# Authorisation and Accounting Services for the World Wide Grid

This thesis is
Submitted to the
School of Computer Science
and Software Engineering
for the degree of
Master of Science
of the
University of Western Australia

By

Alexander Barmouta

June 2004

Supervisor: Chris McDonald

# Abstract

Computational Grids are emerging as a new programming paradigm and infrastructure for Internet-based parallel and distributed computing. They enable the sharing, exchange, discovery, and aggregation of resources distributed across multiple administrative domains, organisations and enterprises. To accomplish this, Grids need infrastructure that supports various services: security, uniform access, resource management, scheduling, application composition, computational economy, and accounting. Although several initiatives are engaged in the development of Grid technologies, accounting for Grid resources has been on the ad-hoc basis and there is a need for a worldwide standard. To address this need, a new infrastructure named GridBank is presented, which provides services for authorisation, payment, accounting, and audit.

The support of computational economy and accounting services can lead to a self-regulated accountability in Grid computing. It is critical for providing a user-centric allocation of the available Grid resources to users that need them based on the user's Quality-of-Service requirements such as deadline and budget. Accounting is crucial for production deployment of Grid applications, supporting payment or barter for the use of computing resources and providing incentives to the owners to contribute idle capacity of resources to the Grid.

This thesis identifies the key requirements of Grid authorisation and accounting, and proposes Grid Accounting Services Architecture to meet them. A number of payment protocols are discussed and adopted to suit various trust relationships between Grid Service Consumers and Grid Service Providers. Furthermore, co-operative and competitive economic models are evaluated with respect to exchange of Grid services for currencies and an infrastructure named GridBank is described. GridBank allows unambiguous recording of resource usage and service cost. Grid Service Consumers use it to gain access to heterogenous computing resources dispersed around the world. Grid Service Providers conduct accountability audits by requesting GridBank statements.

# Acknowledgements

I would like to thank my supervisor Chris McDonald for his wisdom and patience. His extensive and invaluable academic experience has made my journey a pleasurable and a thought-provoking one.

I would like to thank Rajkumar Buyya of the Melbourne University for all his contributions and expert advise. I thank him for hosting my visit to Melbourne initially in May 2002, and then again in June 2003. I would like also to thank him for co-authoring an article with me titled "GridBank: A Grid Accounting Services Architecture for Distributed Resource Sharing and Integration" [BB03]. His writing and editing skills were of enormous assistance as well as helping to search for a suitable conference.

I would also like to thank members of the GRIDS laboratory at the Melbourne University for their warm welcome and technical support during my visits. I would like to personally thank Srikumar Venugopal and Jia Yu for their constructive comments. I wish them very best in completing their PhD dissertations.

Finally, I would like to thank Alexandre Chagoubatov, the Master of Commerce in Marketing candidate at the Business School of the University of Western Australia, for his expert advise on economic and marketing strategies. I wish him a successful outcome on the assessment of his thesis.

# Table of Contents

# Table of Figures

# CHAPTER 1
# Introduction

## 1.1 Motivation for Computational Grids

Electrical power is one of the basic building blocks of our modern day society. All electric appliances in our homes require electricity to run. Electricity Grids emerged at the beginning of the nineteenth century as the infrastructure to provide reliable, pervasive, consistent and affordable access to electrical power. Nuclear, solar, hydro, wind and conventional fuel power stations convert heterogeneous types of natural energy to electricity. Electricity corporations buy the power from the providers and manage it on its way to the consumer. Such management involves technical aspects such as load balancing, routing, distribution networking, emergency procedures, as well as economical aspects such as metering, accounting and charging policies. Local Grids owned by one company are combined to form regional Grids, which are combined into national Grids, which in effect constitute the global Grid, owned and managed by corporations with their own administrative control and policies. Economic principles have been successfully applied to managing electrical power.

Driven by this analogy from the electricity power Grid [CB02] and building on top of the World Wide Web, *computational power Grids* are emerging as a new computing paradigm for Internet-based parallel and distributed application processing [BFPT00, CKKG99, FK97, FK98b, FK99, FKNT02, FKT01, TCFF02]. The motivation for computational Grids is driven by computationally and data intensive applications, which demand more computational power than is available at a maximum capacity of resources (PCs, clusters, supercomputers) in a single administrative domain. Another reason is the dissemination of enterprise data distributed across multiple administrative domains while preserving data integrity and consistency.

The idea of distributed computational and data processing is not novel. Cluster systems such as Condor [FTLF01] have been researched and developed for the last couple of decades. Remotely submitting jobs to stand-alone workstations and supercomputers has been a common practice since the advent of computer networks. Similarly, the generation of electricity was possible about 100 years ago, but the need for each user to build and operate a new generator hindered use. The greatest merit was not the discovery and utilisation of electricity itself, but the electricity grid that allowed pervasive, consistent, and affordable access to it for everybody.

The World Wide Grid concept was formulated in the mid-nineties of the last century to define the architecture and to develop the corresponding infrastructure that consolidate heterogeneous computing and data resources distributed around the planet and beyond (one can envision a high-performance computing station built on Moon, which is connected to Earth via satellites), and present them as one unified powerful resource accessible via standard interfaces. Why would any organisation consider sharing computing facilities with others and why would it utilise resources offered by others?

The answer lies in the concept known as economies of scale. Building a multi-billion dollar electricity plant is beyond capability of any organisation. Similarly, buying a supercomputer is infeasible for most businesses that perform computationally intensive tasks. Combining resources into compute farms, which are shared and utilised more efficiently, saves money and offers orders-of-magnitude improvements in underlying technology, which enables revolutionary, often unanticipated, applications of that technology.

Distributed supercomputing is one of those applications. For example, the quantum chemical reaction dynamics simulation performs calculations of cross-sections of chemical reactions by solving the six-dimensional Schrodinger equations for the triatomic system [FK99]. The prediction of the rates and cross-sections of such reactions is a primary goal of computational chemistry, which allows modelling of complex chemical systems such as combustion, plasmas, and the atmosphere. Global climate modelling is a major application, which has been shown to yield significant results by utilising heterogeneous systems of supercomputers connected by a gigabit-per-second network.

High-throughput Grid computing involves scheduling and execution of large number of independent jobs, with the goal of utilising unused processor cycles on stand-alone workstations. The microprocessor manufacturer AMD used this technique to analyse an enormous number of CPU designs spread over a thousand computers. Molecular drug design considers infinite number of combinations of basic compounds of which only a small fraction would be effective. Nimrod-G [BAG00, B02a] is a tool that automates such parametric studies and drastically reduces time-to-market of such products.

Data-intensive applications focus on acquiring and processing of vast amounts of data maintained in digital repositories, libraries, and databases. High-energy physics experiments, such as the Belle data Grid [BADG03], will generate petabytes of data, of which a large fraction has to be accessed to detect events of interests to scientists. Collaborations that access the data are geographically distributed and hence the data is expected to be distributed among participants.

