An OGSA-Based Bank Service for Grid Accounting Systems*

Erik Elmroth[1], Peter Gardfjäll[1], Olle Mulmo[2], and Thomas Sandholm[2]

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

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

Abstract. This contribution presents the design and implementation of a bank service, constituting a key component in a recently developed Grid accounting system. The Grid accounting system maintains a Grid-wide view of the resources consumed by members of a virtual organization (VO). The bank is designed as an online service, managing the accounts of VO projects. Each service request is transparently intercepted by the accounting system, which acquires a reservation on a portion of the project's bank account prior to servicing the request. Upon service completion, the account is charged for the consumed resources. We present the overall bank design and technical details of its major components, as well as some illustrative examples of relevant service interactions. The system, which has been implemented using the Globus Toolkit, is based on state-of-the-art Web and Grid services technology and complies with the Open Grid Services Architecture (OGSA).

Keywords: Grid accounting, allocation enforcement, OGSA, SweGrid.

1 Introduction

This contribution presents the design and implementation of a bank service, constituting a key component in the SweGrid Accounting System (SGAS) [10] – a recently developed Grid accounting system, initially targeted for use in SweGrid [15]. A Grid accounting system maintains a Grid-wide view of the resources consumed by members of a *virtual organization* (VO) [6]. The information gathered by the system can serve several useful purposes, such as to allow enforcement of project quotas.

The bank is designed as an online service, handling accounts that contain the resource allocations of Grid projects. Each service request (job submission) is transparently intercepted by the accounting system, which acquires a reservation on a portion of the project account prior to job execution (cf. credit card reservations). Upon job completion, the project account is charged for the consumed resources and the reservation is released. The decentralized nature of Grids, assuming the absence of any central point of control, adds complexity to the problem. The problem is further complicated by the distributed

* This work was funded by The Swedish Research Council (VR) under contracts 343-2003-953 and 343-2003-954 and The Faculty of Science and Engineering, Umeå Univ.

resource administration in Grids, i.e., the requirement that resource owners at all times retain local control over their resources.

The rest of this paper is outlined as follows. Section 2 gives a brief introduction to the accounting issues addressed, the SweGrid environment, the concept of Grid services, and an overview of the accounting system in which the bank is a key component. The bank design is described in Section 3, including a presentation of the services constituting the bank and their relationships. Some implementation details, including a fine-grained and customizable authorization framework, are summarized in Section 4. Finally, Section 5 concludes the paper.

2 Background

The accounting system is primarily targeted towards allocation enforcement in SweGrid [15], although it has been designed to allow simple integration into different Grid environments. SweGrid is a Swedish national Grid, initially including 6 geographically distributed Linux clusters with a total of 600 CPUs dedicated for Grid usage 24 hours a day.

The computer time of SweGrid is controlled by the Swedish National Allocations Committee (SNAC) [12], which issues computer time, measured in *node hours* per month, to research projects. Node hours are assigned to projects based on their scientific merits and resource requirements, after a peer-reviewed application process. The accounting system must provide coordinated enforcement of these quotas across all sites. That is, the whole allocation may be consumed at one cluster or in parts at any number of clusters.

The design considerations for the bank service is closely related to those of the accounting system as a whole. The design is based on the assumption that a user can be a member of several VOs and participate in one or more projects within each VO. Each project's resource allocation is kept in an account, which is handled by the bank.

The information maintained by the accounting system can, e.g., form the basis for direct economic compensation, usage quota enforcement, tracking of user jobs, resource usage evaluation, and dynamic priority assignment of user requests based on previous resource usage.

The system performs soft real-time allocation enforcement. The enforcement is real-time in that resources can deny access at the time of job submission, e.g., if the allocation has been used up, and soft in that the level of enforcement strictness is subject to local policies.

2.1 Accounting System Overview

Figure 1 presents the main entities and interactions of SGAS. Entity interactions are illustrated in a scenario where a job is submitted to a computer cluster, although it could be generalized to cover a generic service request to an arbitrary Grid resource.

