is in fact violated it must be possible for the management infrastructure to validate it automatically and deterministically whenever the agreement is in effect.

Agreement offers that may be negotiated are referred to as SLA templates, and play an important role in the discovery of appropriate services. Hence, the templates give providers a way to communicate possibly dynamic properties such as load constraints to consumers. The templates can also be viewed as the typical contract that the provider is most likely to be able to fulfill based on assessment of previous contracts.

SLA Languages have been proposed in [18-20] and are now converging within the Grid community in the WS-Agreement specification in the Global Grid Forum. Some initial experiments on WS-Agreement templates have been performed in [21]. As yet, negotiation and self-adapting capabilities have not been explored in this context.

3 National Grid Requirements

SweGrid is a national Grid providing compute resources to scientific projects in Sweden. It comprises 600 commodity PCs interconnected with a 10Gb/s high performance network evenly distributed among six High Performance Computing (HPC) centers. A Grid meta- scheduler [22] is responsible for publishing resource information as well as selecting the appropriate cluster where a compute job could run. The matchmaking process must, for instance, make sure that the user has the required authorization rights to run the job, and that the correct OS and run time environments are installed on the target machine. Most of the jobs are long running and trivially parallel (with minimal inter-subjob communication). A typical job could, for example, perform some brute-force, number-crunching task on a series of machines in an iterative manner, improving the precision of the calculation the longer the job runs. It is therefore important that the allocated and requested execution (wall) time of the job is actively enforced, as many of these jobs never stop running by themselves. The local cluster scheduling system has the low-level control of jobs. What makes this environment a typical Grid network is that the various HPC centers involved are very heterogeneous in terms of local polices used, such as security, and job reservation and prioritization strategies employed.
To enable efficient and fair resource sharing in this Grid an accounting system built on top of standard Web and Grid services protocols and middleware was developed [11] to track resource usage, and to enforce Grid-wide resource quotas in real time. Figure 1 shows a high-level overview of the accounting-enabled Grid.

1. In order to submit jobs that are charged against a quota granted to a research project, an account for the project needs to be created. The account embodies the resource entitlement of a group of researchers and is decorated with project specific policies that are enforced before the job is submitted at a cluster. Deployment and activation of the account is typically done by an accounting service administrator.
2. All the resources shared on the Grid need to run a resource information service collecting resource capability and QoS parameters. The parameters may be both static and highly dynamic.
3. The information service, in turn, registers itself and its cluster in one or more global index services with a subset of the collected QoS in order to allow for more efficient resource discovery.