A successful development and deployment of the World Wide Grid[1] requires a number of services. They include low-level services such as service discovery (information directory), security, resource management (trading, allocation, submission and Quality-Of-Service), data management, and high-level tools for Grid application development, resource brokerage, and scheduling. Provided that all of these technical issues of Grid application deployment are solved, one question remains: how does a user acquire access to the resources if they are owned and managed by organisations that the user has neither prior knowledge of, nor any authority to use their computing facilities. This thesis proposes an infrastructure that provides global resource access authorisation based on payment and accounting practices in the real world.

## 1.2 Market-based Grids and their Accounting Limitations

Grids enable sharing and aggregation of geographically distributed heterogenous computing resources owned and managed by multiple administrative domains with their own security and scheduling policies. Resource management in such an environment is a complex undertaking. The computational *service providers* (resource owners) and *service consumers* (resource users) have different goals, objectives, strategies, and Quality-of-Service provisions.

[1] The term “Grid” is used in the rest of the thesis to mean computational, data or specialised Grid

Traditional scheduling algorithms are primarily concerned with guaranteeing fairness in executing jobs waiting in a queue of a resource management system. The First-In-First-Out (FIFO) model, although fair for participants with equal priorities, is not suitable in most Grid environments [SALH02]. Some jobs submitted to a resource are considered to be of a greater importance. The priority of a job is defined by the organisational politics, influenced by administrators and managers according to the position or role of each member within the organisation. Various priority queue models [FTLF01, PBS03, SGE03, ZZWD93] have been proposed, which are sufficient within a single co-operative domain. In a World Wide Grid environment no assumption can be made about the nature of the organisation of computational resources. In fact, it is most likely that different commercial organisations will compete against each other for the best resources. For example, the most successful genetic engineering enterprise will be the one that can compute DNA sequences fastest among other factors that influence success of any information technology-based business.

One approach to tackle the resource allocation and job priority problem on the World Wide Grid is to institutionalise economic principles into the management of computational resources. Such a concept involves introduction of computational tokens or currencies, which reflect the market value of each resource. Once all resources are priced, the scheduling problem is reduced to efficient utilisation of the best and cheapest resources available on the market at the time. The idea of applying economics to resource management in distributed systems has been explored in a number of systems and architectures [BAG00, BMBB02, BV01, B02a, GB03, MD98]. Unfortunately, many of them have mentioned but do not address the problems of authorising access on the World Wide Grid, accounting for resource consumption, and auditing to determine which allocations have been utilised.

Grid audit allows clients to evaluate application performance and adapt to changes in resource availability and costs dynamically [B02a]. Resource providers can analyse demand for their resources and adjust prices accordingly. In order to conduct the audit there is a need for a World Wide Grid accounting infrastructure. Most resource management systems such as cluster systems (chapter 2) and operating systems log accounting information against local system accounts. The resources they account for, such as CPU time, main memory, secondary storage and network consumption, vary from one system to another. Also the internal format of the accounting data is different between the systems.

The criteria for the commerce on the World Wide Grid are:

Ability to access resources in any administrative domain that enforces its own local security policy,

In the economic context provide payment for the resource access authorisation such that the payment can be exchanged for another currency or service,

Account for consumed resources by accurately recording exactly what has been used, how much, and how long for, and

Ability to conduct reliable audits of resource usage.

## 1.3 Grid Authorisation, Payment, Accounting and Audit

The area of computational accounting has not been formally researched in traditional computing environments. Such environments usually span only one organisation or administrative domain and resource usage is recorded against local system accounts. Such accounts are local to the site (administrative domain) and all accounting information recorded against the accounts is often in a proprietary format, which cannot be easily shared with other sites.

Collaborating organisations overlook the importance of auditing access to and usage of their computing facilities since they own resources and are more concerned with performance issues. Grid middleware, such as the Globus Toolkit [FK98b, FK97, FK99], provides security, but once users and resources are authenticated, access is authorised by consulting a list of authorised users that every provider must maintain. Community Authorisation Service (CAS) offers a centralised access control, but does not account for consumed resources [PKWF03, PWFK02].

A single site relies on traditional scheduling algorithms, which allocate resources according to user priorities assigned by domain administrators. Whereas co-operative organisations might neglect resource accounting, this is certainly not true for a competitive or commercial environment. As soon as a resource becomes overloaded, there is an issue of how to reduce demand for it.

In a co-operative environment decreasing priorities of some users provides a short-term solution, but can be potentially damaging, especially if the resource is overloaded only at certain times. Therefore it can lead to under-utilisation. Accounting information can point out busy periods and administrators can restrict access only during those periods.

In a competitive environment the supply and demand can be regulated by adjusting resource cost. The price can be defined similar to user priorities. Important users should be assigned more resource tokens or currency.

Whether organisations collaborate or compete for computing power, they covet a common accounting infrastructure to monitor resource loads. The current practice of collecting accounting information involves manual examination of operating system logs on each machine or cluster on the Grid, hindering the audit procedure [M01].

To overcome these limitations the thesis proposes the Grid Accounting Services Architecture (GASA) [BB03]. It provides authorisation, payment, accounting and auditing services at the Grid middleware level, which is elaborates upon in chapter 2. The resource access authorisation procedure is similar to the Community Authorisation Service, and is very simple – if both a service consumer and a service provider have accounts with any accounting branch, then the consumer is authorised. CAS can be integrated with GASA for more sophisticated access control.

Grid Accounting Services Architecture provides payment protocols specific to Grid computing. A number of payment mechanisms from existing payment systems are surveyed in chapter 2 and are adapted to Grids in chapter 3. The three proposed payment protocols derive two payment modes: direct and deferred. A deferred payment is similar to the use of a credit card from actual banking. A customer presents the credit card to a merchant, and the merchant requests the bank to make the payment. The banks claim aggregated amounts from each other periodically if the customer and the merchant have accounts with different banks [CSHM02]. However, a service provider might not trust a service consumer or might not trust the service consumer's bank, and requires a cash payment. For such cases an electronic cash direct payment protocol is provided, which withdraws the funds from the buyer's account, transfers the amount to the seller's bank, and the seller's bank immediately deposits the funds into the seller's account.

The World Wide Grid consists of millions of resource consumers and providers. An accounting infrastructure must be scalable such that as the number of users increases, an accounting system performance is decreased linearly in proportion to the participants. GASA's scalability is achieved by establishing multiple accounting servers. Each branch serves a Grid community small enough to offer a minimal response time for every transaction. Branch administrators negotiate financial agreements such that inter-branch transactions can proceed without accounting servers needing to contact each other at the time of a transaction. Transactions are accumulated and are reconciled at the end of a financial period. Different currencies can be used, whether actual or virtual, and an exchange rate infrastructure performs real-time currency conversion.

GASA protocols allow users from one Grid community with an accounting server to access resources managed by another branch as long as the two branches authorise each other. Branch authorisation is performed by administrators who must decide what currency and exchange rates to use. In a direct payment mode, a service consumer contacts the local accounting server that immediately transfers the funds by contacting the foreign branch, which contacts the resources and authorises access for the user. In a deferred payment mode, the local branch directly contacts the foreign resource to authorise access and accumulated inter-branch transactions are reconciled later. In both modes, the service provider produces a Resource Usage Record (RUR) [BB03], which is a common Grid middleware format for resource accounting data. RUR is stored at the foreign branch.