Each VO has an associated *Bank* service to manage the resource allocations of the VO research projects. The Bank is primarily responsible for maintaining a consistent view of the resources consumed by each research project, and enables coordinated

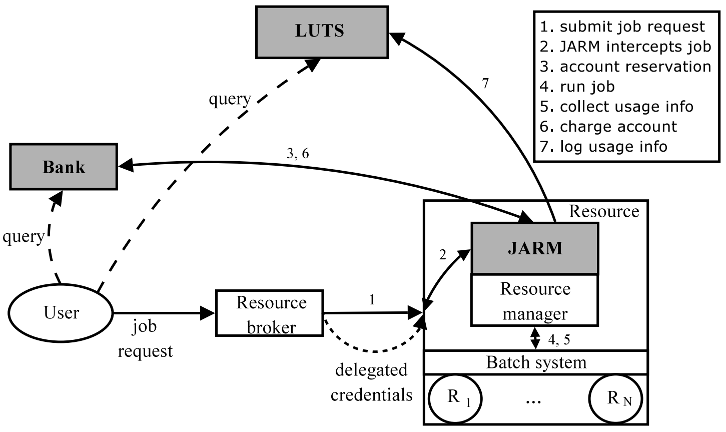

Fig. 1. Interactions among accounting system entities (shaded) during job submission

quota enforcement across the Grid sites. From a scalability perspective, a single bank per VO might seem restrictive. However, it should be stressed that the Bank service is not confined to a single site. It could be implemented as a virtual resource, composed of several distributed services, to achieve load-balancing and scalability. The *Job Account Reservation Manager* (JARM) is the (single) point of integration between SGAS and the underlying Grid environment. On each Grid resource, a JARM intercepts incoming service requests, performs account reservations prior to resource usage and charges the requester's account after resource usage. A *Log and Usage Tracking Service* (LUTS) collects and publishes Usage Records [8], holding detailed information about the resources consumed by particular service interactions.

In a typical scenario, the user, or an entity acting on behalf of the user, such as a broker, sends a job request to a resource that has been selected to execute the job. During the job request process, mutual authentication is performed and the user's credentials are delegated to the resource (1). The job request, which includes the identity of the project account, is intercepted by the resource's JARM (2), which contacts the VO Bank to acquire a time-limited reservation on a portion of the project allocation (3). Such an account reservation is referred to as a *hold*. If a hold can be granted, the JARM forwards the job request to a local resource manager (4) that runs the job and gathers information about the resources consumed by the job. At job completion, the JARM collects the usage information (5), charges the project account utilizing the hold (6), and records the usage information in a Usage Record which is logged in a LUTS (7). Any residual amount of the hold is released. Notably, a user can query both the Bank and the LUTS, e.g., for various account and job information.

2.2 The Grid Service Concept

The implementation of the accounting system, including its bank component, is based on the concept of *Grid services* as defined by the Open Grid Services Architecture (OGSA) [5]. OGSA extends the concept of *Web services* [17] by introducing a class of transient (bounded lifetime) and stateful (maintain state between invocations) Web services. Web services represent an XML-based distributed computing technology that is independent of platform, operating system and programming language. Web services are ideal for loosely coupled systems where interoperability is a primary concern.

The transient nature of Grid services makes them suitable for representing not only physical resources but also more lightweight entities/activities, such as a video conference session, a data transfer, or in this case, different parts of a Grid accounting system.

All services expose *service data*, a dynamic set of XML-encapsulated information about service metadata and local service state. Grid services provide operations for querying (service introspection) and updating service data.

The Open Grid Services Infrastructure (OGSI) [16] defines a core set of composable interfaces which are used for constructing Grid services. The bank component presented in Section 3 makes use of the OGSI interfaces for, e.g., service creation, lifetime management, and service introspection. The interfaces are defined in Web Services Description Language (WSDL) [18] portTypes and specify the operations as well as the service data exposed by each service.

3 Bank Design

The Bank component of SGAS has been designed with generality and flexibility in mind. Specifically, the bank does not assume any particular type of resources, and as such it can be integrated into any Grid environment. For example, all bank transactions are performed using *Grid credits* – an abstract, unit-less currency that can be used to charge for arbitrary resource usage. Prior to charging an account, a resource may apply any transformation function to map different kinds of resource usage into Grid credits.

Since SweGrid resources are currently homogeneous and node-hours is the only resource type being accounted for, the resource-to-Grid credits mapping is trivial. In the general case of a heterogeneous Grid environment, different types of resource usage as well as differing resource characteristics need to be considered. For example, storage utilization could be accounted for as well, and faster processors might be more expensive. In case dedicated allocations need to be maintained for different types of resources, separate accounts may be provided for each resource type.

The bank is also neutral with respect to policies. Policy decisions are left to users, resource managers and allocation authorities. The bank provides the flexibility to accommodate such policies, as well as means of enforcing them. For example, policies dictated by the allocation authority or the resource might allow a job to run even though the project allocation has been used up. However, if project quotas are not strictly enforced by a resource, the user can still decide only to perform safe withdrawals that do not exceed the project allocation.