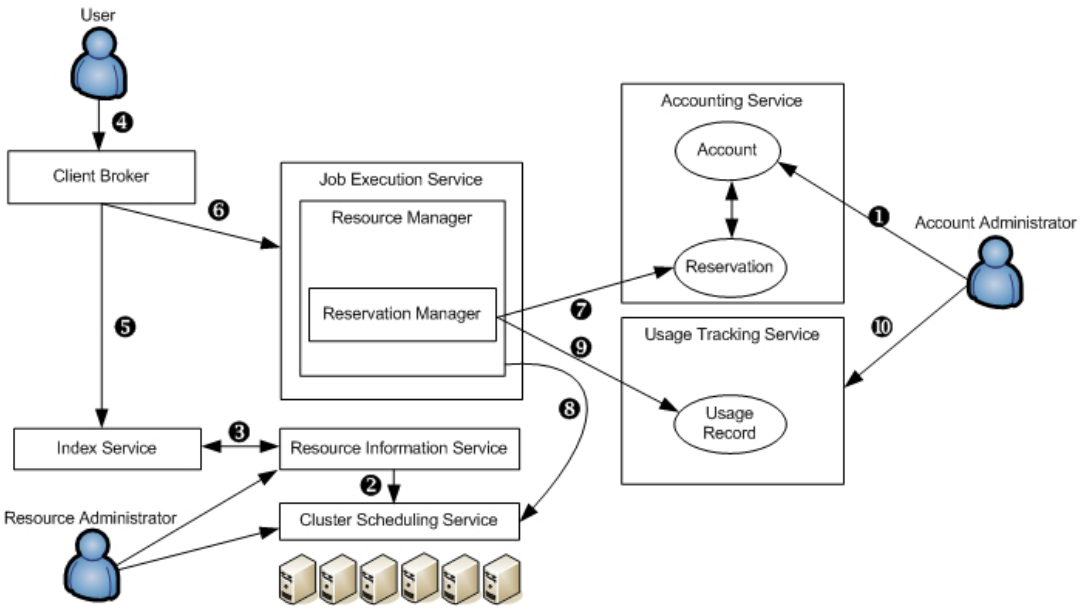

Figure 1: SweGrid Architecture

4. The user composes a resource request using a job specification language. The specification defines what capabilities are required to run the job, such as the required wall or CPU time, the number of cluster nodes needed, and the OS that the job executable can run on. It also contains job specific information including the name of the executable to run, and the required input parameters that need to be staged in to the target resource.
5. The job and resource specification is then passed into a client broker for submission. The first task of the client broker is to select a cluster that matches the user request using some matchmaking algorithm to distribute the load efficiently.
6. The second task of the broker is to submit the job to the cluster selected in the previous step and to perform all the required file transfers to stage in the input parameters to the local resource manager and scheduling system.
7. Before the job is actually put in a queue to run on one or more cluster nodes, the accounting system intercepts the call via a reservation manager component in the resource manager. The reservation manager looks at the resource specification that was initially composed by the client to determine what resources need to be reserved, for how long, and at what price. This decision is taken after evaluating local resource policies, which may differ between the clusters in the Grid depending upon, for instance, current load and cluster scheduling features available. A time-limited reservation is then created in the appropriate account, which was created in Step 1.
8. The job specification is now translated into the local scheduling system syntax and then submitted. The priority of the job is determined by the results of the reservation attempt in the previous step. If the quota of the account was exceeded, the resource, again depending on policy, may decide to downgrade the priority of the job or refuse to submit it altogether.
9. After the job has finished, the accounting system intercepts the job execution process once again in order to calculate the actual price of the job, log an accounting record, and charge the account. If the job finishes prematurely, a discount may, for example, be given on the price reserved in Step 7. The cost must, however, not be greater than was reserved by the user. How to treat and charge jobs that failed to execute is also subject to local resource policy. Regardless of what happened during the job submission, an accounting log is always recorded both locally at the resource as well as globally in a logging and usage tracking service.
10. To monitor how the accounting policies were enforced and to get a general idea of the performance of the system, the account administrator may query transaction information in the accounting service and more detailed usage information in the usage tracking service.

There are a number of limitations to and issues with the scenario just described. These issues are discussed next to give some background to the SLA re-factoring proposed in the next section.

- Account administrators currently have to add all account members (researchers who may consume quota granted to the account) manually and set up scripts to distribute a biannual allocation according to some policy, such as monthly staggered allocation chunks using a command line administration tool. This is error prone and exposes too much internal service detail to administrators. Further, there is no convenient way to see what allocation approach is in effect.
- The index service only exposes very limited QoS parameters for a cluster, typically only execution permissions. More detailed information may be queried by federating the query to resource information services, but the information gathered there is not usage based. Hence, it is, very hard to determine the ability of service providers (in this case the clusters) to meet their promised QoS parameters, and to support quality of compliance or reputation (trust) based service selection.
- The client broker algorithm is static and provides no flexible way of applying a policy to a group of job submissions to be performed as part of the same task. Handling of failed jobs, possible migrations, cancellation and monitoring of jobs must be done manually by the end-user. The result is that users either write their own scripts on top of the broker to their best ability without any form of algorithm or best practices reuse, or that they do the coordination by hand. In other words, the user-task goal fulfillment is completely manual and without any form of guarantees.
- The local resource policy is not directly linked to the user request or the service promised by the provider. Thus, negotiating policies is difficult, and as a result only very basic *demand for service* [7] negotiation is supported (the resource must accept all requests as is, or the job will fail). No automatic QoS monitoring or enforcement is carried out in relation to the job request. Therefore, violations cannot be detected and the parties involved cannot be notified to take countermeasures, such as migrating to another cluster if the job gets stuck in a queue due to some temporary node overload or downtime. Furthermore, there is no well-defined endpoint where these violation notifications can be sent. Because there is no infrastructure to detect violations, there is no support for predicting violations either.
- The resource-administrator policy for distributing resources fairly among clients while maintaining high utilization may be customized, but there is no middleware support for simplifying this task. There is no means to adapt the system automatically to changed circumstances or user requirements, or to automatically learn from past experience. The policies are all managed on a trial and error basis and by using off-line *a posteriori* analysis. It is not possible to correlate policy settings and resource infrastructure configuration with the ability to meet the QoS parameters promised to users.
- In general, the accounting-enabled Grid system described above is very flexible in terms of the ability to configure local stakeholder policies. However, this flexibility comes at the price of complexity, and it is very hard to map policies to higher-level goals. Examples include minimizing the overall completion time of a series of end-user job submissions while keeping the cost within a given budget; and maximizing utilization, throughput and resource sharing profits while meeting the offered QoS guarantees.
- The current use of implicit QoS contracts between the stakeholders makes it hard to offer discounts and enforce penalties to reflect the actual level of service delivered. The implementation of various economic market-simulation and pricing models is also complicated for the same reason. Further, there is no formal contract signing procedure making it hard for users or resource providers to enforce non-repudiation.

The issues listed above are summarized and mapped to the stakeholders, who are most negatively affected by the issue, in Table 1.

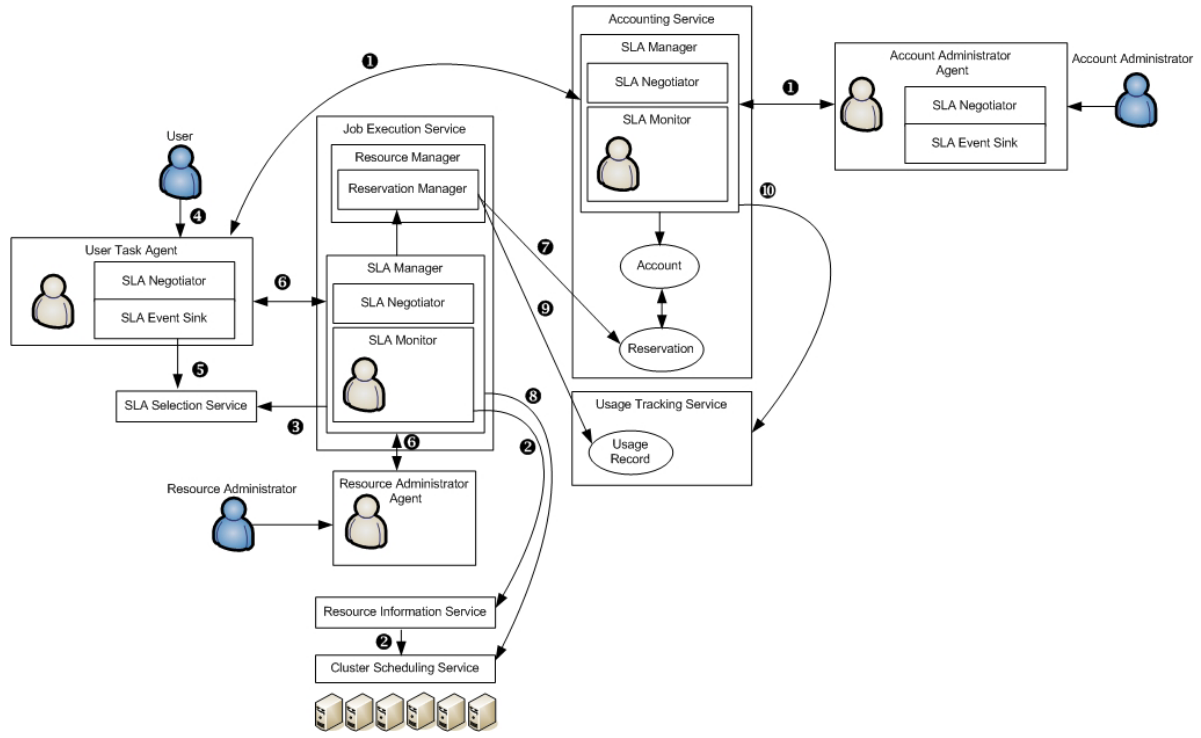

Figure 2: SLA-aware SweGrid Architecture

Table 1: Issues with current Grid Accounting System

Issue	Stakeholder(s)
Manual, error prone account management	Account Administrator
No usage based service selection	End-User
Manual, error prone multi-job coordination	End-User
No service level negotiation and enforcement leading to problems detecting service level violations	Resource Administrator, End-User
Complex policy configuration to optimize higher level goals	Resource Administrator, Account Administrator, End-User
Lack of adaptation and automatic reconfiguration support	Resource Administrator
No explicit, non-repudiation agreement support between stakeholders	Resource Administrator, Account Administrator, End-User

4 An SLA-Aware National Grid

We now propose an architecture aimed at eliminating the limitations and issues discussed in the previous section. The approach taken is to introduce an SLA-aware, middleware-driven architecture. Akin to the Grid accounting system presented in [11, 12] we want the new infrastructure to be as non-intrusive as possible and based on standard Web and Grid services protocols, because experience has shown that this is the best way to achieve both interoperability and early and smooth adoption. The internals of the SLA-aware components are presented more elaborately in the next section. Here we just show how these components can be applied to the Grid system presented in the previous section.
Figure 2 shows the SLA-enabled Grid architecture corresponding to the accounting-enabled Grid depicted in Figure 1.

1. We introduce an account administrator agent responsible for executing tasks on behalf of the account administrator and negotiating SLAs with the end-user representatives holding accounts in the accounting services administered. The contract negotiated by this agent can be seen as a fairly static overarching VO policy [23]. Contracts negotiated between end-users and resources in order to consume account quota must all comply with the terms in this contract. The agent can also set up new accounts and distribute allocations periodically according to a policy in the contract determined by the administrator. A sample SLA is shown in Figure 3. The contract must be signed by the account administrator credentials and may optionally be signed by the end-user representative. One-party signing is acceptable since the contract will only describe the obligations of the accounting service.
2. As in the SLA-unaware architecture the Resource Information service collects resource information from the local scheduling service. This information may for instance be in the form of the standard GLUE schema or JSIM resource models.
3. The SLA Manager component introduced within the Job Execution Service (JES) is now responsible for collecting the information pertaining to the QoS parameters in the contracts offered by the JES, and constructing SLA templates that are published in a discovery service together with contract compliance history and other reputation and trust building information.
4. Mirroring the administrator agent design, we introduce a user task agent responsible for executing job-submission-related tasks on behalf of end-users. The user describes a set of jobs and their resource requirements in a language such as JSDL and passes it into the agent together with some higher-level goals expressed in terms of policies applying to the task. For example, all the jobs must finish before a certain time and must not cost more than a budget limit. If a job fails to meet its SLA, then the agent will receive a notification and resubmit or migrate the job to another provider automatically. If the agent receives warnings in the form of violation predictions, it may try to renegotiate contracts to still meet the higher-level goals of the end-user.
5. The agent is also responsible for selecting an appropriate set of resource providers (clusters) by querying the discovery service, SLA Selection Service. A subset of the clusters may be selected with which individual call-for-proposal based bid and contract-negotiation interactions are started. For instance, the price may not be available in the SLA Selection Service, but may need to be dynamically negotiated just before submission time so the current load on the respective resource can be taken into account.
6. The end-user agent drives the negotiation interaction with the clusters selected in the previous step and signs agreements on behalf of its user. The resource administrator agent signs the same contract on behalf of the resource administrator. The resource administrator agent is responsible for simplifying and automating resource configuration and learning-capable adaptation based on higher-level resource administrator goals. The resource administrator agent controls the SLA manager component of JES and indirectly sets policies that determine the price of resources and the job prioritization policies. A sample account (VO) SLA is shown in Figure 4.
7. The price information is then used by the reservation manager as well as by the negotiation component to drive the appropriate quota reservation and enforcement interaction with the accounting service; and to give end-users service-level, price-quote offers in the SLA templates respectively.
8. The job is submitted to the local cluster scheduler with the negotiated priority and other job specific properties.
9. When the job has completed, the JES SLA manager must calculate the final charge of the job based on the service level delivered and possible SLA violations, obligations, and penalties. The outcome is then stored locally in the resource administrator agent's knowledge repository to adapt the existing configuration and policies to better meet the resource administrator goals. The accounting information is logged in the usage tracking service and the appropriate account is charged; the reservation committed and removed; the consumed quota withdrawn from the account; and finally a transaction record is logged in the accounting service.

10. The accounting service SLA manager can now access the usage information and the transaction record to update its policy and configuration to better meet the higher-level goals of the account administrator.

```
account_sla = { parties,
                monitoring,
                qos_parameters,
                guarantees,
                constraints }
parties = { account_administrator,
            project_leader,
            project_members }
project_members = Σ project_member
monitoring = { monitoring_services, report_specification }
monitoring_services = { account_service, logging_service }
report_specification = { report_interval, report_parameters }
report_parameters = { overdraft,
                      allocation,
                      distribution,
                      job_usage_record }
qos_parameters = {account_service.account_data,
                  logging_service.usage_record }
guarantees = {overdraft < 10,
              allocation = 360000,
              distribution_type = staggered,
              distribution_interval = monthly }
constraints = { start_time = 2005-01-01:10:00,
                end_time = 2005-07-01:10:00 }
```

Figure 3: Sample Account SLA

```
resource_sla = { parties,
                 monitoring,
                 qos_parameters,
                 guarantees,
                 constraints }
parties = { resource_administrator,
            project_member }
monitoring =  resource_information_service
qos_parameters = resource_information_service.jobs
guarantees = { priority = high,
               cpu_time = 36000,
               completion_time = 2005-02-01:10:00,
               cost = 36000,
               charge_account = snic-01-01 }
constraints = { start_time = 2005-01-01:10:00,
                end_time = 2005-02-02:10:00 }
```

Figure 4: Sample Resource SLA

5 SLA Manager Architecture

The SLA Manager component that we propose for making Web and Grid services SLA-aware in a non-intrusive way is described in some more detail in this section. The manager can be divided into six subcomponents 1) SLA Negotiator (SLAN), 2) SLA Monitor (SLAM), 3) SLA Event Sink (SLAS), 4) SLA Policy Manager (SLAP), 5) SLA Rating Engine (SLAR) and 6) SLA Task Manager (SLAT).