## 1.4 GridBank

GridBank [BB03, B02a] is an implementation of the Grid Accounting Services Architecture. It is a secure authorisation, payment, accounting and audit system leveraging Grid and Web services technologies. The Globus Toolkit [FK98b, FK97, FK99] was chosen to provide secure authentication and resource management services because of its modular architecture. Grid components can be developed by independent researchers and easily integrated with the toolkit. Other Grid middleware systems, such as Legion [CKKG99] and Condor-G [FTLF01], follow a monolithic or vertical integration approach to their implementation and are hard to extend and integrate with third-party components. GridBank adheres to the Grid philosophy adopted by the Open Grid Services Architecture (OGSA) [FKNT02], which the Globus Toolkit implements. Such a philosophy has the disposition towards a single sign-on policy, which requires a Grid user to authenticate only once, and the user can acquire access to multiple resources across multiple sites (administrative domains), as well as delegating a subset of access rights to other users or resources acting on the user's behalf. In addition, the current trend in Grid computing is towards a Web services implementation of OGSA using Simple Object Access Protocol (SOAP) for invocation of Grid services and XML for self-describing interfaces and external data representation.

*Grid Service Consumers* (resource users) and *Grid Service Providers* (resource owners or administrators) open accounts with GridBank with the help of a GridBank administrator. A Grid Service Consumers is authorised to access a Grid Service Provider if both have accounts with any GridBank branch. The consumer's and provider's accounts need not be with the same branch. Payment is considered as the resource access authorisation in the context of GASA. Using any one of the payment protocols, a service consumer allocates a budget for a job or a set of jobs and submits them normally through the Globus middleware. At the end of a computation the accounting data is collected and forwarded to GridBank for storage.

GridBank customers can easily conduct audits by requesting GridBank statements for specified periods. They can only access those records that have their identities. The X.509v3 certificate subject name [FFW98] is considered as the globally unique Grid-wide identification of a Grid participant. The public key infrastructure already exists as part of the Globus Toolkit (i.e. Grid Security Infrastructure [FKTT98]). GridBank administrators have the authority to request a statement listing all usage records for the specified period for the site managed by the GridBank branch. Since statements are in XML format, resource usage records can be freely shared between sites and have the advantage of being human-readable.

## 1.5 Thesis Organisation

The thesis has been organised as follows:

Chapter 2 is composed of two parts. The first part categorises Grid technologies. A selected number of systems that compose the Grid paradigm are examined next with the emphasis on security for its importance in payment processing. The second part discusses application of economics to Grids. The Grid Architecture for Computational Economy [B02a] suggests an accounting infrastructure as one of the future directions of research. Payment mechanisms and existing Grid accounting systems are presented next highlighting their inadequacies for the World Wide Grid.

Chapter 3 conceptualises the banking and accounting principles and presents an authorisation, payment and accounting architecture named Grid Accounting Service Architecture for the World Wide Grid. The architecture is generic enough to be applied to any kind of computing device, which has the abstraction of local account for the purposes of local process creation and local file access control. Grid Accounting Services Architecture extends Grid Architecture for Computational Economy such that Grid participants can operate with actual or virtual currencies, offering incentives to resource owners to contribute their resources to the Grid.

Chapter 4 presents an implementation of GASA named GridBank. GridBank is used in conjunction with the Globus Toolkit version 2.0, but can be easily extended to other versions of the toolkit because of its modular design. System components are thoroughly described and all database tables, external interfaces and data representations are presented in the Appendix.

Chapter 5 summarises and concludes the thesis and future research directions are presented thereafter.

## 1.6 Summary

High-performance computationally and data intensive applications promote a new computing paradigm for Internet-based parallel and distributed computing. Heterogenous types of computational, data and specialised resources distributed geographically can be accessed by clients through middleware components researched and developed by the international Grid community. The successful deployment of Grids depends on numerous factors among which is the secure access authorisation and accounting [FK99].

Managing access to computational and data resources distributed geographically and across multiple administrative domains is a complicated task. Market-based Grids solve this problem by requiring a payment. Grid transactions involving payment need an accounting infrastructure that stores resource usage and issues bills on demand. The Grid Accounting Services Architecture provides well-established payment protocols and enforces resource usage data collection for audit purposes. An implementation of GASA named GridBank is a Grid service used by service consumers to acquire access to resources and gather resource consumption statistics for formulation of pricing and scheduling strategies.

In order to develop a World Wide Grid authorisation and accounting infrastructure it is important to understand what types of resources the Grid is composed of and what each resource does. Besides technical issues the question is who authorises resource access and whether it can be performed on-demand. The next chapter presents an assorted review of existing Grid technologies, payment protocols, and existing Grid accounting systems in order to build a market-based Grid.

# CHAPTER 2

# Grid and Economics

## 2.1 Introduction

The last decade has seen a boom in the global networking infrastructure. Internet technologies contributed towards the success of the World Wide Web that allowed simple, uniform and pervasive access to multimedia content shared globally. A vast number of on-line databases provide an enormous amount of information that was previously kept on paper or on private computers. However, if the data extracted from a database require further computational processing, the client has to manually log in to compute nodes, manually copy the data and start the application on each node. Such an approach is time-consuming and error-prone, as the user has to remember several passwords and other configuration information.

The World Wide Grid concept was formulated in the mid 1990s to mimic electrical power grids and provide affordable and easy access to computing power in wide-area networks. The ubiquity of the Internet and the availability of powerful PCs and high-bandwidth low-latency networks as low-cost commodity components are altering the computing landscape and the way the scientific and commercial research is conducted. Grid computing aims at coupling geographically distributed heterogenous computing resources such as PCs, clusters, supercomputers, data stores and various instruments, and present them as one unified resource with common access methods.

Several initiatives have been established that employ the use of computational Grids. These organisations form Virtual Organisations (VOs) consisting of multiple administrative domains with computing resources that are shared. Parallel and distributed applications are executed on the high-performance hardware. In the past the supercomputing paradigm was dominated by the up-market state-of-the-art parallel CPUs and massive memories of supercomputers. Today the trend is towards combining inexpensive desktop PCs into clusters.  Grid middleware provides common interface to uniformly access all of these types of distributed resources: personal computers, clusters, and supercomputers.

In this chapter several concepts are reviewed. Firstly, Grid software systems are classified and a few examples are investigated. The Grid technologies reviewed in section 2.2 are chosen because they have been integrated with the Globus Toolkit [FK98b, FK97, FK99]. As part of the toolkit, the Globus I/O library used in the construction of GridBank (chapter 4) builds on top of the Grid Security Infrastructure, which is described in section 2.3.1. A Grid resource access authorisation system named Community Authorisation Service is examined in section 2.4.1. Secondly, economic concepts from macroeconomics are presented and applied to computational resource management in the context of computational Grids. Payment systems for e-commerce and existing Grid accounting systems are reviewed thereafter.

## 2.2 Grid Technologies

The Grid is not a monolithic concept. It divides various issues and concepts into the four basic Grid layers [FKT01]. Figure 2.1 depicts them.

