An OGSA-Based Accounting System for Allocation Enforcement across HPC Centers

Thomas Sandholm[1], Peter Gardfjäll[2], Erik Elmroth[2], Lennart Johnsson[1,3], Olle Mulmo[1]

[1] Dept. of Numerical Analysis and Computer Science and PDC
Royal Institute of Technology
SE-100 44 Stockholm, Sweden
+46-8-7907811
{sandholm, mulmo}@pdc.kth.se

[2] Dept. of Computing Science and HPC2N
Umeå University
SE-901 87 Umeå, Sweden
+46-90-7866986
{elmroth, peterg}@cs.umu.se

[3] Dept. of Computer Science and the Texas Learning and Computation Center, University of Houston
Houston, TX, 77204 USA
+46-8-7909275
johnsson@tlc2.uh.edu

ABSTRACT

In this paper, we present an Open Grid Services Architecture (OGSA)-based decentralized allocation enforcement system, developed with an emphasis on a consistent data model and easy integration into existing scheduling, and workload management software at six independent high-performance computing centers forming a Grid known as SweGrid. The Swedish National Allocations Committee (SNAC) allocates resource quotas at these centers to research projects requiring substantial computer time. Our system, the SweGrid Accounting System (SGAS), addresses the need for soft real-time allocation enforcement on SweGrid for cross-domain job submission. The SGAS framework is based on state-of-the-art Web and Grid services technologies. The openness and ubiquity of Web services combined with the fine-grained resource control and cross-organizational security models of Grid services proved to be a perfect match for the SweGrid needs. Extensibility and customizability of policy implementations for the three different parties the system serves (the user, the resource manager, and the allocation authority) are key design goals. Another goal is end-to-end security and single sign-on, to allow resources—selected based on client policies—to act on behalf of the user when negotiating contracts with the bank in an environment where the six centers would continue to use their existing accounting policies and tools. We conclude this paper by showing the feasibility of SGAS, which is currently being deployed at the production sites, using simulations of reservation streams. The reservation streams are shaped using soft computing and policy-based algorithms.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: distributed systems – *client/server, distributed applications.*

General Terms

Management, Design, Security.

Keywords

Grid computing, Grid accounting, Web services, OGSA, HPC, Security policy management

1. INTRODUCTION

Advances in network technology, in addition to the more distributed and collaborative nature of today's research projects, have prompted high-performance computing centers to improve the ease of use of their resources to a larger and more dispersed user base, as well as responding to the need for unified access procedures to collections of resources from multiple administrative domains. As a result, monolithic and esoteric systems, albeit more performance tuned, have had to make way for ubiquitous and open, standards-based solutions. It has became feasible to integrate the centers into Grids [22] that enable flexible resource sharing and load balancing across organizational boundaries.

Virtualization across management and security policy domains not only leads to a complex resource negotiation situation, but also makes it harder to track usage and enforce allocations. It is the latter issue that we address in this paper. We have developed an accounting system to enforce nationally allocated resource quotas across six high-performance computing centers in Sweden.

Key requirements on the accounting system include: soft real-time allocation enforcement based on resource usage collected from existing site schedulers; coordinated quota management across all clusters; uniform usage retrieval; policy negotiation and customization between user, resource, and allocation authority; and finally a flexible, policy driven, and standards-based authorization framework.

Our contribution and differentiator against existing accounting systems is fourfold: (1) we provide a decentralized accounting solution based on standard, open protocols in

This work was partly funded by The Swedish Research Council (VR) under contracts 343-2003-953 and 343-2003-954, the Royal Institute of Technology, and The Faculty of Science and Engineering, Umeå University.

compliance with the proposed Open Grid Services Architecture (OGSA) [21, 23], (2) we facilitate 3-party (user, resource, allocator) policy customization, (3) our system is non-intrusive to existing local site accounting systems and end-user tools, and thus offers light-weight deployment, and (4) all accounting components are governed by a scalable cross-organizational authorization framework based on state-of-the-art Web services protocols.

The paper is organized as follows: Section 2 contains an overview of recent standards efforts in the field of wide area distributed computing relevant to Grid computing. The SweGrid network and its accounting requirements are outlined in Section 3. In Section 4, we present some existing accounting systems and architectures, and we discuss why they do not meet the SweGrid needs. Section 5 describes the SweGrid accounting system design and Section 6 the implementation. Section 7 presents some results from simulations of reservations against our system. Finally, in Section 8, we summarize our contribution and our future research and development plans.

2. OGSA AND WEB SERVICES

OGSA was developed in order to solve the complex task of sharing and integrating fine-grained heterogeneous resources distributed across security domains in a wide area network. The architecture combines the elaborate control mechanisms of mainframe systems with ubiquitous Web and Internet technologies. Key concepts include virtualization and discovery of resources based on service-oriented interactions. It can be seen as the Web Services Architecture (WSA) [15] applied to Grid computing. Another key aspect of OGSA is the management of distributed state, such as discovery, introspection, notification, and lifetime management. Although Web services by design are considered stateless for scalability and decoupling reasons, state needs to be managed to control shared resources in an application and client agnostic way and thus enabling interoperable state-aware interactions. Decoupling is a sound design goal for managing change, but in highly dynamic systems there is often a trade-off between fine-grained control and adaptation, and too strict enforcement of abstraction and decoupling. The core Web services specifications such as SOAP [31] and WSDL [17] do not address this problem. The often quoted REST [20] architecture for interacting with resources only solves the problem partially by putting the burden of maintaining state on clients or client agents, and is further targeted towards large-grain hypermedia transfers, and thus very limited in its scope. REST is a very similar approach to the one taken by Web services workflow languages, such as BPEL4WS [11]. Request-broker-influenced specifications [37], however, address client agnostic fine-grained resource state sharing. We based our work on the broker model, because it also fits better with existing programming language technology.

Web services protocols all use XML as a foundational building block, and therefore are convenient for self-describing, document-centric interactions (as opposed to the less flexible API-centric model) often used in large-scale integration environments with little or no control over the participating parties' implementation policies.

In OGSA-based Web-services environments, the complexity of setting and applying policies to optimize the user quality of service, as well as resource utilization leads to the need for Service Level Agreement (SLA) management. Agreement, and negotiation protocols such as Global Grid Forum's emerging WS-Agreement [12], and FIPA's Contract Net [3] protocols are example technologies addressing that need.

3. SWEGRID

SweGrid is a national computational resource, initially joining together one cluster at each of six high-performance computing centers across Sweden, and currently comprising 600 nodes. The clusters located at the individual sites are interconnected with the 10 Gb/s GigaSunet network. The sites also operate several other resources for computation and storage, and they have developed their own security- and accounting systems over time to serve local needs and the requirements following from different sources of funding. SweGrid job submissions are currently done using the Globus Toolkit [4] or the NorduGrid [35] job submission tools, interfacing cluster-level schedulers at the local sites. Compute time on the SweGrid resources are allocated to research projects by the Swedish National Allocations Committee (SNAC), akin to the NRAC (National Resource Allocations Committee) in the US. Projects within the Swedish science community and with the appropriate needs and promising research may apply for SNAC allocations. The allocations are currently made in node hours, and the decisions are made after a scientific peer-review process evaluating the research proposals. Prior to interconnecting the HPC centers in a Grid, allocations were targeted at individual clusters, and the prospective research participants would have to acquire valid user accounts at each of the centers at which quotas were awarded to be able to run their jobs. This manual and static allocation thus not only caused sub-optimal job-to-resource mappings, but further led to large administrative overhead. SNAC is hence now allocating quotas to the SweGrid as a whole. This, however, has a large impact on how accounting is done, because it is thereby not enough to just do local site accounting, and quota enforcement. The allocation enforcement must be coordinated across all sites, and the sites must be able to produce usage records that comply with each other.

Thus a real-time enforcement solution is required in order to make resource-mapping decisions *a posteriori*, considering current user policies, resource policies, and allocation-authority policies. The resource may, for instance, allow jobs lacking sufficient quota to be run and put in a low priority queue if the current utilization is low. The user may on the other hand only want to run the job if there are sufficient funds, and finally some allocation authorities (e.g., SNAC or project leaders) may not allow jobs to go through from certain users who have used up a large chunk of a common project quota. This three-way negotiation needs to be flexible enough to allow various parties to configure their system according to local policies.

Even though SweGrid currently consists of a fairly homogenous compute farm with similar middleware installations, it is expected that both the hardware and software solutions may evolve and become distinctly heterogeneous in the future, as more resources are added. The SweGrid accounting system must hence be non-intrusive to the existing systems, i.e., easy to deploy or plug into existing infrastructure, without replacing the local accounting and scheduling systems.

4. GRID ACCOUNTING

Cluster-targeted scheduling systems as well as operating systems commonly have built-in accounting systems to track resource usage. However, they often assume a homogenous run-time environment, and they lack standard ways to obtain and coordi-

nate information among several heterogeneous clusters. Thus there has been a strong need for Grid accounting systems that integrate local accounting solutions similarly to the way Grid meta-schedulers and co-allocation managers coordinate, and administer job submissions across several schedulers. Some general issues that need to be solved by distributed accounting systems on the Grid, including the need for a standard usage record format, are outlined in [36].

In the European Data Grid Accounting System (DGAS) [24], users need to pay for resources they use in a fictive or virtual currency called *Grid Credits*. Resources earn Grid Credits when they offer their services to users, thus stimulating market-economy driven resource sharing. All currency transactions are mediated by decentralized bank services. The implementation is tightly coupled to the DataGrid workload manager software, and thus hard to deploy without affecting the local cluster software environment.

GridBank [13] is provided as an extension to the Globus job manager, and it calculates job cost based on standard XML usage records. It is thus not as intrusive as DGAS, but it still requires modifications of a particular workload manager. An interesting feature of GridBank is that it makes use of decentralized Trade Servers to negotiate resource prices. GridBank is also modeled around an economy-driven workload management [8] system utilizing resource price matrices. Neither GridBank nor DGAS are based on open, standard Grid protocols such as Web services or OGSA, thus limiting their prospective scope of interoperability with other Grid systems.

The Grid Economic Services Architecture (GESA) [32] specified by the Global Grid Forum (GGF), presents an OGSA-based architecture using the concept of chargeable services. When developing a service one may associate it with a cost that can be charged in a bank based on standard usage records. GESA is, hence, quite intrusive to the service since it requires the service interface to be changed in order to charge for its usage. Furthermore, GESA was designed to be orthogonal to the security model chosen, and does not address the security issues related to accounting.

SNUPI [25] provides extensions to the Linux operating system, and it allows cluster usage data to be collected and stored in RDBMS databases, and then queried from user-friendly portal Web interfaces. SNUPI is however not service-oriented and assumes a homogenous cluster environment.

QBank [28] is a resource allocation management system developed for parallel computers. Its successor, Gold [27], adds more advanced accounting features such as price quotes, funds transfers, and timestamped allocations. Gold also allows role-based authorization and transaction journaling. Although it would fulfill most of the core accounting needs discussed in this paper, it is not developed using open, Grid, or Web services protocols, and is thus limited in its interoperability as well as cross-platform support. Its security model is also limited compared to our work.

5. SGAS DESIGN

We have developed the SweGrid Accounting System (SGAS) [5], with the aim of meeting the accounting needs of SweGrid presented in Section 3, and with a particular focus on shared quota enforcement across organizational boundaries, security, and simplicity of deployment. The accounting system is fully transparent, or imposes only marginal additional requirements, to the end-users, allowing for a smooth transition into an accounting-enabled Grid. In this section, we present the design rationales of the various system components.

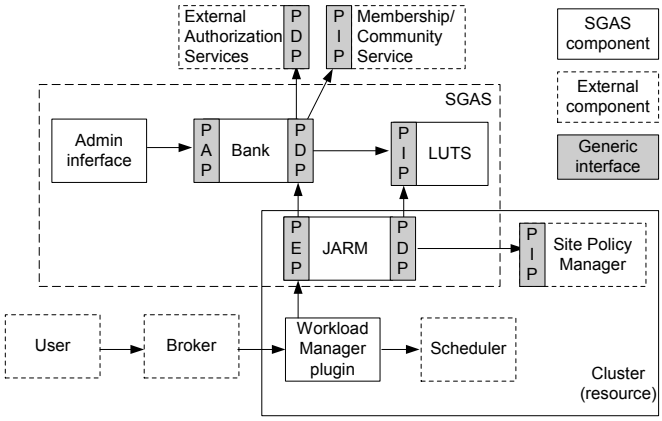

Figure 1. SGAS components overview.

We start by describing the flexible authorization framework (Section 5.1). Apart from a bank service (Section 5.2), which provides most of the accounting functionality, there is a workload manager integration component (Section 5.3), and a usage tracking service (Section 5.4). Figure 1 shows an overview of SGAS.

The operational flow is as follows: a user submits a job (potentially via a brokering service) to a workload manager service running on the resource. (We make use of a generic term here to stress that SGAS is a generic system that can be integrated with more than a single software stack.) The resource integration component intercepts the request by way of a workload manager plugin, and it interacts with the bank to reserve sufficient quota. This interaction is further explained in Section 5.2 through 5.4.

5.1 Authorization Framework

Three parties are involved in our accounting scenario: the user, the resource, and the allocation authority (front-ended by the bank). Our system has multiple decision points at various levels, allowing for both policy overlay (combining policies from multiple sources when making a decision) as well as retention of local control. This allows us to honor the requirements of all three stakeholders, as well as facilitating decentralized control and system management.

We make use of the terminology and overall architecture as proposed by the Global Grid Forum working groups on Authorization Frameworks and Mechanisms [30], and OGSA Authorization [38]. We allow access permission policies to be specified in XACML (eXtensible Access Control Markup Language) [10]. Figure 2 shows a simple example policy in XACML, governing what set of users may use a certain allocation in the bank. While the implementation makes use of XACML, we emphasize that the framework allows for any policy language understood by the pluggable authorization engines. To illustrate this we have successfully experimented with Delegent [34], an authorization service capable of rights management delegation (not supported in XACML) as an alternative back-end authorization service, or Policy Decision Point (PDP), for the bank.

Multiple information providers are used in the system. The bank, for instance, may be configured to associate any user in a particular Virtual Organization (VO) with a particular account. To

```
<Policy PolicyId="SweGridTestProjectPolicy"
        RuleCombiningAlgId=
            "urn:oasis:names:tc:xacml:1.0:rule-
combining-algorithm:permit-overrides">
<Target>
 <Subjects><AnySubject/></Subjects>
 <Resources><AnyResource/></Resources>
 <Actions><AnyAction/></Actions>
</Target>
<Rule RuleId="RequestHoldRule" Effect="Permit">
 <Target>
  <Subjects>
   <Subject>
    <SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:stri
ng-equal">
     <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">
/O=Grid/O=NorduGrid/OU=pdc.kth.se/CN=ThomasSandholm
     </AttributeValue>
     <SubjectAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:s
ubject-id"/>
    </SubjectMatch>
   </Subject>
   <Subject>
    <SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:stri
ng-equal">
     <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">
/O=Grid/O=NorduGrid/OU=cs.umu.se/CN=PeterGardfjäll
     </AttributeValue>
     <SubjectAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:s
ubject-id"/>
    </SubjectMatch>
   </Subject>
  </Subjects>
  <Resources><AnyResource/></Resources>
  <Actions>
   <Action>
    <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:stri
ng-equal">
     <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">
      requestHold
     </AttributeValue>
     <ActionAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:ac
tion-id"/>
    </ActionMatch>
   </Action>
  </Actions>
 </Target>
</Rule>
<Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Figure. 2. Example of a Bank Account policy granting withdrawals for two project members, encoded in XACML.

achieve this, external services such as VOMS [9] and CAS [33] are used to gather membership evidence (Policy Information Point, PIP).

The allocation authority adopts a delegated security model, controlled by policies that can be associated within each research project. The highest level of authorization authority is the national allocations committee, which allocates quotas to projects/VOs. On a project level, the principal investigator (PI) can specify additional policies through the Policy Administration Point (PAP), to allow various project members to use the quota. To enable flexible self provisioning, the PI may additionally give away a possibly restricted subset of its own management privileges to other members.

Allocation requests and decisions are authenticated and integrity protected by the use of XML digital signatures [14], and/or WS-SecureConversation [18]. In addition, when the resource contacts the bank, both the user's delegated credentials associated with the requested job (made available by the workload manager plugin), as well as the resource's own credentials, are used to authenticate the allocation request.

The resource checks the quota on a soft real-time enforcement basis. The check is soft (as opposed to strict) in that policies on the client, as well as on the resource, can allow the job to be run even if the allocation authority decides that a sufficient quota is not available. This is a critical requirement from the HPC centers, because the allocations are done periodically (for 6 or 12 months) whereas user resource usage tends to be bursty (e.g., just before the scientists are to publish a paper, usage goes up). Additionally, the users may specify that they only want to execute long-running jobs on resources that allow the jobs to complete within the available quota limit, and the resource may disallow jobs without enough available quota during peak utilization periods. Such usage-based allocation decisions can be made by querying the usage service, and by allowing the resource Policy Enforcement Point (PEP) to shortcut the bank authorization (modulo local site manager configuration), and overrule the final decision.

5.2 Bank

The bank component is central to the design of SGAS. It implements coordinated quota enforcement across all the SweGrid sites. The component consists of three OGSA-compliant services, the `Bank`-, the `Account`-, and the `Hold`- services. The bank design is presented in some more detail in [19].

The `Bank` service is responsible for creating and locating `Accounts`, corresponding to a research-project allocation. The `Account` service hands out soft-state, lease-based fund reservations called `Holds` to authorized `Account` members. `Account` members may be added or removed, or their rights may be modified through policies defined in XML. When a `Hold` is created, a specified amount of the total quota or funds is locked, meaning it may not count towards other reservations or withdrawals (c.f. making reservations on a credit card). The `Hold` is further only accessible by the party creating the `Hold`, typically the resource. The `Hold` can be renewed (its lifetime extended) and it can be committed (released). A commit operation will trigger an accounting transaction record in the `Account` that the `Hold` was held against. The amount reserved in a `Hold` does not have to match the amount committed, because they correspond to estimated vs. actual cost to run a job. It is up to the resource to decide whether a conservative overbooking reservation strategy should be applied to be sure that the job completes within the reservation time, or to be more optimistic and reserve a smaller amount that potentially can be renegotiated if the job did not manage to complete in time. The cost and the

allocations are expressed in a virtual currency, Grid Credits, and may thus be mapped into any physical resource-specific cost. Typically, the cost is mapped directly to wall-clock time, because it makes it easy for the existing HPC centers' scheduling infrastructure to enforce as well as to measure the quota. How physical costs should be mapped into Grid Credits is, however, something that is decided by resource policy. A resource may, for instance, use a standard usage record and give the various containing attributes weights used to calculate the total cost. The typical wall-clock approach can thus be seen as giving the wall-clock attribute the weight of 1 and all other attributes the weight of 0. Another advantage of the wall-clock mapping is that it becomes intuitive for users to set a maximum wall-clock time attribute, which corresponds to the granted SNAC time, in their job specifications using, e.g., the Globus [4] Resource Specification Language (RSL). They thereby initiate an implicit in-blanco signing process with the resources. Figure 3 shows the design of the bank, and Figure 4 depicts a common resource and bank interaction scenario. The interfaces shown should be seen as conceptual entities or roles of responsibility rather than programming language constructs. The interface technology used, typically WSDL or Java, depends on the distribution of the components. The bank is designed with a minimal set of data-centric operations to make it as easy as possible to interact with and to allow for future extensions at the same time. Security interfaces are clearly separated from application interfaces. This allows the security implementation to be easily customized or replaced without affecting the core bank implementation.

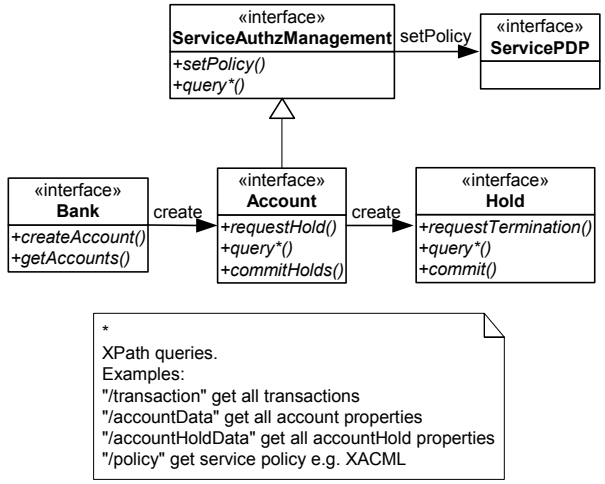

Figure 3. Bank interfaces.

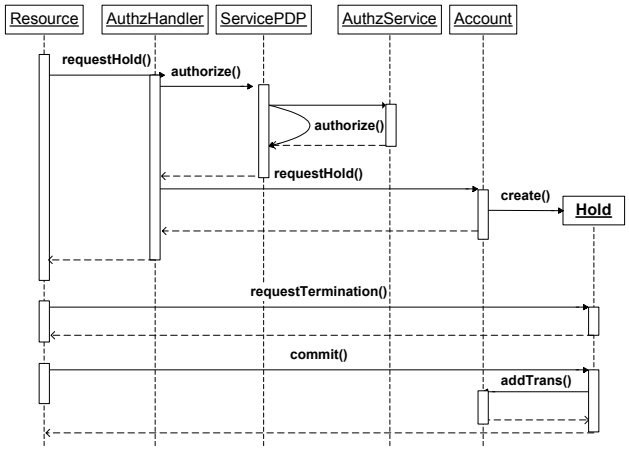

Figure 4. Bank and resource interactions.

5.3 LUTS

The Logging and Usage Tracking Service (LUTS) is used to store usage records compliant to the GGF Usage Record (UR) XML format [29]. Depending on who should have access to the service, resources may share the same LUTS in order to allow users to query for detailed information regarding the resources consumed by their jobs across multiple sites. The query language is XPath-based [7] and thus very flexible and extensible. LUTS is schema agnostic, which means that the UR may be extended with information, such as job tracking information, that a particular subset of resources and users understands without having to change or reconfigure LUTS. A batch of Usage Records may be logged at the same time to improve performance and scalability. The service builds on the same security infrastructure as the other SGAS services allowing, for instance, dynamic access control permissions to be set up specifying who is allowed to query or publish data in the service, and allowing message payloads to be encrypted and/or signed.

5.4 JARM

The Job Account Reservation Manager (JARM) component is responsible for integrating local cluster systems into SGAS. JARM intercepts a job submission and calculates the estimated cost of the job based on, for example, the user's job specification (using RSL in our case), and current system load. It then contacts the appropriate `Account`, which is either specified in the RSL by the client or alternatively searched for in the `Bank`. A `Hold` is created with the estimated cost, and the timeout of the `Hold` is set to the estimated duration of the job plus a margin. The resource also lets the Bank know whether overdrafts are accepted, a policy that may be requested by the client. If the `Hold` was created successfully, JARM lets the local workload manager continue with the job submission; otherwise, an error is generated and logged.

After the job has completed, JARM collects the usage information, converts it into the standard GGF UR format, logs it into LUTS, calculates the actual cost of the job, and commits the `Hold` (which is then destroyed). All this typically happens in batch mode, asynchronously in regard to the job submission, to induce as little overhead as possible to the user-perceived response time. In addition, it allows for higher throughput at moments of peak load. A Site Policy Manager implementation can easily be customized for particular workload managers and site policies. Note that JARM shields the Site Policy Manager from knowledge about the bank system (see Figure 1). A generic NorduGrid Site Policy Manager has currently been implemented.

SGAS is mainly concerned with allocation enforcement, and because it is workload-manager agnostic, scheduling and brokering functionality is outside of its scope. However, we recognize

that economic brokering algorithms based on a thorough analysis of economic models and business needs belongs to the future of both scientific and industrial Grids, and that the use of cyber money as well as virtual money is going to be a future requirement. We therefore provide plug points for calculating, setting, and publishing the price in the Site Policy Manager component. Note that this does not mean that the resources need to decide on appropriate prices in isolation to the rest of the system. Trading and pricing services as described in [16, 24] may, for instance, be used. The use of cyber money or real money in conjunction with Grid Credits, is in SGAS best done at the allocation authority level, where Bank services may charge real money for filling up accounts with quotas.

6. SGAS IMPLEMENTATION

In this section we present experience gained from implementing the accounting solution described in the previous section.

6.1 Implementation Approach

For interoperability reasons, the SGAS design is based on the latest Grid and Web services protocols. In our implementation, we wanted to go one step further by reusing toolkits implementing these standards. The general approach taken was to compose the solution from standards-based toolkit primitives, as opposed to re-implementing low-level middleware or communication libraries. Apart from the obvious advantages of developing complete applications more rapidly and following the latest specifications closer, we also safeguard our solution against protocol changes in the standards, and we can leverage the interoperability testing done by the protocol implementers.

Reuse is done on three levels: development environment (e.g., build system), compile-time environment (APIs), and run-time environment (application server containers and system-level services). The first two are commonly applied by most projects, whereas the third is more common in the software industry than in academia. We focus our discussion here on run-time environment reuse in a Grid environment.

6.2 Container Framework

The Globus Toolkit (GT) [4] provides a Java-based container implementation of the GGF Open Grid Services Infrastructure (OGSI) protocol [37], a realization of the OGSA model. Both OGSI and GT are designed as a set of primitives that can be freely mixed, composed, extended, and embedded. OGSI facilitates cross-language interoperability, whereas the GT Java container provides a consistent, portable programming model. Below, we first summarize how the various OGSI concepts were leveraged in SGAS, and we then continue with describing how the GT container features were used to achieve this.

Soft-state management (server-side managed, client-lease controlled state) is commonly applied in both the Internet and Grid networks, and it is a fundamental component of OGSI. We control expiration and extension of `Holds` using the OGSI soft-state protocol. Service property introspection (with its associated query and notification framework), as a means to minimize brittle APIs for flexible information retrieval, is another key component of OGSI. We use this concept to query transaction records in the Bank and Usage Records in LUTS, and to get notifications when `Holds` are about to expire. OGSI specifies a factory pattern to create stateful resources in a uniform manner. The factory pattern is used to create both `Accounts`, and `Holds`.

GT allows code to be plugged into the container on three different levels: message, operation, and back-end storage. Message interceptors are mainly used for service-orthogonal functionality, such as transaction management and security. In SGAS, a GT-provided authorization-interceptor plugin is used to implement the interaction between the PEP in JARM and the PDP in the bank. Furthermore, mutual authentication, message encryption, and message signing, are all carried out by GT transparently to the application code in message handlers using generic implementations of WS-SecureConversation [18], XML-Encryption [26], and XML-Signature [14], respectively.

Operation providers allow decoupled implementations of parts of service interfaces. A service implementation is typically made up of a set of toolkit-supplied operation providers, and one or many application-specific providers. The providers are specified at deployment time, and thus promote a development model based on composition of primitives. All SGAS services (the bank services and LUTS) are made up of operation providers. LUTS is composed of GT supplied operation providers exclusively, and thus does not have any application-specific code or APIs. The unique behavior of LUTS is achieved by a back-end storage and query plugin that leverages an XML database implementation (Xindice [2]) and XPath (Xalan [7]) as a query engine. GT operation providers implement soft-state management, service creation, notification and inspection of service state transparently to the SGAS code.

6.3 Systems Integration and Scalability

Although the general design is to introduce as few new APIs as possible, there are a number of high-level APIs that may be used as a means to integrate SGAS with other systems. We expect other Grid services to be built on similar core OGSA fabric, and infrastructure components in the future, such as WSDL and WSRF. This in itself offers a baseline for low-level API interoperability that could be used e.g., by generic management tools. As an example, Globus service data browsers and monitors were used to manage the Bank and LUTS services. Further, the high-level Bank and Policy management APIs provided by SGAS and expressed in WSDL, serve as a public integration point to other accounting and authorization components. The Bank APIs are discussed more thoroughly in [19].

Simplicity and scalability are central to the SGAS design. SGAS should be able to scale down to very small, as well as to large-scale nation- and Grid-wide deployments. As a means to scale up, the load was balanced across many Bank and LUTS services. Additionally, charging and logging was done in batches with intervals customized to the overall system load.

6.4 Toolkits and Standards

We summarize the toolkits and standard protocols used to implement central features of SGAS in Table 1.

SunXACML [6] is used in the bank as a standard, self-contained PDP engine, which checkpoints policies to the Xindice database.

Some schedulers already have support for the GGF UR format, but for others we provide an XSLT [7] style sheet transformer based framework to simplify SGAS integration at local sites.

Table 1. Toolkits and Standards

Toolkit	SGAS Feature	Standards Implemented
Globus	Service state inspection and notification, soft-state management, factory pattern	OGSI
Globus	Mutual authentication, credential delegation	GSI profile of WS-SecureConversation
Globus	Payload integrity, and privacy	XML-Signature, XML-Encryption
Xindice	XML database (for policy and service state)	XML:DB
Xalan	Query engine, stylesheet (Usage Record) transformation	Xpath, XSLT
SunXACML	XACML PDP	XACML
Axis [1]	Web services engine	SOAP, WSDL
SGAS	SGAS Usage Records	GGF XML Usage Record

Table 2. Simulation Setup

Simulation Property	Time (s)
Fair Submit Interval	10
Unfair Submit Interval	5
Reservation	60
Allocation	300
Allocation Interval	50
Overdraft Penalty	60

Table 3. Fuzzy Variables

Fuzzy Variables	Fuzzy Values
Allocation proximity	recent, soon
Overdraft amount	low, high
Allocation left	little, much
Reservation	allow, disallow

7. RESULTS

In order to test the feasibility of the SGAS design we built a simulation framework aimed at measuring reservation throughput. The usage pattern that was simulated consisted of an allocation authority periodically adding new allocations to a bank account, and users making and committing reservations on that account continuously. The behavior was studied for two separate flows. First, a *fair flow* is a flow that is produced by a user, who does not try to make more reservations than is allocated to him/her within a given time period. Second, an *unfair flow* is a flow that is produced by a user, who tries to make reservations of twice the allotment. The unfair flow can be shaped using various policies and overdraft protection algorithms to optimize fairness and resource utilization. The first set of algorithms is based on the theory of fuzzy logic [39], whereas the second set is based on access control policy (XACML) rule conditions that may be set by account administrators. If the reservation may not be performed due to an overdraw violation, then there is a penalty in job execution time, simulating the job being put in a low priority queue by the scheduler.

It should be noted that both the fuzzy logic rule base, and the XACML policies used are mere examples of viable algorithms that may easily be applied using the SGAS customizability, and extensibility features. That is, the aim here is not to show an optimal algorithm, but rather to exemplify how a certain policy (overdraft protection in this case) can be implemented in SGAS.

Table 2 lists the configuration used in the simulation runs. Allocation interval refers to the time between two successive allocations, which in the SNAC case typically is 6 or 12 months (see Section 3 and 5.1). Overdraft penalty is the extra execution time added due to an overdraft that is not allowed.

All the simulations can be reproduced, and the source code can be obtained by downloading the SGAS Open Source distribution [5].

7.1 Fuzzy Overdraft Protection

A rule base was constructed of fuzzy variables to mimic the intuitive policies of account administrators. The set of fuzzy variables that were used are listed in Table 3. The allocation proximity variable denotes whether an allocation just occurred, or whether the next allocation will occur soon. The four rules comprising the overdraft policy were: (fuzzy values in italics)

R1: overdraft is *low* ∧ allocation left is *much*

⇒ *allow* reservation

R2: overdraft is *high* ∧ allocation left is *little*
⇒ *disallow* reservation

R3: allocation proximity is *soon* ∧ overdraft is *high* ∧ allocation left is *much*

⇒ *allow* reservation

R4: allocation proximity is *soon* ∧ overdraft is *low* ∧ allocation left is *little*

⇒ *allow* reservation

The decision is then a defuzzification of **R1** ∪ **R2** ∪ **R3** ∪ **R4.** I.e., if any of the rules above fire with a high certainty, the result will also equal the final outcome. Trapezoid truth functions were used in the fuzzification step, and the resulting values were defuzzified by using a mean of maxima algorithm.

Figure 5 compares the affect of the fuzzy overdraft policy applied to the unfair flow with a fair flow, and an unfair flow disallowing all overdrafts. The larger the area of the flow graph is, the worse is the reservation throughput, and the higher is the penalty execution time. The peaks of the flows represent overdrafts that were disallowed. Thus, the thinner the peaks are,

the more successful is the shaping of the flow. Note that the fair flow was shaped to get maximum throughput by avoiding overdrafts. The occasional peak at the beginning of the fair flow simulation was caused by the fact that the reservations and the periodic allocations were not started simultaneously, and thus the first allocation happened too late to avoid an overdraft. Over time, however, the allocations and reservations were synchronized. The periodicity of the peaks in the unfair and fuzzy flows corresponds to the available quota running low shortly before the new allocations are granted.

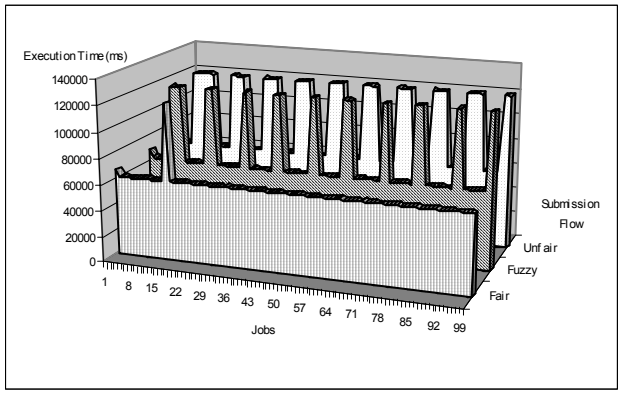

Figure 5. Submission flow simulation using fuzzy- and no shaping on fair- and unfair flows.

```
<Condition FunctionId=
  "urn:oasis:names:tc:xacml:1.0:function:integer-
less-than-or-equal">
  <Apply FunctionId=
    "urn:oasis:names:tc:xacml:1.0:function:integer-
one-and-only">
  <EnvironmentAttributeDesignator
    AttributeId=
      "sgas:overdraw:percent:requested"
    DataType=
      "http://www.w3.org/2001/XMLSchema#integer"/>
  </Apply>
  <AttributeValue DataType=
    "http://www.w3.org/2001/XMLSchema#integer">
     175
  </AttributeValue>
</Condition>
```

Figure 6. Example of overdraft policy allowing 75% overdraw.

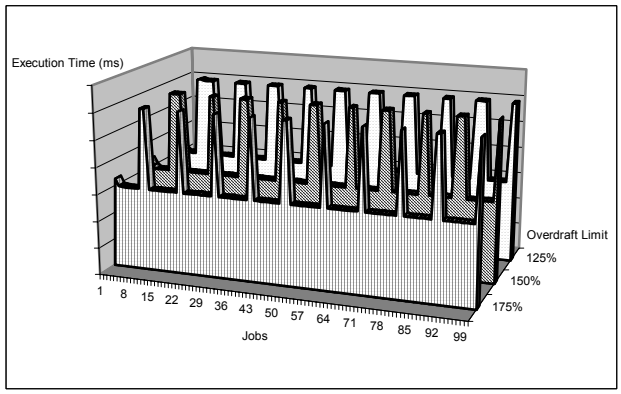

Figure 7. Policy-based overdraft protection simulation.

7.2 Access Policy Overdraft Protection

In Section 5.1, the authorization framework used by SGAS is discussed. It allows account owners to set policies regulating access to their accounts with XACML policies. An example policy condition is given in Figure 6. The actual value of the XACML attribute `sgas:overdraw:percent:requested` is calculated as:

$$(as + ar + rr) / ta,$$

where *as* is the allocation spent, *ar* is the allocation reserved, *rr* is the requested reservation, and *ta* is the total allocation. The value may hence be less then 100%. In that case reservations must not completely exhaust the total allocation available in order to be successful. The condition can be associated with any rule like the ones exemplified in Figure 2.

In our simulations we tested three different policies allowing 25, 50, and 75% percent overdraft against the unfair job reservation flow. The results can be seen in Figure 7. We note that a 50% overdraft policy roughly corresponds to the fuzzy overdraft policy in terms of job throughput, and as expected the 25% policy is close to the unregulated unfair flow, whereas the 75% policy is getting closer to the throughput in the fair flow.

8. CONCLUSIONS AND FUTURE WORK

We have presented an architecture, and an implementation of an accounting system based on open, standard Grid and Web services protocols to solve the resource quota-enforcement needs of a national-scale Grid network. Easy non-intrusive deployment, and integration with pre-existing, local accounting solutions prompted the use of XML document-centered communication and transformations and the use of a minimal set of APIs. This design is apparent in the policy administration API allowing arbitrary XML-specified policies to be defined for allocation decision points with a single operation. Another example is the non-existing API between the workload manager and the JARM component. It is designed as a message interceptor, obtaining its required input via runtime context and environment settings.

A customizable security model based on multiple PDPs, and PIPs, but with a single PAP and PEP, makes it possible to easily add new authorization services without affecting the service usage.

The 3-party policy negotiation design allows the resources to implement site-specific policies to optimize utilization and prioritize between users with different usage patterns and job-specification requirements. Furthermore, it enables allocation authorities such as SNAC, PIs, or individual project members to restrict the quota distribution according to dynamic policies.

The novel set of accounting features presented in this paper to solve the particular needs of SweGrid, and implemented in the SGAS system, do not exist in any other existing Grid accounting system to date. Although SGAS is primarily developed for SweGrid, it is based on open protocols, and has generic-enough functionality to be used in any Grid accounting setting.

The design is made general with respect to the type of mechanisms that are used for balancing load between resources or for achieving fairness between users. For example, the bank can be used in an environment driven by market-economy strategies where resources and resource brokers negotiate price and QoS

agreements solving the supply and demand problem. It fits equally well into a more planned-economy model where the main aim is to achieve fairness between users, based on given allocations to users or projects.

We intend to continue to improve this system mainly in two directions: (1) more sophisticated pre-allocation mechanism to allow, for instance, SAML assertions to be used as quota cheques for a collection of jobs, and thus limiting the bank interaction overhead of individual jobs, (2) use of more elaborate negotiation protocols such as Contract Net and WS-Agreement to handle Service Level Agreement (SLA) contract policing and obligation enforcement. With a more advanced negotiation protocol in place, we also intend to investigate soft computing, and game theory based decision-making procedures to automate SLA refinement.

The evolution of OGSI into the WS-Resource Framework is also something that we welcome and look forward to incorporate into our work, as it fits well into the state-management model of SGAS.

9. ACKNOWLEDGMENTS

We would like to thank our colleagues, Åke Sandgren, Lars Malinowsky, Michael Hammill, and Bo Kågström for their feedback on this work; and Leif Nixon, and Aleksandr Konstantinov for their help with the NorduGrid integration. We would also like to thank Babak Sadighi, Tomas Olsson, Ludwig Seitz, and Erik Rissanen for their work on integrating Delegent into our authorization framework. Finally, we would like to thank Martin Folkman for his work on developing SGAS administration tools.

10. REFERENCES

[1] Apache Axis, http://ws.apache.org/axis, Apache Software Foundation, 2003.

[2] Apache Xindice, http://xml.apache.org/xindice, Apache Software Foundation, 2004.

[3] Contract Net Interaction Protocol Specification, FIPA, 2003.

[4] Globus Toolkit, http://www.globus.org/toolkit, Globus Alliance, 2004.

[5] SGAS, http://www.sgas.se, 2004.

[6] Sun's XACML Implementation, http://sunxacml.sourceforge.net/, Sun Microsystems, 2004.

[7] Xalan Java, http://xml.apache.org/xalan-j, Apache Software Foundation, 2004.

[8] Abramson, D., Giddy, J., and Kotler, L., High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? in *International Parallel and Distributed Processing Symposium (IPDPS)*, (Cancun, Mexico, 2000), 520-528.

[9] Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Frohner, Á., Gianoli, A., Lõrentey, K., and Spataro, F., VOMS, an Authorization System for Virtual Organizations. in *1st European Across Grids Conference*, (Santiago de Compostela, February 13-14, 2003).

[10] Anderson, A., Nadalin, A., Parducci, B., Engovatov, D., Lockhart, H., Kudo, M., Humenn, P., Godik, S., Abderson, S., Crocker, S., and Moses, T. eXtensible Access Control Markup Language (XACML) Version 1.0. Godik, S. and Moses, T. eds., OASIS, 2003.

[11] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S. Business Process Execution Language for Web Services Version 1.1. Thatte, S. ed., Microsoft, IBM, Siebel Systems, BEA, SAP, 2003.

[12] Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., and Xu, M. Web Services Agreement Specification (WS-Agreement), Draft, Global Grid Forum, 2004.

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

[14] Bartel, M., Boyer, J., Fox, B., LaMacchia, B., and Simon, E. XML-Signature Syntax and Procesing. Eastlake, D., Reagle, J. and Solo, D. eds., W3C, 2002.

[15] Booth, D., Haas, H., McCabe, F., Newcomber, E., Champion, M., Ferris, C., and Orchard, D. Web Services Architecture, W3C, 2003.

[16] Buyya, R., Abramson, D., and Giddy, J. A Case for Economy Grid Architecture for Service Oriented Grid Computing, Global Grid Forum, 2001.

[17] Chinnici, R., Gudgin, M., Moreau, J., Schlimmer, J., and Weerawarana, S. Web Service Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C, 2003.

[18] Della-Libera, G., Dixon, B., Garg, P., and Hada, S. Web Services Secure Conversation (WS-SecureConversation). Kaler, C. and Nadalin, A. eds., Microsoft, IBM, VeriSign, RSA Security, 2002.

[19] Elmroth, E., Gardfjäll, P., Mulmo, O., and Sandholm, T. 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.*

[20] Fielding, R.T. Architectural Styles and the Design of Network-based Software Architectures, Ph.D. Dissertation at the Information and Computer Science Department, University of California, Irvine, 2000.

[21] Foster, I., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Kishimoto, H., Maciel, F., Savva, A., Siebenlist, F., Subramaniam, R., Treadwell, J., and Reich, J.V. The Open Grid Services Architecture, Version 1.0, Global Grid Forum, 2004.

[22] Foster, I., and Kesselman, C. (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[23] Foster, I., Kesselman, C., Nick, J.M., and Tuecke, S. Grid Services for Distributed System Integration. *Computer*, *35* (6). 37-46.

[24] Guarise, A., Piro, R., and Werbrouck, A. DataGrid Accounting System - Architecture - v1.0, EU DataGrid, 2003.

[25] Hazelwood, V., Bean, R., and Yoshimoto, K., SNUPI: A Grid Accounting and Performance System Employing Portal Services and RDBMS Back-end. in *Linux Clusters: The HPC Revolution*, (Urbana/Champaign, USA, 2001).

[26] Imamura, T., Dillaway, B., and Simon, E. XML Encryption Syntax and Processing, W3C, 2002.

[27] Jackson, S. The Gold Accounting and Allocation Manager, http://sss.scl.ameslab.gov/gold.shtml, 2004.

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

[29] Jackson, S., and Lepro Metz, R. Usage Record -- XML Format, Global Grid Forum, 2003.

[30] Lorch, M., and Skow, D. Authorization Glossary, Global Grid Forum, 2004.

[31] Mitra, N. SOAP Version 1.2 Part 0: Primer, W3C, 2003.

[32] Newhouse, S. Grid Economic Services Architecture, Global Grid Forum, 2003.

[33] Pearlman, L., Welch, V., Foster, I., Kesselman, C., and Tuecke, S., A Community Authorization Service for Group Collaboration. in *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, (2002).

[34] Sadighi, B., Olsson, O., and Rissanen, E. Managing authorisations in dynamic coalitions, Swedish Institute of Computer Science, 2003.

[35] Smirnova, O., Eerola, P., Ekelöf, T., Ellert, M., Hansen, J.R., Konstantinov, A., Kónya, B., Nielsen, J.L., Ould-Saad, F., and Wäänänen, A. The NorduGrid Architecture and Middleware for Scientific Applications. *Lecture Notes in Computer Science*, *2657*. 264-273.

[36] Thigpen, W., Hacker, J., McGinnis, L., and Athey, B. Distributed Accounting on the Grid, Global Grid Forum, 2001.

[37] Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maquire, T., Sandholm, T., Snelling, D., and Vanderbilt, P. Open Grid Services Infrastructure (OGSI) Version 1.0, Global Grid Forum, 2003.

[38] Welch, V., Siebenlist, F., Chadwick, D., Meder, S., and Pearlman, L. Use of SAML for OGSA Authorization, Global Grid Forum, 2004.

[39] Zadeh, L.A. Fuzzy Sets. *Information and Control*, *8*. 338-353.