The subcomponents are governed by a couple of general-purpose services, the Decision Making System (DMS) for pluggable decision making engines such as AI goal-driven, rule-based engines, and the Component Bus (CB) for efficient asynchronous inter and intra component communication. Figure 5 depicts this architecture and the most important interactions. All of these components are designed with clearly defined interfaces and interactions to make it easy to replace a subcomponent with a custom implementation.

5.1 Decision Making System (DMS)

The DMS service exposes an abstract set of interfaces that can be used to communicate with arbitrary decision making software plugged into the SLA framework. Figure 6 gives some examples of various DMS implementations that might be integrated.

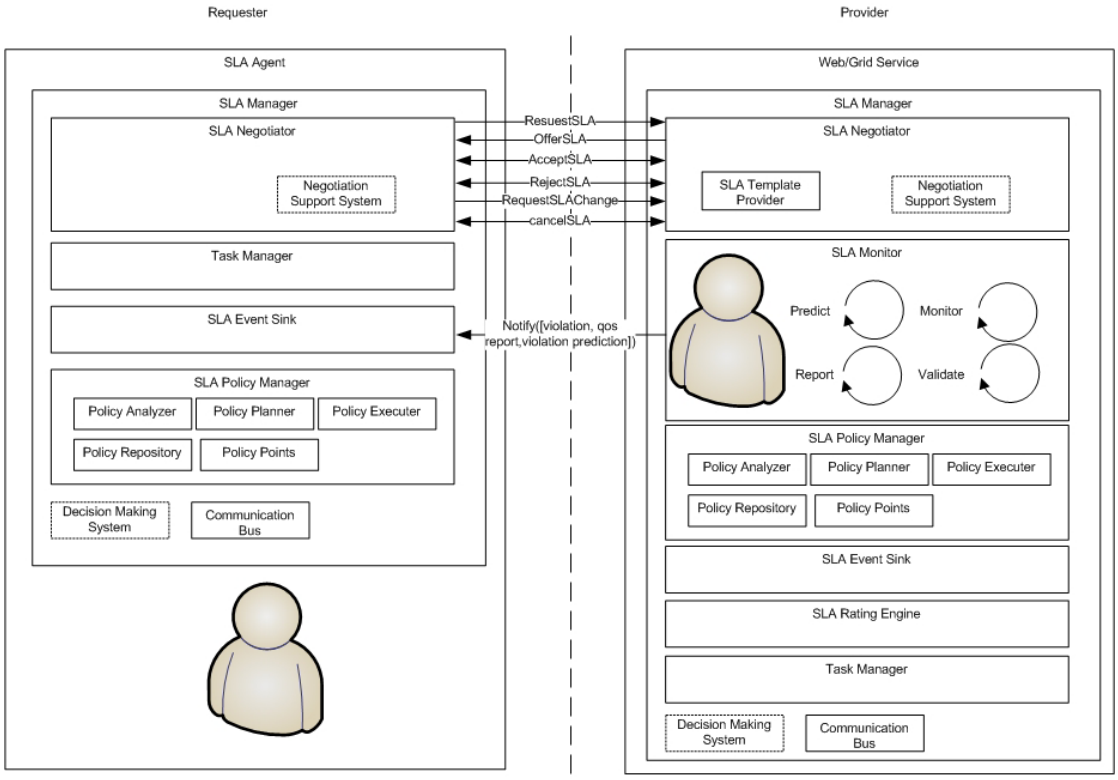

Figure 5: SLA Manager Architecture

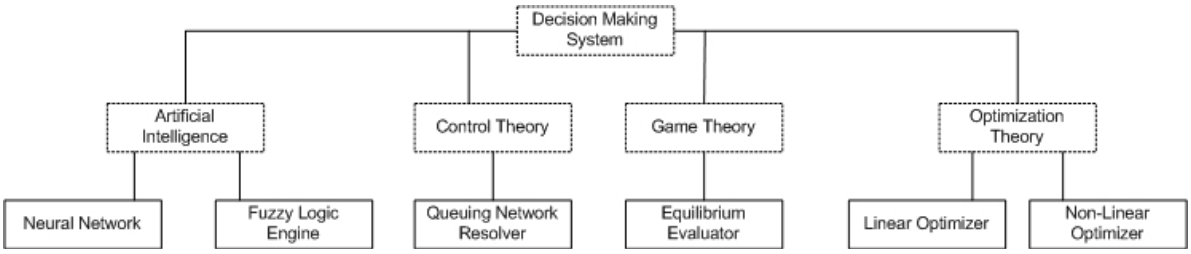

Figure 6: Decision Making Systems

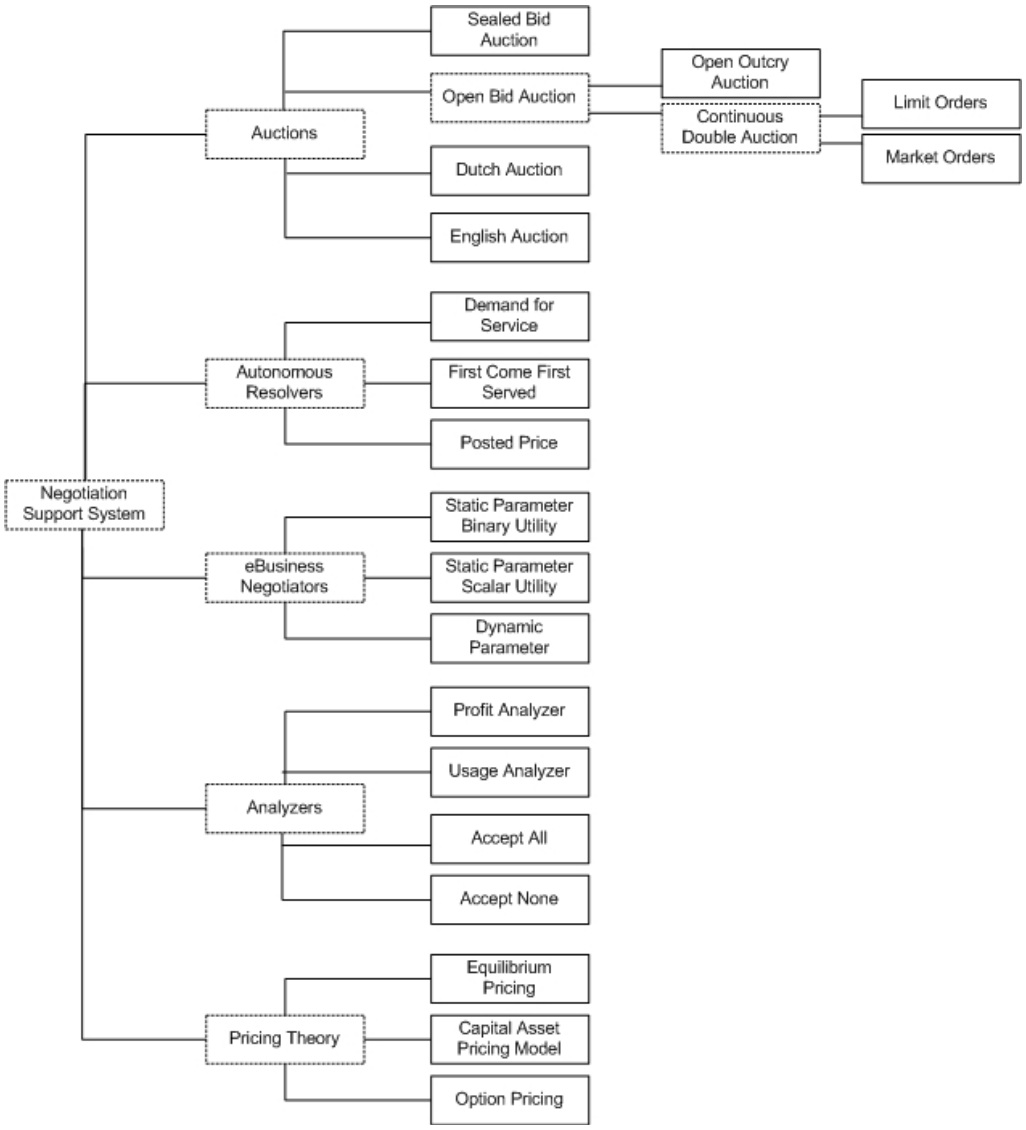

Figure 7: Negotiation Support Systems

5.2 Component Bus (CB)

The different SLA manager sub-components need a consistent and efficient way of communicating with each other asynchronously. Similarly, distributed SLA Managers need to be able to communicate with each other, for example, to negotiate contracts or share provider reputation knowledge or usage reports. The Component Bus hides the protocol specifics from the rest of the SLA Manager to make it easier to switch between different transport bindings and implement reliable messaging on behalf of the component. Protocol details such as WSDL, SOAP, WSRF, and WSDM specifics are also encapsulated in this service. The set of WSDL interfaces supported are, however, published to make it easy to write custom, remote components that communicate with this service.

5.3 SLA Negotiator (SLAN)

The negotiator subcomponent is responsible for constructing SLA templates that can be offered in bid publications. It must therefore collect enough information from other subcomponents in order to offer SLAs that are possible for the provider to attain. During a point-to-point negotiation phase more information can be collected about the other endpoint in the negotiation, and usage history may be taken into consideration. The interaction consists of 6 message primitives with payloads defined by the ontology in effect, which in turn is determined by the Negotiation Support System (NSS). The interaction is carried out between two SLAN components and the primitives are:

- **RequestSLA**. The service user issues a call for proposal to service providers
- **OfferSLA**. The service provider sends an offer to one or more service requesters
- **AcceptSLA**. Either the provider or the requester may confirm acceptance of the SLA. As a result the contract is signed by the party sending the message
- **RejectSLA**. Either the provider or the requester may reject the SLA offered or requested
- **RequestSLAChange**. The requester can try to change a previously negotiated SLA, for example, to request fewer resources to get discounts. Provider and NSS policies may determine whether the requester is allowed to renegotiate after the SLA has been signed and other temporal restriction. One policy might be that SLAs cannot be renegotiated but must be cancelled, and then a new SLA has to be negotiated from scratch. Our model does not support provider-initiated renegotiation since this is modeled through penalties in the SLAs.
- **CancelSLA**. This message could be used both by the requester and the provider to cancel a request or an offer. This is typically done before the contract has been signed but some policies and negotiation algorithms may allow SLAs to be canceled at any time to avoid wasting time on enforcing SLAs that one or more parties have lost interest in.

NSS is designed akin to the DMS described in Section 5.1. Figure 7 shows anticipated Negotiation Support System implementations.

5.4 SLA Monitor (SLAM)

The SLA Monitor subcomponent is responsible for automatically measuring, monitoring, validating, and triggering reports and violation notifications according to the SLA specification. The only input to this component is the signed SLA, which thus must be self-contained and possible to validate automatically to detect violations. The SLA Monitor may also provide some prediction functionality. Forecasts on possible future violation risks may be compiled based on usage history and sent out to the SLA parties to take countermeasures. The predictions may also be used internally in the SLA Manager to modify configuration and policies to avoid violations. Although this component in reality may interact with legacy information-provisioning, back-end services, it must be transparent to the implementation in order to make the component easy to deploy in any infrastructure. Wrappers may for instance be written, whose endpoints are made available in the SLA.

5.5 SLA Event Sink (SLAS)

Outbound messages are put on the CB. Incoming messages on the bus are, however, forwarded to the SLA Event Sink where the filtering occurs to find the correct sub component configuration specifying how the event should be treated. Automated actions may, for example, be specified to trigger a chain

of events or to take counter measures automatically in case of violations. The SLAS may internally be implemented as a JMS or MQSeries enabled event channel offering component specific QoS.

5.6 SLA Policy Manager (SLAP)

At the heart of autonomous computing is the MAPE control loop (Measurement, Analysis, Planning, Execution) and the Policy-based management architecture with policy decision and enforcement points. The SLA Policy Manager subcomponent marries these two architectures and maintains a policy repository with the most recent policies, which are continuously updated as a result of further usage and SLA compliance history analysis.

5.7 SLA Rating Engine (SLAR)

The rating engine is responsible for setting the price on resources and services provided, and for calculating discounts and rates applicable. It should be able to give both price quotes as well as to calculate the actual price of a delivered service to be charged in a billing or accounting system. The inputs to the engine are the price plan, the usage record, and the SLA.

5.8 SLA Task Manager (SLAT)

The final subcomponent is the SLA Task Manager, which embodies the high level goal that is to be achieved, as well as a schedule for how to achieve it, defined in some workflow engine interpretable language. Clients may group a set of jobs to be executed in a task, with well-defined criteria and actions to handle faults and perform automatic migration. The account manager discussed in Section 3 may set up a reoccurring, cron-job like, task to distribute granted allocations to research projects on a monthly staggered basis. The SLAP, discussed in Section 5.6, may use the task manager to execute its plans to maintain and attain service levels committed in the SLA being managed.

6 Conclusions and Future Work

We now refer back to the requirements summarized in Table 1 to highlight how they were addressed in our SLA solution.

An account administrator agent was introduced to simplify and automate management of accounts. This involves monitoring usage to adjusted policies automatically as well as setting up repeating tasks such as account allocations. Services can be selected based on usage by means of the SLA selection service exposing SLA templates that may be negotiated. Both the construction of the templates as well as the point-to-point negotiation may involve consulting the usage records. If users misconduct and submit more jobs than they were granted on the resources all the stakeholder agents, the user, the resource, and the account administrator, automatically detect the violation and can react upon it. The resource may stop jobs from being submitted or submit them into a low priority queue, alternatively change the price of the resource for that user in future negotiations. The account administrator may stop the particular user from using up shared account quota for a limited time. Finally, the user will also get a notification and can then choose to either pay more for the same service to maintain the job flow, or decrease the submission flow. By the introduction of decision making engines capable of inference reasoning over rule-bases, high-level policy rules may be entered into the agent knowledge bases to optimize certain QoS criteria in response to system events and to set more fine-grained policies and configurations automatically. The most important benefit from the SLA framework is, however, the formal signing of agreements, which may be used as a more controlled way of using resources at certain guaranteed service levels at a certain price. The agreements allow all stakeholders to merge their preferences to achieve the best overall system performance, utilization, and usability.

Future work includes implementing the contracts using WS-Agreement and the management infrastructure on top of WSRF and WSDM. For automatic metrics measurements we also intend to utilize WSLA. Fuzzy Logic rule engines and OWL-based inference engines and usage-based analyzers will provide the first candidates for the initial prototype of the Decision Making System and Negotiation Support System incarnations. Simulation test-beds will be developed allowing statistical

measurements to be performed in a controlled environment to fine-tune the framework and the policies before deploying it in the SweGrid production Grid.

Acknowledgments

I would like to thank my colleagues Lars Malinowsky, Peter Gardfjäll, and Olle Mulmo for many discussions related to the topics discussed in this paper. I would also like to thank Sandra Brunsberg, and my supervisor Lennart Johnsson for reviewing the paper.

References

[1] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss, "Overview of the I-WAY: Wide Area Visual Supercomputing," *International Journal of Supercomputer Applications*, vol. 10, pp. 123-130, 1996.

[2] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, 1999.

[3] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, 2 ed: Morgan Kaufmann, 2003.

[4] F. Berman, G. Fox, and A. J. G. Hey, *Grid Computing: Making The Global Infrastructure a Reality*: John Wiley & Sons, 2003.

[5] J. Joseph, M. Ernest, and C. Fellenstein, "Evolution of grid computing architecture and grid adoption models," *IBM Systems Journal*, vol. 43, pp. 624-645, 2004.

[6] J. Carolan, S. Radezsky, P. Strong, and E. Turner, *Building N1 Grid Solutions Preparing, Architecting, and Implementing Service-Centric Data Centers*: Sun BluePrints, 2004.

[7] J. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, pp. 41-50, 2003.

[8] S. Blake, D. Black, M. Carlson, E. Davis, W. Zheng, and W. Weiss, *RFC 2475: An Architecture for Differentiated Services*: IETF, 1998.

[9] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, *RFC 2205: ReSerVation Protocol (RSVP) version 1 functional specification*: IETF, 1997.

[10] K. Nahrstedt, H. Chu, and S. Narayan, *QoS-aware resource management for distributed multimedia applications*: IOS Press, 1998.

[11] T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo, "An OGSA-Based Accounting System for Allocation Enforcement across HPC Centers," presented at 2nd International Conference on Service Oriented Computing, New York, NY, 2004.

[12] E. Elmroth, P. Gardfjäll, O. Mulmo, and T. Sandholm, "An OGSA-based Bank Service for Grid Accounting Systems," *Applied Parallel Computing. State-of-the-art in Scientific Computing. Lecture Notes in Computer Science. (to appear) Springer Verlag.*, 2004.

[13] A. Barmouta and R. Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration," presented at International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, 2003.

[14] S. Jackson, "QBank: A Resource Management Package for Parallel Computers," Pacific Northwest National Laboratory, Washington, USA 2000.

[15] J. Hendler, T. Berners-Lee, and E. Miller, "Integrating Applications on the Semantic Web," *Journal of the Institute of Electrical Engineers of Japan*, vol. 122, pp. 676-680, 2002.

[16] D. De Roure, N. R. Jennings, and N. R. Shadbolt, "The Semantic Grid: A future e-Science Infrastructure," in *Grid Computing - Making the Global Infrastructure a Reality*: John Wiley and Sons Ltd., 2003, pp. 437-470.

[17] R. Housley, W. Ford, W. Polk, and D. Solo, *RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile*: IETF, 1999.
[18] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Cordinating Resource Management in Distributed Systems," *Lecture Notes in Computer Science*, vol. 2537, pp. 153-183, 2002.
[19] D. D. Lamanna, J. Skene, and W. Emmerich, "SLAng: A Language for Defining Service Level Agreements," presented at The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03), 2003.
[20] A. Dan, E. Davis, R. Kearney, A. Keller, R. P. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and Y. A., "Web services on demand: WSLA-driven automated management," *IBM Systems Journal*, vol. 43, 2004.
[21] H. Ludwig, A. Dan, and B. Kearney, "Cremona: An architecture and Library for Creation and Monitoring of WS-Agreements," presented at 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York, 2004.
[22] O. Smirnova, P. Eerola, T. Ekelöf, M. Ellert, J. R. Hansen, A. Konstantinov, B. Kónya, J. L. Nielsen, F. Ould-Saad, and A. Wäänänen, "The NorduGrid Architecture and Middleware for Scientific Applications," *Lecture Notes in Computer Science*, vol. 2657, pp. 264-273, 2003.
[23] G. Wasson and M. Humphrey, "Toward Explicit Policy Management for Virtual Orginizations," presented at 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, 2003.