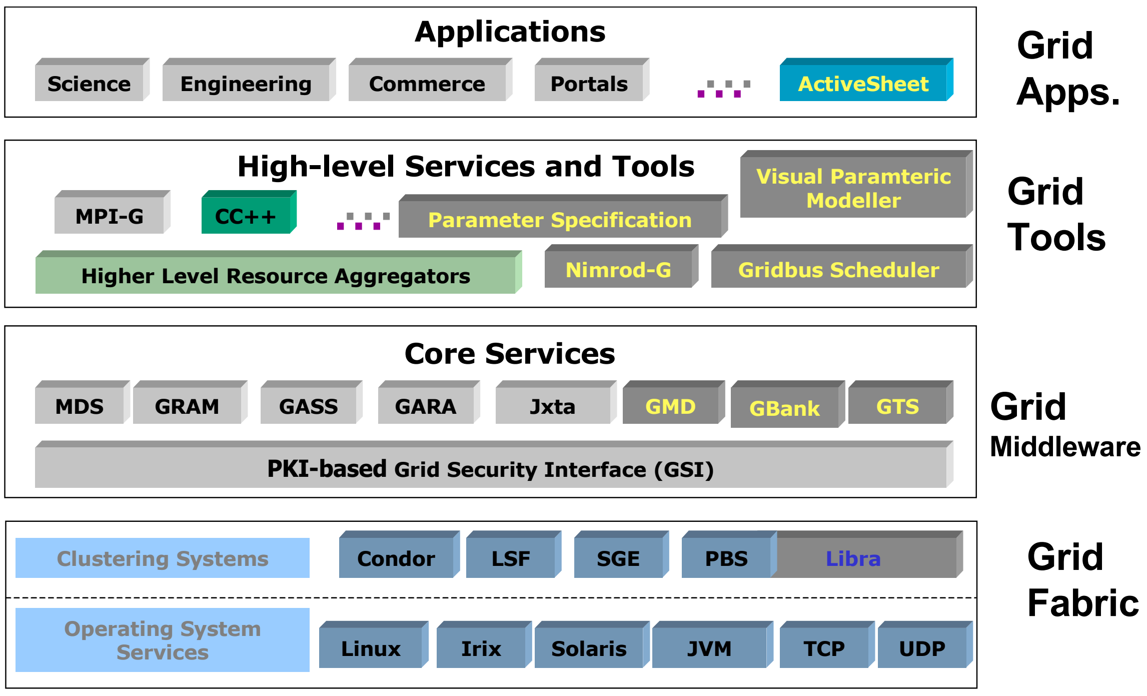


**Figure 2.1: Grid Software Architecture**

At the very bottom, the Grid Fabric layer consists of the physical hardware and associated local resource management systems. These systems range from single workstation operating systems to a variety of cluster management systems as well as other services typically provided by the operating system. The goal of these services is to schedule and dispatch tasks to local hardware devices. They are local in the sense that they reside within the same computer architecture, or on the same LAN (Local Area Network), but within a single administrative domain.

The Grid Middleware layer contains the majority of protocols that allow clients to use common connectivity mechanisms to access a Grid service. Connectivity issues range from the choice of transport protocols to use (e.g. TCP/IP), the authentication scheme (e.g. Public Key Infrastructure), the authorisation system to use in conjunction with the authentication and, finally, how to specify service requests and responses with a common syntax. The middleware layer has one goal to achieve: to connect geographically distributed heterogenous computing devices (PC, clusters, supercomputers, instruments) and to facilitate a uniform access to these resources via common interfaces and protocols.

The low-level middleware provides the means to execute a single job, whereas the high-level middleware provides Grid application development and deployment tools for submission of multiple jobs. The job management involves resource discovery to locate appropriate computing resources for a particular application, mapping each individual job to some selected resource, submitting and monitoring the execution of every job, and optionally restarting those that failed. The goal of the Grid Application Development Tools layer is to simplify the development and deployment of applications by detaching the issues involved in resource aggregation from application programming.

At the very top, the Grid Application layer consists of various proprietary Grid applications and other Grid-enabled compute and data intensive programs. They include computational problems in biology, chemistry, physics, pharmaceutical research, DNA sequence modelling, weather prediction, financial analysis, and other applications requiring globally distributed data and computation nodes. The nature of these applications allows the computation to be partitioned into parallel jobs. The computational Grid is the ideal computing paradigm for such tasks.

In the following sections the selection of Grid technologies is examined by identifying key components and exposing architectural decisions for component interactions. Such decisions determine the system's advantages and disadvantages with respect to distributed system issues such as scalability, performance, security, fault-tolerance, compatibility that leads to usability, and many other considerations such as Quality-of-Service. The research on Grid accounting enhances usability of tools and services provided by the Globus Toolkit, which has been integrated with the Portable Batch System (section 2.2.1), Load Sharing Facility (2.2.2), Condor (2.2.3), Sun Grid Engine (2.2.4) and Libra (2.2.5) cluster management systems. By looking at these systems, a set of guidelines can be drawn for designing a World Wide Grid accounting infrastructure, in particular, what accounting statistics do these systems provide already, and which accounting attributes are essential for World Wide Grid scheduling, audit, security, and other applications.

### 2.2.1 Portable Batch System

The Portable Batch System (PBS) is a resource management system for a single computer system or a cluster of computer systems. Therefore it forms part of the Grid Fabric layer. It was originally developed for NASA between 1993 and 1997 because existing operating system resource management facilities were inadequate or inefficient for parallel and distributed computations conducted by NASA. The goal of the system was to develop fully customisable independent system components to accept, store, schedule, and execute batch jobs [PBS03].

The PBS consists of four major components: the Job Client (commands, API), the Job Server, the Job Executor and the Job Scheduler. The Job Client is installed on the requesting machines and allows users to submit, modify, monitor and delete jobs. The Job Server is the central component of PBS and its main functionality is to receive and create job batches, modify an individual job upon client's request, protect jobs against system crashes, and place it into execution. The Job Executor is installed on every host that accepts jobs from the Job Server. The Job Executor presents the execution platform in a uniform manner to the Job Scheduler. Job Scheduler is the module that decides which jobs are to run when and where as well as when to stage in the executables and data, and then submits them to the Job Server via the same API as used by the Job Client. In other words, Job Scheduler appears just like a client to the Job Server, but executes with the privileges of an administrator.

Batches of jobs are submitted via the Job Client. Upon receiving them, the Job Server saves job requests and sends scheduling commands to the Job Scheduler. The scheduler requests resource information from the Job Executor(s) and job resource requirements from the Job Server. If there are multiple machines, their host names are provided to the scheduler. It makes a policy decision on what job to select next and sends execution commands to the Job Server, which dispatches the job to a particular Job Executor.

The Job Scheduler, being implemented as a separate component, allows for different scheduling algorithms to be deployed transparently. The default scheduler employs a simple FIFO (First-In-First-Out) queue model and can be replaced by a user-defined scheduling policy. More sophisticated models with job routing can be implemented in a variety of languages, including C, Tcl, and BaSL (a C-like language for writing scheduling policies).  A job scheduler only needs to use APIs of Job Server and Job Executor.

The PBS has the ability to checkpoint a job and migrate it to another node in the advent that the machine it is executing on becomes unavailable (e.g. owner log-in). Job migration is performed by the Job Server. The PBS builds on top of an operating system (e.g. Linux, Windows) and is a distributed system since each component (Job Client, Job Server, Job Scheduler, Job Executor) does not need to reside on the same host.

The PBS provides a local authentication and accounting, and has been integrated with Globus middleware. The accounting information is, however, specific to the system and is in proprietary format. It needs to be converted into a standard format that can be shared and recognised by other sites that use different cluster management systems.

### 2.2.2 UTOPIA: A Load Sharing Facility

UTOPIA, also known as the Load Sharing Facility (LSF), is a Grid Fabric resource management system similar to the PBS. It is a general-purpose cluster system requiring little or no changes to existing applications for them to be run remotely. Remote execution is transparent, that is, there is no difference between a job executed locally or remotely. Scheduling algorithms respond to load changes dynamically and exploit the full potential of the cluster by seeking out idle or lightly loaded resources. The LSF's scalability allows the system to be employed on systems with thousands of nodes [LSF03, ZZWD93].

The LSF's modular architecture identifies four major system components: the Load Information Manager, the Remote Execution Server, the Load Sharing Library, and the load sharing applications.

The Load Information Manager (LIM) module resides on every host. It collects each node's load information from the node's operating system and disseminates it to other nodes for scheduling decisions. The LSF system scalability depends on the complexity of algorithms that are used for current load data exchange between hosts. A naïve algorithm is for each host to send its current load information to all other nodes. In a system with thousands of hosts the network would be quickly congested with load information messages. The LSF employs a *clustering* technique to minimise network overhead by organising nodes in the system into clusters. A cluster has a master LIM node. Each slave node sends its load information to the master LIM, which is informed about every node in the cluster. Host selection process chooses an optimal node for a particular task by contacting only master LIMs in all clusters. Therefore all nodes are considered by receiving information about every host (from each LIM) in every cluster (all LIMs).

The Remote Execution Server (RES) module resides alongside the LIM on every node and provides an API and protocols for transparent remote execution of arbitrary tasks. Its primary purpose is to create each task's execution environment on the remote host and emulate I/O behaviour as if the task is being executed locally. UTOPIA provides two models for remote execution. In the *remote operation model* the RES server on the remote machine is contacted and the job is submitted once after the execution environment is established. Alternatively, the *client-server model* allows the RES to create a dedicated server process that can accept a series of tasks to be executed in the same environment, avoiding the overhead involved in contacting the RES each time to create the execution environment. The overhead is incurred only once during initialisation.

The Load Sharing LIBrary (LSLIB) module uses the services of the LIM and the RES modules to provide a uniform API for developing load-sharing applications. This module acts as a client to applications and facilitates job placement by selecting a node using the LIM API, and then submits the job to the node via the RES module. The scheduling decisions made by LSLIB are based on resource availability and task requirements. The resources available, reported by the (master) LIMs, are matched against application resource requirements. The LSF's modular architecture allows applications themselves to make scheduling decisions, therefore the development of application tools such as a scheduler is possible. A user-defined scheduler uses the RES API to dispatch jobs and the LIM's node load information for the node selection decisions.

The LSF supports only initial job placement, thus jobs cannot be migrated. It does, however, provide a uniform file name space within the system, allowing the executable and the data to reside on any node. The system provides local accounting data that has to be converted to a middleware format, which can be shared with other Grid sites.

### 2.2.3 Condor

Condor is a high-performance, high-throughput computational system that harvests the power of idle CPU cycles. It is typically set up on a cluster of UNIX workstations. Instead of executing their jobs in the background, users submit their computationally intensive jobs to the master node, which dispatches all jobs to workstations that would otherwise be idle. When a workstation becomes unavailable, the Condor job is check-pointed and migrated to an idle node. If all workstations are unavailable, the job is saved to permanent storage until a node becomes free. Condor has been extended to handle job submissions to Grid-enabled resources using Globus Toolkit services [FTLF01].

The way Condor handles job allocations to resources is analogous to classified advertisements. Machines advertise their performance metrics such as their CPU speed, RAM, Hard Disk space, network and other I/O metrics with the *Match-Making* process. When submitting a job, users construct a resource request with the required Quality-of-Service parameters. Also there are several layers of priorities: the priority of the user (assigned by the administrator), who submitted the request, the priority of the resource requirements assigned by the user, and priorities that machines assign to their own advertisements.

The Match-Making uses a simple data model to represent the principles of the system and integrates the query language into the data model by allowing participants to publish queries (requirements) as attributes. Resource discovery is achieved through centralised querying with periodic soft registration, which means that an advertisement is removed from the registry if the update query does not arrive at the end of the registration period.

The Condor system has been modified [FTLF01] to support creation of several Condor sites with their own administrative control. There are no QoS guarantees across the sites. Condor can be considered as a computational Grid with a (one layer) flat organisation. It does not provide interfaces to easily access accounting information.

### 2.2.4 Sun Grid Engine

The Sun Grid Engine (SGE) was developed by Sun Microsystems for managing a cluster of computers. It provides all of the traditional distributed resource management functions such as batch queuing, load balancing, job accounting, user-defined resources, job control functions (submit, suspend, resume, terminate) and other cluster-wide resources (e.g. network file system) [SGE03].

There are four fundamental components in the system: master, execution, administration and submit nodes. The SGE has a central point in the system (the master node) that accepts job requests, schedules them using available resources, and submits them to execution hosts. The execution nodes are the machines that accept computational tasks. The submit nodes are used by users to submit job batches. Scheduling policies are defined and enforced according to a unique technique implemented by the administration module.

The policies are defined via a concept called *tickets*. An administrator can assign a different number of tickets to each of the four policy categories. The *functional priority policy* provides special treatment according to a job's affiliation with a certain user, group, or project. The *Share-based policy* dictates assigned resource entitlements, corresponding shares of other users and groups, past resource usage and current system loads. The *deadline policy* issues tickets to jobs that have a particular deadline to meet, whether it is a job with a high priority or a job that has been "starving"; job "starvation" occurs when high priority jobs keep postponing lower priority jobs and is resolved by increasing priority of "starving" jobs. The *override policy* requires manual intervention of a SGE administrator to increase priorities of a particular job type. Jobs score tickets depending on the job type, and each job type earns tickets in the four policy categories. Job priorities are directly proportional to how many tickets a job has collected. The number of tickets assigned to each policy determines the relative importance of policies for each job type.

When an execution node becomes overloaded, jobs can be migrated to another execution node. The master node introduces the central point of failure in the system, but there is an option to establish a shadow node. Such a node monitors the master node, and assumes the master node's role in the advent that the master node fails. Optionally, a few shadow nodes can be created, and to decide which one should become the master node each shadow node participates in the election by executing the distributed election algorithm [SGE03].

SGE provides an accounting infrastructure, but stores the accounting data in a proprietary format. Such a format needs to be converted into a middleware record that can be recognised and shared by a wider Grid community outside the system. Authorisation is achieved through creation of local accounts maintained by the system administrator.

### 2.2.5 Libra

Libra is an economy driven cluster resource broker that is currently implemented as a plug-in scheduler for PBS [SALH02]. Section 2.2.1 describes the PBS architecture, and in particular the scheduler module, which is the component that Libra does not replace, but rather complements by providing Quality-of-Service guarantees when scheduling jobs. The traditional scheduling approach as exhibited by a variety of scheduling modules that come with the PBS package, has been system-centric. Various PBS schedulers strive at maximizing system parameters such as the throughput and the response time of jobs on a particular platform. The Libra scheduler adopts an user-centric approach to resource scheduling by allowing user to specify QoS parameters such as the budget and deadline. The scheduler calculates whether a job can be completed by the deadline depending on the current PBS system load and the job's requirements. Once a job is accepted, priorities of jobs are based on how much each paid for the job; a job that paid higher price in terms of service rates (i.e. cost per unit of time) is selected over other jobs that paid less. In case another batch is submitted, Libra accepts only those jobs that can be completed by the job's deadline and have the highest price offers among others. Such resource allocation policy is motivated by the economic principle of higher bid wins.

Libra's scheduling algorithm deems the following assumptions critical to correct operation of the system. There has to be a centralised resource management component, which is the only component that makes scheduling decisions. Accepted jobs are the only ones executing on any node, therefore nodes are dedicated and are not shared with other jobs that the scheduler is not aware of. The operating system on the node on which the job is executed accepts a parameter that is the percentage of CPU cycles the job must be allocated, and must enforce this value in order to deliver the Quality-of-Service promised. The estimated execution time of a job must be known and is correct. These assumptions must be satisfied in order to deliver the QoS that might be required as per service-level agreement.

Libra is one of the few compute resource schedulers that is user-centric and provides Quality-of-Service support. The economically driven scheduling policy reflects the market value of the resource, i.e. how much the user is willing to pay. The implication is that a job with a tight deadline should cost more, since relaxing the deadline for a particular job allows other jobs with tighter deadlines and bigger budgets to proceed first [B02a].

## 2.2.6 Globus Toolkit

The Globus Toolkit is an open-source collection of protocols for wide-area processing [FK98b, FK97, FK99]. It provides middleware that assembles local Resource Management Systems into the World Wide Grid. It conceals the heterogeneity of the Grid fabric and provides common interfaces for secure computational job submission, data access, monitoring of progress and other distributed operating system services. The core of the toolkit is implemented in the C programming language and is intended for different variants of UNIX. The current implementation direction of the Open Grid Services Architecture (OGSA) is towards narrowing the gap between Grid and Web services, and APIs of the Globus Toolkit version 3 are currently being implemented in the Java programming language and XML to implement communication protocols to enable the middleware to run on any platform supporting the Java Virtual Machine. Regardless of the implementation the toolkit consists of the following generic architectural components.

The Meta Directory Service (MDS), based on Light-weight Directory Access Protocol (LDAP), is a Grid service discovery protocol used to locate and select Grid resources. It supports soft-state registration, which means that services periodically register themselves with the MDS server and, if at the end of a period a service has not re-registered, it is removed from the listings [CFFK01].

The Globus Advanced Reservations Architecture (GARA) addresses issues involved in dynamic discovery and advance or immediate reservation of resources. The architecture separates resource co-allocation problem into two distinct phases: reservation and allocation. During the advanced reservation stage a co-reservation agent assures QoS as specified by the application requirements. The allocation phase involves actual instantiation of reserved services. In the case of an immediate reservation, it is followed by the allocation [FKLL99].

The Global Resource Access and Management (GRAM) protocol ensures that a submitted job is allocated sufficient resources for its execution, monitors execution progress, and reports on the job's execution status (e.g. successfully completed, failed, suspended or active).

The Global Access to Secondary Storage (GASS) protocol is responsible for transferring data to and from remote locations. The protocol allows an executable and data to be staged-in to a compute server, and results, if any, to be returned to the client. GASS library provides the API for Grid applications to open files by specifying Uniform Resource Locator (URL) of the file, which achieves secure data access transparency [BFKT99].

The GASS protocol is sufficient for files of relatively small size as it uses the HTTP protocol as the underlying transport. The gridFTP protocol is built on top of the FTP protocol and is optimised for files of relatively large size. In fact, gridFTP is a GSI-enabled File Transfer Protocol, which means clients use the same Grid credentials to access FTP servers as they do to access other Grid services.

The Grid Security Infrastructure (GSI), described in more detail in section 2.3.1, provides middleware security protocols, which use Public Key Infrastructure with digital certificates as the global authentication scheme. The Globus I/O library uses the Secure Socket Layer (SSL) [SSLP96] technology, which has positioned itself as the dominant cryptography library for on-line secure transactions, such as Internet banking. Globus I/O authenticates the resource provider and consumer using their digital certificates and establishes a secure communication channel between the two. By default, all messages are passed in clear text but with Message Integrity Code (MAC) [B00] ensuring their authenticity. Optionally, messages can be encrypted for maximum security. Cryptographic strength can be varied between 512-bit, 1024-bit, 2048-bit and 4096-bit key lengths.

Other tools in the Globus Toolkit include data replica management protocols, the Dynamically-Updated Resource On-line Co-allocator (DUROC) that allocates multiple resources using GRAM to access individual resources, the Nexus communication library that provides various transport services such as multicasting while building on top of the Grid Security Infrastructure. It is envisioned by the Open Grid Services Architecture (OGSA) that new tools will be developed by independent providers that will cover other aspects of Grid resource management, including wide-area scheduling, resource brokerage, and accounting.

The success of the Globus Toolkit is partially attributed the fact that is has been integrated with each of Portable Batch System (and Libra with slight modifications), Load Sharing Facility, Condor and Sun Grid Engine cluster systems. The Globus community consists of independent research collaborations that share experiences and co-operate with community efforts. Being the largest community is one of the reasons that distinguishes the toolkit from other middleware systems such as Legion [CKKG99, HKFG00].

### 2.2.7 Nimrod-G

Nimrod-G is a wide area (Grid) version of Nimrod [ASGH95] – a tool that manages deployment of parametric experiments. Such experiments involve running the same executable with some range of input parameters. Nimrod has been applied to a range of application areas, including bioinformatics, operations research, electronic CAD, ecological modelling, and computer movies [BAG00, B02a].

The system architecture consists of the *Client/User Station,* the *Parametric Engine,* the *Scheduler,* the *Dispatcher,* and the *Job-Wrapper.* The Client/User Station acts as a user interface for controlling and supervising an experiment. It allows the user to specify the executable and the parameterised input data as well as other QoS requirements such as deadline and budget. The Parametric Engine accepts experiment requests from client stations and acts as a persistent job control agent that shepherds all individual jobs from creation to completion, restarting those jobs that have failed. The Scheduler is responsible for resource discovery, resource selection, and mapping jobs to resources. The Dispatcher initiates the execution of a task on the chosen resource as per scheduler instructions. It also reports the status of a job execution to the Parametric Engine. The Job-Wrapper is responsible for the staging of an application, i.e. uploading the executable and data to the compute node and downloading the results to the Parametric Engine via the Dispatcher.

The Scheduler uses Globus' MDS services to locate suitable resources when a user first requests a computational experiment, and GSI and GRAM (section 2.2.6) services to execute jobs via the Dispatcher. In effect, Nimrod-G implements resource brokering services specialised for parameterised Grid applications. It aggregates multiple resources and belongs to the Grid Application Development Tools layer (figure 2.1) as it eliminates the need to manually start every job on a compute node [B02a].

## 2.3 Grid Authentication

Authentication involves verifying identities of system entities such as users and machines. The simplest and vulnerable form involves a user typing in a password and it is transmitted in clear text over network and matched against a database.

Kerberos is a much more sophisticated system, employing secure protocols that allow users to identify themselves to the authentication server, which then issues credentials (Kerberos tickets) to access services [CDK01, RFC1510]. It relies on manual distribution of secret keys. System and network administrators enter users' secret passwords into the system's database.

Typically authentication involves the system sending an user an arbitrary message and requesting the user to encrypt it with the secret key. The system retrieves the corresponding key from the database, encrypts the original message and compares it with the one received from the user. If they match, the user is authenticated.

In global settings, with potentially millions of users and resources participating in the Grid, the distribution of secret keys poses an access scalability problem because secret keys must be manually distributed by human operators. It is desirable to automate the key distribution; therefore some other form of authentication is required. The Globus Toolkit solves the problem by providing a Public Key Infrastructure (PKI).

Grid participants are issued digital certificates with the corresponding public-private key pair. The private (secret) key is generated automatically and is known only by the certificate holder. The public key is contained within the certificate and is freely distributed among participants. PKI authentication is based on mathematical properties of large prime numbers. A message encrypted with the private key can only be decrypted with the public key, thus any participant can authenticate a message from a particular sender. A message encrypted with the public key can be decrypted if, and only if the user possesses the private key, thus any participant can securely send a message to the intended recipient.

The PKI authentication between two entities (user to service, service to service) assumes that they are issued digital certificates. Entity A challenges entity B by sending B an arbitrary message and asks to encrypt it with B's private key. When reply is received, A extracts B's public key from B's certificate and decrypts the received message. If the decrypted message matches the original unencrypted message, then B is authenticated.

Digital certificates are issued by third parties called Certificate Authorities (CAs). A CA is typically created for each Grid site, and the PKI authentication can only be achieved if participants have at least one CA they both recognise. The CA is the trusted third party that digitally signs all of the Grid participants' certificates and a certificate is validated by decrypting the signature with the CA's public key and comparing the result with the message digest of the certificate. A hierarchy of CAs can simplify the security management such that even if two entities belong to different CAs, they can still authenticate each other if their CA's certificates were signed by a higher-level CA that they both trust [FFW98].

A PKI scheme serves as the middleware security infrastructure at the Grid Middleware layer (figure 2.1). Its purpose is to interface with various local security systems (such as Kerberos) and to "translate" middleware security policies enforced by the Grid community into local security policies enforced by resource owners. The following section presents GSI's features that are essential for World Wide Grid computing.

### 2.3.1 Grid Security Architecture (GSA)

Grid security will have a major impact on the successful deployment of computational Grids. In order to encourage service providers to contribute their resources to the Grid they must be assured that all local security policies are preserved and that no unauthorised and potentially malicious access occurs. The Grid Security Architecture requirements are dictated by various factors. The potential Grid clientele and resource pool are large (in the order of millions) and dynamic (consumers and providers change frequently). Any computation started by a user may acquire and release additional resources dynamically. The processes comprising the computation may need to communicate, and thus need to authenticate each other if the computation spans multiple administrative domains [FKTT98].

These requirements are further refined into specific design goals. *Single sign-on* allows an entity to authenticate once (enter the password once), and access resources located on multiple sites without authenticating to each site, eliminating the need to log in to each site manually. User credentials must be thoroughly protected to avoid compromising them since processes acting on a user's behalf can potentially acquire a large number of resources from all authorised sites without user's consent. *Interoperability with local security* systems ensures access to local resources remains determined by the local security policy, rather than replacing it. *Uniform credentials and certification infrastructure* achieves a standard way of expressing the identity of a Grid participant. *Support for secure group communication* provides a secure data transport for processes in a computation that change dynamically and need to communicate with each other. *Support for multiple implementations* decouples the security architecture from the actual implementation, which allows for the implementation of the security policy with a wide range of cryptographic technologies.

With these goals in mind the security policy deems the following considerations essential to Grid security. Multiple trust domains, also known as Grid sites, each with their own Certificate Authority, enforce a local security policy, and the GSA has limited or no influence over it. Operations performed within a single domain are subject to the local policy only. Global identities (i.e. certificate subject names) are partially mapped to local resource accounts, which can be pre-defined for a particular user or allocated from a pool of "free" accounts, which are not associated with any particular user, but are allocated and de-allocated dynamically. An authenticated global identity mapped to a local account is equivalent to being locally authenticated as that local identity, and all local access control decisions are made on that basis. Interactions between entities on different sites require mutual authentication – one entity (e.g. user, service) authenticates itself to the second entity, and then the second entity authenticates itself to the first. A process executing on the user's behalf is allowed to delegate a subset of the user's access rights to another process to acquire additional resources without user intervention. Processes running in the same trust domain may share a single set of credentials for efficiency's sake, enabling the security architecture to scale well to large parallel applications that may spawn hundreds of processes on each site.

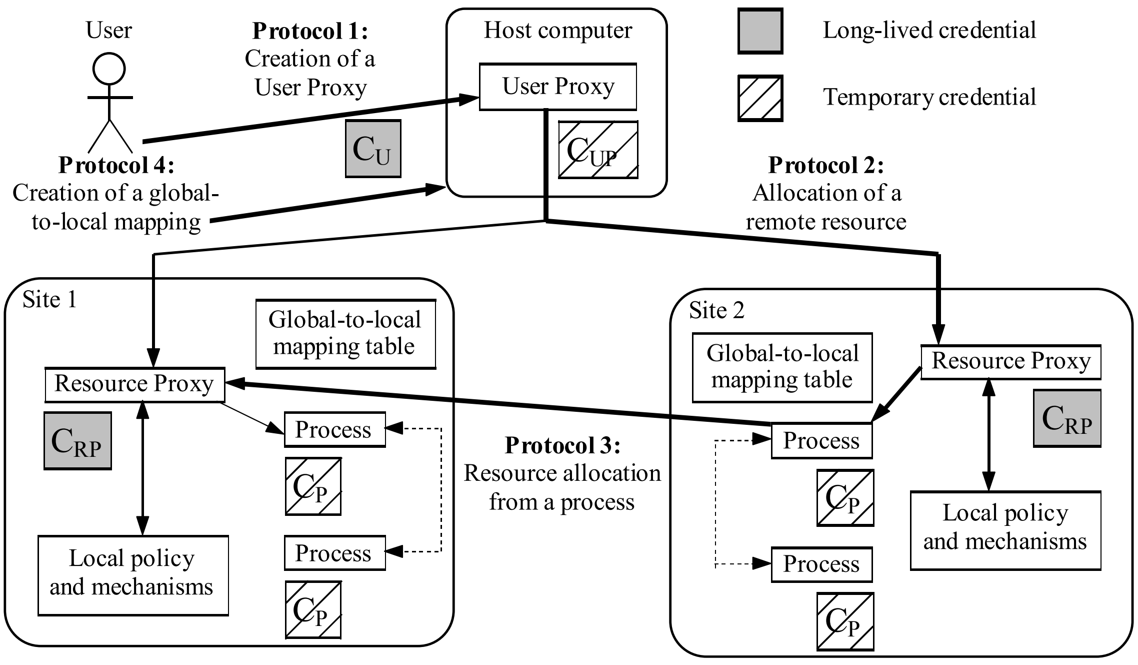


**Figure 2.2: Grid Security Architecture**

The protocols and entities interactions are shown in figure 2.2. Protocol 1 is used to create a *user proxy.* A user proxy is a session manager process, which is delegated permission to act on behalf of the user for a limited period of time. The process is issued a digital certificate valid only for the expected duration of the computation. The user signs the proxy certificate with the user's private key. The signature on the proxy certificate is verified by using the public key from the user's certificate. Once an user proxy is created, protocol 2 is responsible for authentication to remote resources. Each resource has a certificate and creates a *resource proxy* that has an equivalent functionality as the user proxy. User and resource proxies perform mutual authentication, and each resource proxy generates credentials for the remote processes. Protocol 3 allows one remote process to dynamically allocate additional resources on the user's behalf. Protocol 4 provides the partial mapping from the global identity (i.e. certificate subject name) to a local subject (i.e. local system account).

The Globus Toolkit implements the Grid Security Architecture and provides Globus Security Infrastructure (GSI), using the X.509v3 digital certificate format for encoding of credentials. The security protocols build on top of the Generic Security Services API (GSS-API) to separate the protocols from the underlying local security mechanisms such as Kerberos [RFC1510], RSA [RSA03], DES (Data Encryption Service), SSL (Secure Socket Layer) [SSLP96] and other future protocols. Figure 2.3 shows how GSI protocols from figure 2.2 interface with local security infrastructures, thus providing security middleware for computational Grids.

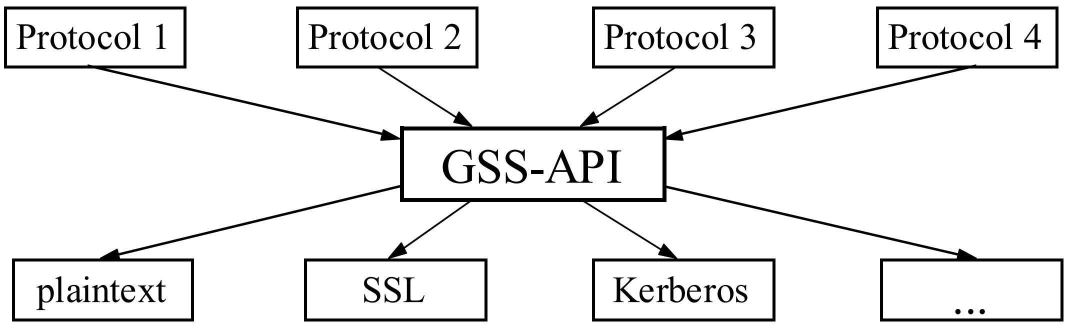


**Figure 2.3: Use of GSS-API in Globus**

The GSS-API defines an abstract interface to establish two-party security contexts (secure communication channels between two entities). Its functions provide for obtaining credentials, performing authentication, generating the MAC (Message Authentication Code) [B00], which preserves message integrity, and optional encryption and decryption of messages. The interface can be implemented by a variety of mechanisms shown in figure 2.3. The Globus Toolkit implements the GSS-API by the Public Key Infrastructure (with the X.509v3 digital certificate format) and the Secure Socket Layer (SSL) – a technology proven successful in financial transaction systems, such as Internet banking. The SSL protocol uses symmetric (secret) key cryptography and Public Key Infrastructure is used to exchange the (secret) session key. SSL uses TCP/IP sockets as the underlying transport, which can in principle be replaced by another reliable transport protocol.

## 2.4 Grid Authorisation

The Globus Toolkit uses X.509v3 digital certificates and private keys as credentials to access Grid resources [FKTT98, FK98b, FK99]. Once the entities are authenticated, the question is how do resource providers authorise the access? The current authorisation mechanism in the Globus Toolkit is such that each user has to be manually entered into the authorisation file on every resource. The authorisation file contains lines of text, each line consisting of a mapping from a certificate subject name to a local account name. The certificate name represents the global user identification whereas the local account name is only identifiable at the local Resource Management System, such as an operating system or the middleware systems surveyed in section 2.2.

Each resource server makes its own decision with regard to access rights to the service. The resource owner retains a fine-grained control of the resource by specifying local account permissions. The gatekeeper process, which is part of the Grid Security Infrastructure, is responsible for authorisation and ensures that the global user credentials (i.e. the certificate and private key) are mapped to the local security system credentials, such as a Kerberos subject, or to any other local security infrastructure credentials that are in place.

The implication of such an approach raises the issue of access authorisation scalability problem. When a new user joins a community of resource consumers and providers every resource provider must be notified. Although this method places unduly burden on resource administrators, it avoids a central point of failure and potential bottleneck issues present in centralised systems. However, managing a Grid with a substantial number of nodes rapidly becomes difficult, and centralised systems tend to achieve higher access authorisation scalability and flexibility in managing resources in dynamic organisational environments when employees change or are promoted to a different role with different access privileges. Instead of creating a new account for each new user, an account can be dynamically allocated from a collection of accounts with pre-defined permissions according to the user's group or role within the organisation.

### 2.4.1 Community Authorisation Service (CAS)

Originally, computational Grids were built for collaborations – groups of cooperating scientists solving computationally and data intensive problems in scientific, medical and financial research. Each group would have its own Grid with its own Certificate Authority (CA). Such collaborations are now known as Virtual Organisations (VO)

[FK99, PKWF03, PWFK02]. Each member of the co-operation is issued a certificate from the site's CA and can only access local resources.

Accessing resources located at another site with a different CA would require each user to have a certificate from the site, or each provider has to be able to authenticate using Certificate Authorities of all participating sites. Either approach is rather cumbersome. Community Authorisation Service builds on top of the Grid Security Infrastructure delegation mechanisms and allows resource providers to specify course-grained access control policies in terms of the community as a whole, delegating fine-grained access control policy management to the community itself [PKWF03, PWFK02].

Resource providers grant access rights to the community. The community decides what users and groups are granted what permissions to access what types of resources if the community's policies are compatible with the resource provider's policies. The policy itself can be of any format and is included along with the GSI credentials. Only the community recognises the format.

GSCs with certificates from their site contact the CAS server and request access to resources at other sites. