International Journal of Cooperative Information Systems
Vol. 15, No. 3 (2006) 439–459

World Scientific
www.worldscientific.com

A SERVICE-ORIENTED APPROACH TO ENFORCE GRID RESOURCE ALLOCATIONS

THOMAS SANDHOLM

Department of Numerical Analysis and Computer Science and PDC
Royal Institute of Technology, SE-100 44 Stockholm, Sweden

PETER GARDFJÄLL and ERIK ELMROTH

Department of Computing Science and HPC2N
Umeå University, SE-901 87 Umeå, Sweden

OLLE MULMO and LENNART JOHNSSON

Department of Numerical Analysis and Computer Science and PDC
Royal Institute of Technology, SE-100 44 Stockholm, Sweden

We present the SweGrid Accounting System (SGAS) — a decentralized and standards-based system for Grid resource allocation enforcement that has been developed with an emphasis on a uniform data model and easy integration into existing scheduling and workload management software.

The system has been tested at the six high-performance computing centers comprising the SweGrid computational resource, and addresses the need for soft, real-time quota enforcement across the SweGrid clusters.

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 that 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 to reserve allocations and charge for resource usage on behalf of the user.

We conclude this paper by illustrating the policy customization capabilities of SGAS in a simulated setting, where job streams are shaped using different modes of allocation policy enforcement. Finally, we discuss some of the early experiences from the production system.

Keywords: OGSA; grid; accounting; HPC.

1. Introduction

Advances in network technology, in addition to the more distributed and collaborative nature of todays research projects, have prompted high-performance computing (HPC) 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 become feasible to integrate the centers into Grids that enable flexible resource sharing and load balancing across organizational boundaries.[1]

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 HPC 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 compliance with the proposed Open Grid Services Architecture (OGSA),[2,3] (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 standardization efforts in the field of wide area distributed computing relevant to Grid computing. The SweGrid network and its accounting requirements are outlined in Sec. 3. In Sec. 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 Sec. 6 the implementation. Section 7 presents some results from simulations of reservations against our system, and discusses some feedback obtained from the production system. Finally, in Sec. 9, we summarize our contribution and our future research and development plans.

This paper is an extended and revised version of Ref. 4.

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) applied to Grid computing.[5] 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. In highly dynamic systems, there is often a trade-off between fine-grained control of state, and strict enforcement of decoupling. The core Web services specifications such as SOAP and WSDL do not fully address the problem of managing application state, but they provide extensibility features that may be leveraged by other specifications.[6,7] The often quoted REST 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.[8] REST is a very similar approach to the one taken by Web services workflow languages, such as BPEL4WS.[9] Request-broker-influenced specifications, however, address client agnostic fine-grained resource state sharing.[10]

Our work is based 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, and FIPA's Contract Net protocols are example technologies addressing that need.[11,12] Providing complete SLA lifecycle management support in a computational Grid is outside the scope of the work presented here, but we discuss how such a solution may be designed in Ref. 13.

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 computing 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 performed using the Globus Toolkit or the NorduGrid job submission tools, interfacing cluster-level schedulers at the local sites.[14,15] 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. Hence, SNAC is now allocating quotas to the SweGrid as a whole. However, this has a large impact on how accounting is done, because it is thereby not sufficient 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 uniform usage records that comply with each other.

Thus a real-time enforcement solution is required that allocates resources to projects in a fair manner while taking current user policies, resource policies, and allocation-authority policies into account. A 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. A user, on the other hand, may only want to execute a 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 systems according to local policies.

Even though SweGrid currently consists of a fairly homogeneous compute farm with similar middleware installations, it is expected that both the hardware and software solutions will 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 homogeneous run-time environment, and they lack standard and uniform ways to obtain and represent information from 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 Ref. 16.

In the European Data Grid Accounting System (DGAS), users need to pay for resources that they use in a virtual currency called Grid Credits.[17] Resources earn Grid Credits by offering 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 is provided as an extension to the Globus job manager, and it calculates job cost based on standard XML usage records.[18] 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 system utilizing resource price matrices.[19]

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) specified by the Global Grid Forum (GGF), presents an OGSA-based architecture using the concept of chargeable services.[20] 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 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.[21] SNUPI is, however, not service-oriented and assumes a homogeneous cluster environment.

QBank is a resource allocation management system developed for parallel computers.[22] Its successor, Gold, adds more advanced accounting features such as price quotes, funds transfers, and timestamped allocations.[23] 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), with the aim of meeting the accounting needs of SweGrid presented in Sec. 3, and with a particular focus on shared quota enforcement across organizational boundaries, simplicity of deployment, and security.[24] 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 rationale of the various system components.

We start by describing the flexible authorization framework (Sec. 5.1). In addition to a bank service (Sec. 5.2), which provides most of the accounting functionality,

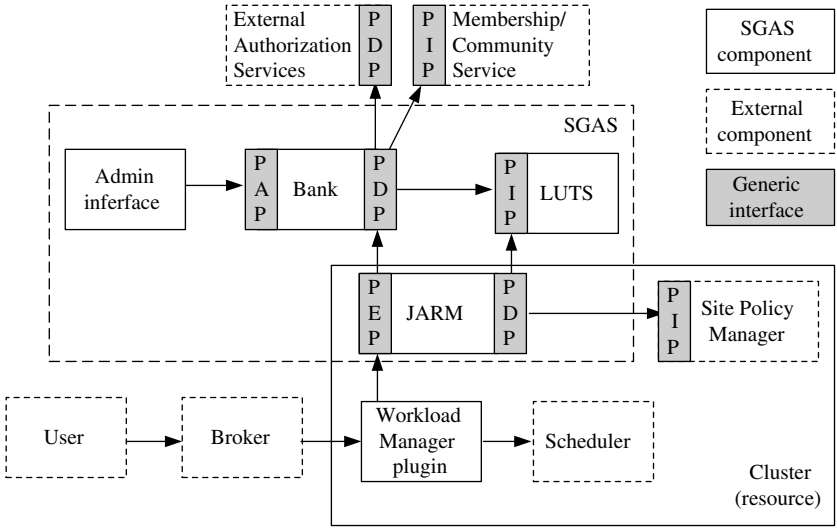

Fig. 1. SGAS Components overview.

there is a workload manager integration component (Sec. 5.4), and a usage tracking service (Sec. 5.3). 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 Secs. 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). The system has multiple decision points at various levels, allowing for both policy overlay (combining policies from multiple sources when making a decision) and 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, and OGSA Authorization.[25,26] We allow access permission policies to be specified in XACML (eXtensible Access Control Markup Language).[27] Figure 2 shows an 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

```
<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:string-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:subject-id"/>
    </SubjectMatch>
  </Subject>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          /O=Grid/O=NorduGrid/OU=cs.umu.se/CN=PeterGardfjell
    </AttributeValue>
    <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                                AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
    </SubjectMatch>
  </Subject>
  </Subjects>
  <Resources><AnyResource/></Resources>
  <Actions>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-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:action-id"/>
    </ActionMatch>
  </Action>
  </Actions>
 </Target>
 </Rule>
 <Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Fig. 2. Example of an XACML-encoded bank account policy granting withdrawals for two project members.

Delegent, 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.[28]

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 achieve this, external services such as VOMS and CAS may be used to gather membership evidence (Policy Information Point, PIP).[29,30]

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, and/or WS-SecureConversation.[30,31] In addition, when the resource contacts the bank, both the users delegated credentials associated with the requested job (made available by the workload manager plugin), as well as the resources 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 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 Web services, the Bank-, the Account-, and the Hold-services. The bank design is presented in some more detail in Ref. 33.

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 XML-defined policies. 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 (cf. making reservations on a credit card). The Hold is further only accessable 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 triggers an accounting transaction record and debits the Account that the Hold was held against. The amount reserved in a Hold does not have to match the committed amount, 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, 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[14] Resource Specification Language (RSL). They thereby initiate an implicit in-blanco signing process with the resources.

Figures 3 and 4 show the bank design, and a 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. Security interfaces are clearly separated from application interfaces, allowing the security implementation to be customized or replaced without affecting the core bank implementation.

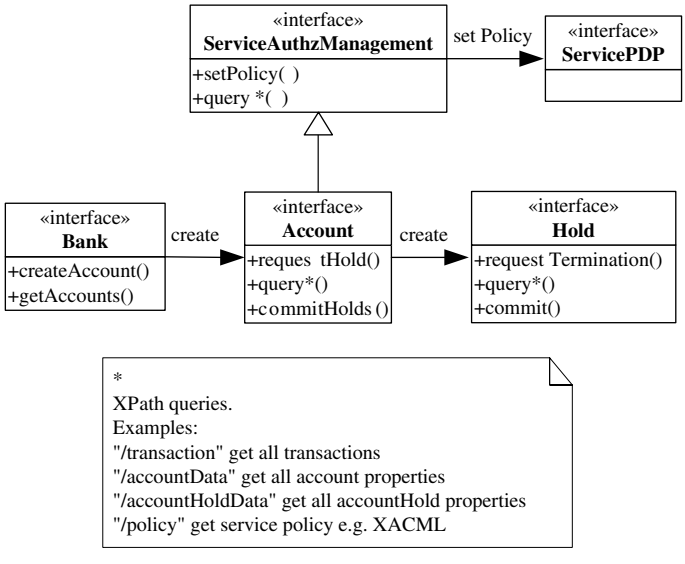

Fig. 3. Bank interfaces.

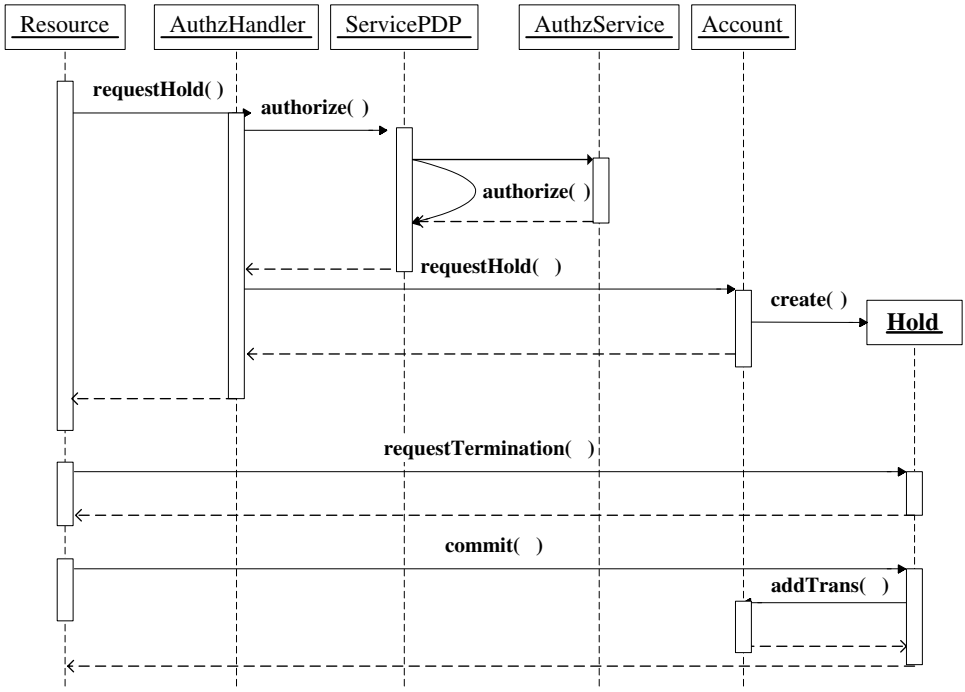

Fig. 4. Bank and resource interactions.

5.3. *LUTS*

The Logging and Usage Tracking Service (LUTS) is used to store usage records compliant with the GGF Usage Record (UR) XML format.[34] 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 and therefore very flexible and extensible.[35] 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 understand 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 users 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 (account reservation) 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 be easily customized for particular workload managers and site policies. Note that JARM shields the Site Policy Manager from knowledge about the bank system (see Fig. 1). A generic NorduGrid/ARC[15] 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 Ref. 36, and Ref. 17 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 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. For example, the initial implementation implemented all services compliant to the OGSI specification whereas the newest version uses WSRF.

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) provides a Java-based container implementation of the OASIS Web Services Resource Framework (WSRF) protocols, a realization of the OGSA model.[14,37] Both WSRF and GT are designed as a set of primitives that can be freely mixed, composed, extended, and embedded. WSRF facilitates cross-language interoperability, whereas the GT Java container provides a consistent, portable programming model. Below, we first summarize how the various WSRF concepts are leveraged in SGAS, and we then continue with describing how the GT container features are 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 WSRF (WS-ResourceLifetime). We control expiration and extension of Holds using the WS-ResourceLifetime 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 WSRF (WS-ResourceProperties). 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. WSRF recommends a factory pattern to create stateful resources in a uniform manner, which we apply. The factory pattern is used to create both Account, and Hold resources.

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, XML-Encryption, and XML-Signature, respectively.[32,38,31]

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 (eXist[39]) and XPath (Xalan[35]) 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 resource property 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 Ref. 33.

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 can be balanced across many Bank and LUTS services. Additionally, charging and logging are 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 is used in the bank as a standard, self-contained PDP engine, which checkpoints policies to the eXist database.[40] Some schedulers already have support for the GGF UR format, but for others, we provide an XSLT style sheet transformer based framework to simplify SGAS integration at local sites.[35]

7. Illustration of Policy Customization Capabilities

In order to illustrate the policy customization capabilities of SGAS and their effects, we built a simulation framework aimed at measuring job turnaround times. The usage pattern that was simulated consisted of an allocation authority that

Table 1. Toolkits and standards.

Toolkit	SGAS Feature	Standards Implemented
Globus	Service state management	WSRF
Globus	Mutual authentication, credential delegation	GSI profile of WS-Secure-Conversation
Globus	Payload integrity, and privacy	XML-Signature, XML-Encryption
eXist	XML database (for policy and service state)	XML:DB
Xalan	Query engine, stylesheet transformation	XPath, XSLT
SunXACML	XACML PDP	XACML
Axis	Web services engine	SOAP, WSDL
SGAS	SGAS Usage Records	GGF XML Usage Record

periodically adds new resource allocations to a bank account, and account members that continuously consume their allocations by submitting jobs (thereby creating and committing account reservations).

The behavior was studied for different policy configurations and two separate job flows. First, a fair flow is a stream of job submissions 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 stream of job submissions 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. In case fairness is the sole objective, allocation enforcement can be carried out strictly which would disallow any quota-exceeding jobs in the unfair flow. In the simulations, we employed soft allocation enforcement, which allows jobs to run even though the user account is overdrawn, thus representing a trade-off where resource utilization is increased at the expense of fairness. If the reservation fails due to an overdraft violation, then there is a penalty in job execution time, simulating the job being put in a low priority queue by the scheduler. The degree of enforcement "softness" may be controlled through an overdraft limit policy, which we base on access control policy (XACML) rule conditions that may be set by account administrators.

It should be noted that the XACML policies used are mere examples of viable algorithms that may be easily applied using the SGAS customizability, and extensibility features. That is, the aim here is not to show an optimal algorithm, but rather to illustrate how a certain policy (overdraft protection in this case) can be implemented in SGAS. However, the simulated policy of tracking overdrafts and causing jobs exceeding their quota to run slower using the algorithm described below is indeed employed in the SweGrid production system today, and the results are hence relevant to the current use of SGAS.

Table 2 lists the configuration used in the simulation runs. Fair and unfair submit intervals denote the duration between successive job submissions for a well- and ill-behaved user, respectively. For each job, an account reservation for 60 credits is placed (corresponding to the 60 second job duration). Allocation interval refers to the time between two successive allocations, which in the SNAC case typically is one month (see Sec. 3 and 5.1). Allocation amount is the size of each such allocation.

Table 2. Simulation setup.

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

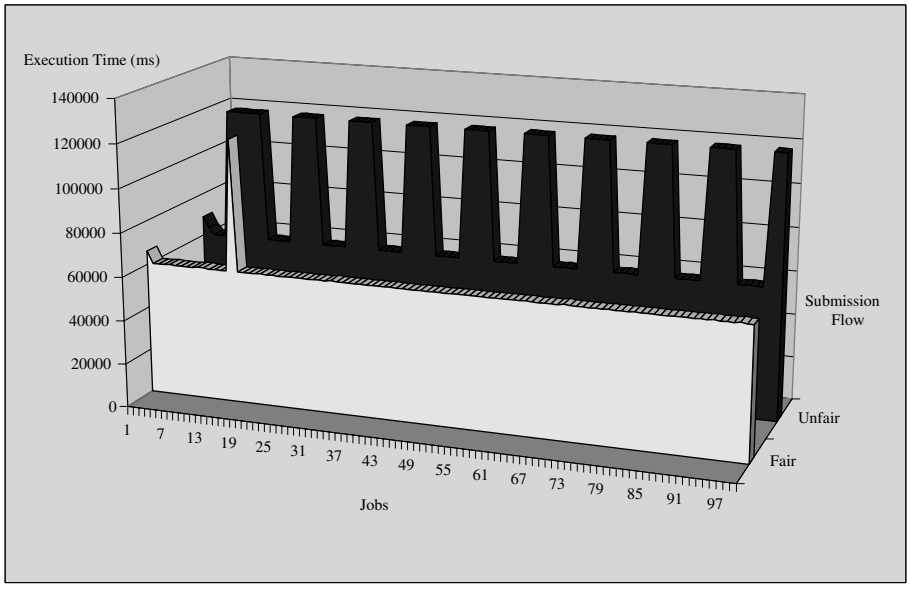

Fig. 5. Submission flow simulation using fair and unfair flows (zero overdraft limit).

Overdraft penalty is the extra execution time added due to an overdraft (that is not allowed by policy). This simulates a scenario where jobs within the allocation limit are started immediately, whereas quota-exceeding jobs are penalized with an extra 60 seconds of queue time.

All the simulations can be reproduced, and the source code can be obtained by downloading the SGAS Open Source distribution.[24]

Figure 5 compares the job turnaround times of a fair and an unfair flow disallowing all overdrafts. The area below the flow curves represents the accumulated execution time of all jobs in the flow. Thus, the larger the area is, the worse is the turnaround, and the higher is the aggregated penalty time. The peaks of the flows resulted from overdraft violations. Thus, the thinner the peaks are, the closer is the user to the actual allocation. Note that the fair flow was shaped to get minimum job turnaround time 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

```
<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>
```

Fig. 6. Example of an overdraft policy allowing 75% overdraft.

happened too late to avoid an overdraft. Over time, however, the allocations and reservations were synchronized. The periodicity of the peaks in the unfair flow corresponds to the available quota running low shortly before the new allocations are granted.

The authorization framework used by SGAS (Sec. 5.1) allows account owners to set policies regulating access to their accounts with XACML policies. An example policy condition is given in Fig. 6. The actual value of the XACML attribute sgas:overdraw:percent:requested is calculated as:

$$\frac{a_s + a_r + r_r}{a_t},$$

where a_s is the allocation spent, a_r is the allocation reserved, r_r is the requested reservation, and a_t 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 Fig. 2.

In our simulations, we tested three different policies allowing 25, 50, and 75% overdraft against the unfair job reservation flow. The results can be seen in Fig. 7. We note that turnaround in the 25% policy case is close to the unregulated unfair flow, whereas, the more permissive, 75% policy is getting closer to the turnaround of the fair flow.

8. Early Experiences

SGAS has been tested in the SweGrid production system since September 2004. After the original submission of this paper, SGAS was ported from OGSI to WSRF and Globus 4. One of the initial experiences gained from the production system was that some large-scale jobs requesting 100s of CPUs could cause the bank communication to become a bottleneck for the sites. We therefore allowed the sites to use transport-level security (SOAP over SSL) as an alternative to WS-Secure Conversation. The WSRF port made the Web container consume less memory (as a result of a cleaner separation between a Web service and the stateful resource being managed) and as a result solved some scalability problems we were experiencing in

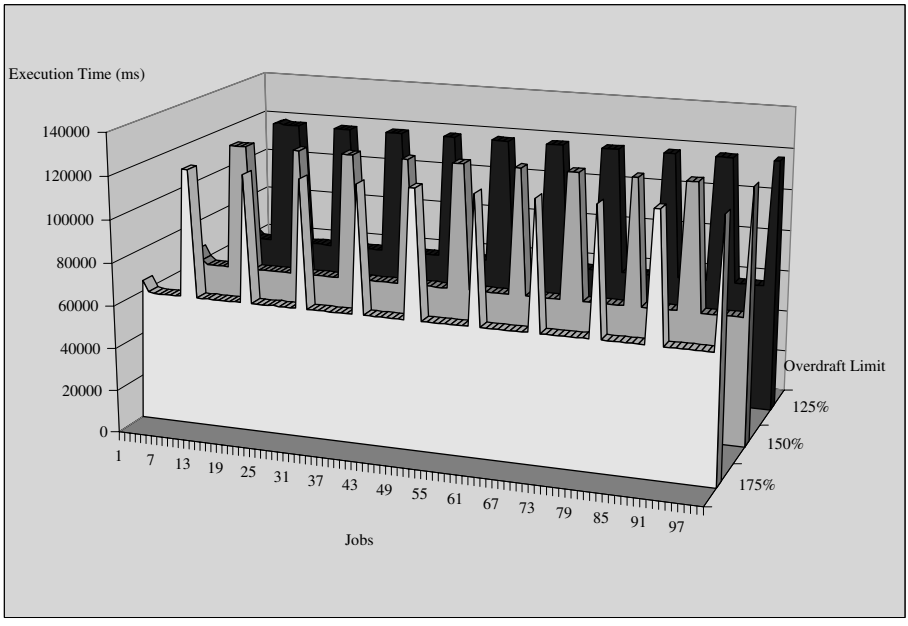

Fig. 7. An unfair submission flow simulation using policy-based overdraft protection (25%, 50%, and 75% overdraft limits).

the persistency layer. We also implemented automatic phase outs of old accounts and LUTS records to cope with the production level load of SweGrid.

From a policy management point of view, it is still a too manual of a task to distribute allocations to projects and too complicated for PIs to redistribute their quotas to project members. Further, the service-level differentiation offered at the resource sites is often too coarse grained (regular operation, or free pool low priority operation). The price setting is not dynamic enough and does not give users an incentive to use less powerful resource at less loaded times. SGAS has improved the fairness of resource sharing in SweGrid but more could be done to allow users who need to submit jobs urgently to pay more and get a higher priority.

To address some of these issues, we have studied related economic theory applicable to computational markets. Tycoon, a proportional share market-based resource allocation system, provides a low-level infrastructure that can ensure both a high degree of economic efficiency in terms of social welfare, or aggregated total user utility as defined in Ref. 41, and a high degree of fairness in terms of envy-freeness, as defined in Ref. 42.[43] Current research focuses on integrating the Tycoon system with Grid environments typified by SweGrid.

As an alternative to a market-driven solution, we have also investigated a complementary resource allocation mechanism in Ref. 44, where we demonstrated the potential of a decentralized architecture for a Grid-wide fairshare scheduling system in SweGrid-like environments and usage scenarios. Here, job priorities are individually assigned to each job based on the deviation between allocated and

consumed share of resources. The system, which preserves local site autonomy, enforces both locally and globally scoped share policies, allowing local resource capacity as well as aggregate VO capacity to be logically divided across different groups of users. The policy model is hierarchical and subpolicy definition can be delegated so that, e.g. a VO can distribute shares of its aggregate resource capacity across its projects, which in turn can divide their shares between project members. Notably, there is no need for a central coordinator as policies are enforced collectively by the resource schedulers. Each local scheduler adopts a Grid-wide view on utilization in order to steer local resource utilization to not only maintain local resource shares but also to contribute to maintaining global shares across the entire set of Grid resources.

We acknowledge that SGAS needs to be tested and validated in more diverse applications and Grid networks, and we hope that the recent inclusion of SGAS in the Globus toolkit distribution and an ongoing integration with the CERN LCG Grid in the EGEE project will give us data for future analysis of this kind.

9. 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-centric 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 three-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 regulate 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.

Besides improving baseline functionality, scalability and robustness, we mainly intend to continue to improve this system 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.

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.

This work has been supported by The Swedish Research Council (VR) under contracts 343-2003-953, 343-2003-954, and 621-2005-3667.

References

1. I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure* (Morgan Kaufmann, 1999).
2. I. Foster, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, H. Kishimoto, F. Maciel, A. Savva, F. Siebenlist, R. Subramaniam, J. Treadwell and J. V. Reich, The Open Grid Services Architecture, Version 1.0, Global Grid Forum (2004).
3. I. Foster, C. Kesselman, J. M. Nick and S. Tuecke, Grid services for distributed system integration, *Computer* **35**(6) (2002) 37–46.
4. T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson and O. Mulmo, An OGSA-based accounting system for allocation enforcement across HPC centers, in *ICSOC'04*, (ACM, 2004), pp. 279–288.
5. D. Booth, H. Haas, F. McCabe, E. Newcomber, M. Champion, C. Ferris and D. Orchard, Web Services Architecture (W3C, 2003).
6. N. Mitra, SOAP Version 1.2 Part 0: Primer — Section 1.1 (W3C, 2003).
7. R. Chinnici, M. Gudgin, J. Moreau, J. Schlimmer and S. Weerawarana, Web Service Description Language (WSDL) Version 2.0 Part 1: Core Language — Section 2.8 (W3C, 2003).

8. R. T. Fielding, 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).
9. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic and S. Weerawarana, Business Process Execution Language for Web Services Version 1.1., ed. S. Thatte (Microsoft, IBM, Siebel Systems, BEA, SAP, 2003).
10. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling and P. Vanderbilt, Open Grid Services Infrastructure (OGSI) Version 1.0, Global Grid Forum (2003).
11. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke and M. Xu, Web Services Agreement Specification (WS-Agreement), Draft, Global Grid Forum (2004).
12. Contract Net Interaction Protocol Specification, FIPA (2003).
13. T. Sandholm, Service level agreement requirements of an accounting-driven computational grid. Technical Report TRITA-NA-0533 (Royal Institute of Technology, Stockholm, September, 2005).
14. I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, in *IFIP International Conference on Network and Parallel Computing*, 3799 (Springer-Verlag LNCS, 2005), pp. 2–13.
15. 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 in Lecture Notes in Computer Science, Vol. 2657 (Springer-Verlag, 2003), pp. 264–273.
16. W. Thigpen, J. Hacker, L. McGinnis and B. Athey, Distributed accounting on the grid (Global Grid Forum, 2001).
17. A. Guarise, R. Piro and A. Werbrouck, DataGrid accounting system — Architecture-v1.0 (EU DataGrid, 2003).
18. A. Barmouta and R. Buyya, GridBank: A grid accounting services architecture (GASA) for distributed systems sharing and integration, in *Int. Parallel and Distributed Processing Symposium (IPDPS'03)*, IEEE, Nice, France (2003).
19. D. Abramson, J. Giddy and L. Kotler, High performance parametric modeling with Nimrod/G: killer application for the global grid?, in *Proc. Int. Parallel and Distributed Processing Symposium (IPDPS)* (Cancun, Mexico, 2000), pp. 520–528.
20. S. Newhouse, Grid Economic Services Architecture, Global Grid Forum (2003).
21. V. Hazelwood, R. Bean and K. Yoshimoto, SNUPI: A grid accounting and performance system employing portal services and RDBMS back-end, in *Linux Clusters: The HPC Revolution*, Urbana/Champaign, USA (2001).
22. S. Jackson, QBank: A resource management package for parallel computers, Pacific Northwest National Laboratory, Washington, USA (2000).
23. S. Jackson, The gold accounting and allocation manager (2004), http://sss.scl.ameslab.gov/gold.shtml.
24. SGAS (2005), http://www.sgas.se.
25. M. Lorch and D. Skow, Authorization glossary, Global Grid Forum (2004).
26. V. Welch, F. Siebenlist, D. Chadwick, S. Meder and L. Pearlman, Use of SAML for OGSA authorization, Global Grid Forum (2004).
27. A. Anderson, A. Nadalin, B. Parducci, D. Engovatov, H. Lockhart, M. Kudo, P. Humenn, S. Godik, S. Abderson, S. Crocker and T. Moses, eXtensible access control markup language (XACML) Version 1.0., eds. S. Godik and T. Moses, OASIS (2003).

28. L. Seitz, E. Rissanen, T. Sandholm, B. Sadighi Firozabadi and O. Mulmo, Policy administration control and delegation using XACML and delegent, in *Proc. 6th IEEE/ACM Int. Workshop on Grid Computing*, Seattle (November, 2005).
29. R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey and F. Spataro, VOMS, an authorization system for virtual organizations in *1st European Across Grids Conference*, Santiago de Compostela, 13–14 February (2003).
30. L. Pearlman, V. Welch, I. Foster, C. Kesselman and S. Tuecke, A community authorization service for group collaboration, in *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks* (2002).
31. M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon, *XML-Signature Syntax and Processing*, eds. D. Eastlake, J. Reagle and D. Solo (W3C, 2002).
32. G. Della-Libera, B. Dixon, P. Garg and S. Hada, *Web services secure conversation* (*WS-SecureConversation*), eds. C. Kaler and A. Nadalin (Microsoft, IBM, VeriSign, RSA Security, 2002).
33. E. Elmroth, P. Gardfjäll, O. Mulmo and T. Sandholm, An OGSA-based bank service for grid accounting systems, in *Applied Parallel Computing. State-of-the-art in Scientific Computing*, Lecture Notes in Computer Science, Vol. 3732 (Springer-Verlag, 2005), pp. 1051–1060.
34. S. Jackson and R. Lepro Metz, Usage record — XML format, Global Grid Forum (2003).
35. Xalan Java (Apache Software Foundation, 2004), http://xml.apache.org/xalan-j.
36. R. Buyya, D. Abramson and J. Giddy, A case for economy grid architecture for service oriented grid computing (Global Grid Forum, 2001).
37. S. Graham, A. Karmarkar, J. Mischkinksy, I. Robinson and I. Sedukhin, Web services resource 1.2 (OASIS, 2005).
38. T. Imamura, B. Dillaway and E. Simon, XML encryption syntax and processing (W3C, 2002).
39. eXist (2005), http://exist.sourceforge.net.
40. Sun's XACML Implementation (Sun Microsystems, 2004), http://sunxacml.sourceforge.net.
41. C. H. Papadimitriou, Algorithms, games and the internet, in *Symposium on Theory of Computing* (2001).
42. H. R. Varian, Equity, envy, and efficiency, *Journal of Economic Theory* **9** (1974) 63–91.
43. M. Feldman, K. Lai and L. Zhang, A price-anticipating resource allocation mechanism for distributed shared clusters, in *ACM Conference on Electronic Commerce* (June, 2005).
44. E. Elmroth and P. Gardfjäll, Design and evaluation of a decentralized system for grid-wide fairshare scheduling, in *e-Science 2005 First IEEE Conference on e-Science and Grid Computing* (IEEE Computer Society Press, USA, 2005), pp. 221–229.