The SGAS bank is composed of three tightly integrated Grid services (`Bank`, `Account`, `Hold`), whose relationships are illustrated in Figure 2.

Bank Service. The `Bank` service is responsible for creating as well as locating `Accounts`. The `Bank` service implements the factory pattern as provided by OGSI.

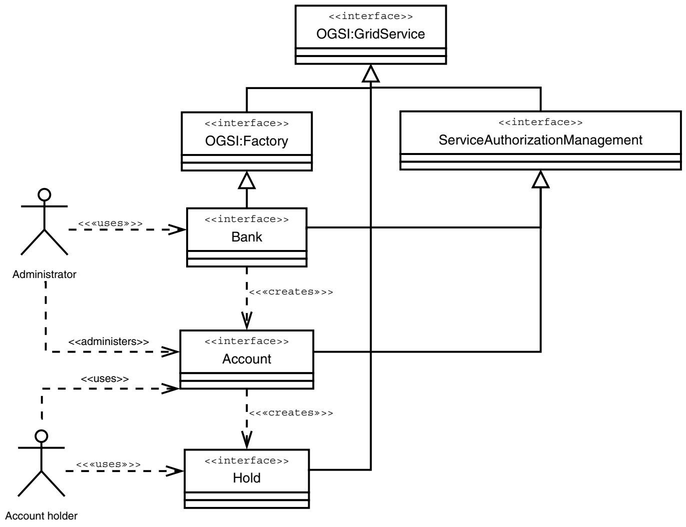

Fig. 2. Bank interface relationships

This allows the `Bank` to create new `Account` service instances. Clients can also query the `Bank` to obtain a list of the `Accounts` they are authorized to use.

Account Service. An `Account` service manages the resource allocation of a research project. A project member can request a hold on the `Account`, effectively reserving a portion of the project allocation for a bounded period of time. A successful hold request results in the creation of a `Hold` service, acting as a lock on the reserved account quota. We refer to the `Account` service that created a `Hold` service as the *parent account* of that hold. `Account` services publish transaction history and account state, which can be queried by authorized `Account` members.

The set of authorized `Bank` and `Account` users, as well as their individual access permissions, is dynamic and can be modified at run-time by setting an authorization policy, defined using XML. To this end, the `Bank` and `Account` interfaces extend the ServiceAuthorizationManagement (SAM) [10] service interface. SAM allows authorization policies to be associated with a service. The authorization policy could, e.g., contain an access control list associating a set of authorized `Account` members with their individual privileges. SAM is customizable, in that it allows different back-end authorization engines, also referred to as Policy Decision Points (PDP), to be configured with the service. Note that there is nothing `Bank`-specific about SAM; it can be used by any service requiring fine-grained management of authorization policies.

Hold Service. A `Hold` service represents a time-limited reservation on a portion of the parent account's allocation. `Holds` are usually acquired prior to job submission and committed after job completion to charge their parent account for the resources consumed by the job. `Hold` services are created with an initial lifetime and can be destroyed either explicitly or through expired lifetime. Hold expiry is controlled using the lifetime management facilities provided by OGSI. On commit or destruction, the Hold service is destroyed and any residual amount is returned to the parent account. In the case of destruction, the entire reservation is released. On commit, a specified portion of the Hold, corresponding to the actual resource usage, is withdrawn from the parent account, and a transaction entry is recorded in the transaction log.

Service Interfaces. The operations exposed by the bank services are presented in Table 1. Note that the service interfaces extend OGSI interfaces, which are used for such purposes as lifetime management, service creation and service introspection. The Bank and Account services provide batch commit operations (commitHolds), which allow resources to perform asynchronous commits of sets of Hold services in a single service invocation. The benefit of this approach is twofold, the perceived resource response time decreases, allowing higher job throughput during periods of high load, and the bank request load is reduced, improving overall system scalability. Table 2 gives an overview of the service data exposed by the bank services. Note that the service data set of each service also contains the service data of extended OGSI interfaces. Furthermore, note that since Bank and Account extend the SAM interface, these services also expose the servicePolicy service data element. Only authorized service clients are allowed to query the service data.

Table 1. The operations exposed by the SGAS bank interfaces

portType	Operation	Description
SAM	setPolicy	Set an authorization policy to be associated with service.
Bank	getAccounts	Get all accounts that caller is authorized to use.
	commitHolds	Batch commit of several Holds (created by any account in Bank).
Account	requestHold	Creates an account Hold if enough funds are available.
	addAllocation	Add a (potentially negative) amount to the account allocation.
	commitHolds	Batch commit of several Holds (created by this account).
Hold	commit	Withdraws a specified amount of the hold from the parent account.

Service Interactions. The Bank allows privileged users to create new Account services. During the lifetime of an Account, its set of members and their access rights can be modified by associating a new authorization policy with the Account, through the setPolicy operation. The project's resource allocation can be updated by an administrator by issuing a call to the addAllocation operation of the Account.

Table 2. The service data exposed by the SGAS bank interfaces

portType	serviceData	Description
SAM	servicePolicy	The authorization policy associated with the service.
Bank	none	-
Account	accountData	The account state: total allocation, reserved funds and spent funds.
	transactionLog	Publishes all transaction log entries.
Hold	holdData	The reserved amount and the identity of the parent account.

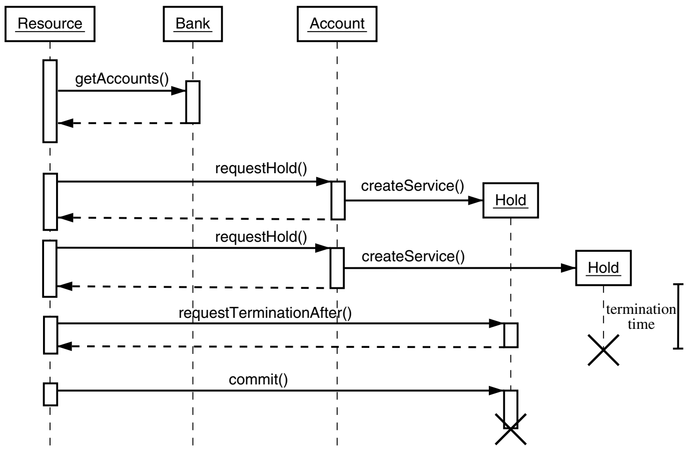

Fig. 3. Hold creation and usage

Figure 3 illustrates typical interactions between a resource and an Account service. A job is submitted by a user to a resource, which invokes the getAccounts operation on the Bank to get the user Account (unless it is specified in the job request). The resource requests a Hold on the Account, prior to job execution. The request includes an initial Hold expiry time. The expiry time can be reset at any time using the lifetime management operations provided by OGSI. A resource may choose to extend the Hold lifetime, e.g., due to a long batch queue. The commit operation is invoked by the resource after job completion to charge the Hold's parent account for the consumed resources. If the Hold is destroyed, either explicitly or through expired lifetime, the Hold amount is returned to the parent Account.

4 Bank Implementation

Interoperability is a primary concern in Grid computing. Thus, rather than implementing our own middleware with ad-hoc protocols and communication primitives, we leverage the latest Grid standardization efforts and toolkits. Current Grid standardization focuses on Web service technologies in general and OGSA in particular.

Globus Toolkit. Our implementation is based on the Globus Toolkit (GT) [11] open-source Java reference implementation of the OGSI specification, implemented on top of the Axis SOAP engine [19]. GT provides a container framework to host Grid services and a set of tools and implementations of core OGSI interfaces, allowing developers to build and deploy custom Grid services.

By basing our solution on GT we conform to the latest standard specifications, thereby achieving desirable interoperability. Furthermore, composing our solution of toolkit primitives cuts down development time.

GT also provides a security framework orthogonal to application code, which includes mutual authentication, message encryption and message signing. These security primitives are based on the WS-SecureConversation [9], XML-Signature [3] and XML-Encryption [7] standards.

Service Implementations. All bank services are implemented using operation providers, which offer a delegation-based implementation approach where services are defined in terms of operation providers, each implementing part of the service functionality. Operation providers facilitate implementation reuse as well as a development model based on composition of primitives. For example, a service can be given the capabilities of the SAM interface simply by configuring the SAM operation provider with the service.

In order to guarantee recoverability, all services are checkpointed to a back-end database that runs embedded in the GT container. Our implementation uses Xindice [2], an open-source native XML database that conforms to the standards developed by the XML:DB group [21].

In the event of a server crash the state of all services needs to be recovered from secondary storage on server restart. The recovery is performed by means of GT service loaders. Besides enabling state checkpointing and recovery, the database solution further allows users to pose non-trivial queries against bank information, such as the transaction log, by exposing database content as service data, and embedding database query expressions in service data queries. The GT query evaluator framework allows us to redirect those service data queries to the back-end database, effectively exposing a database view through the service data framework. For example, an `Account` member can run XPath [20] queries against the transaction log to obtain specific transaction information. The XPath query approach further avoids the inconvenience of defining a query language by means of different interface operations.

Authorization Framework. The GT framework provides a declarative security infrastructure through the use of *security deployment descriptors*. Authentication method as well as handling of delegated credentials can be specified on a per-operation basis. The available framework for authorization, on the other hand, only allows authorization to be specified on a per-service basis.

As the current all-or-nothing access to a service is too coarse for our purposes, we have developed a fine-grained authorization framework [10]. Through the SAM interface, an authorization policy and an authorization engine (a PDP) can be associated with a service. The SAM design is highly flexible and customizable since it allows different authorization back-ends, using different policy languages, to be configured with a service. The current implementation uses a default authorization back-end based on the eXtensible Access Control Markup Language (XACML) [1]. Specifically, Sun's XACML PDP implementation [14] is used as the underlying authorization engine. However, successful experiments have been carried out using other authorization engines as well. Using XACML we have also included an overdraft policy, allowing an administrator to set an upper limit on acceptable account overdraft.

GT uses different handlers that intercept a SOAP [13] request before reaching the target service. To incorporate our authorization framework we provide an authorization handler that delegates the authorization decision to the target service's PDP, if one is available.

The authorization framework is orthogonal to the service implementation. That is, the service implementation is not affected by customization or replacement of the security implementation.

5 Concluding Remarks

We have presented the design and implementation of a bank service for use in a recently developed Grid accounting system. The bank, as well as the accounting system as a whole, is designed to be general and customizable, allowing non-intrusive integration into different Grid environments. The system, which is standards-based and leverages state-of-the-art Web and Grid services technology, offers user-transparent enforcement of project allocations while providing fine-grained end-to-end security.

Our planned future work includes a transition towards the Web Services Resource Framework (WSRF) [4]. Other areas that are subject to further investigation include more flexible handling of project allocations and measures of improving scalability.

Acknowledgments

We acknowledge Lennart Johnsson, Royal Institute of Technology and Bo Kågström, Umeå University, for fruitful discussions and constructive comments. We would also like to thank Martin Folkman, Uppsala University, for his work on developing an SGAS administration tool. We are also grateful for the constructive comments in the feedback from the refereeing process.

References

1. A. Anderson, A. Nadalin, B. Parducci, D. Engovatov, H. Lockhart, M. Kudo, P. Humenn, S. Godik, S. Anderson, S. Crocker, and T. Moses. eXtensible Access Control Markup Language (XACML) Version 1.0, OASIS, 2003.
2. Apache Xindice, 2004. http://xml.apache.org/xindice/.
3. M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-Signature Syntax and Processing, W3C, 2002.

4. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework: Version 1.0, 2004. http://www.globus.org/wsrf/specs/ws-wsrf.pdf.
5. I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. 2002. http://www.globus.org/research/papers/ogsa.pdf.
6. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200 – 222, 2001.
7. T. Imamura, B. Dillaway, and E. Simon. XML Encryption Syntax and Processing, W3C, 2002.
8. S. Jackson and R. Lepro. Usage Record – XML Format, Global Grid Forum, 2003.
9. G. Della-Libera, B. Dixon, P. Garg, and S. Hada. Web Services Secure Conversation (WS-SecureConversation), Microsoft, IBM, VeriSign, RSA Security, 2002.
10. T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo. An OGSA-Based Accounting System for Allocation Enforcement Across HPC Centers. Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04), ACM, New York, USA, November 15-19, 2004 (to appear).
11. T. Sandholm and J. Gawor. Globus Toolkit 3: A Grid Service Container Framework, 2003. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf.
12. SNAC - Swedish National Allocations Committee, 2004. http://www.snac.vr.se/.
13. SOAP Specifications, 2004. http://www.w3.org/TR/soap/.
14. Sun's XACML Implementation, Sun Microsystems, 2004. http://sunxacml.sourceforge.net/.
15. SweGrid, 2004. http://www.swegrid.se/.
16. S. Tuecke, K. Czajkowski, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open Grid Services Infrastructure: Version 1.0, Global Grid Forum, 2003.
17. Web Services, 2004. http://www.w3.org/2002/ws/.
18. Web Services Description Language (WSDL), 2004. http://www.w3.org/TR/wsdl.
19. WebServices – AXIS, 2004. http://ws.apache.org/axis/.
20. XML Path Language (XPath), 2004. http://www.w3.org/TR/xpath.
21. XML:DB Initiative, 2004. http://xmldb-org.sourceforge.net/.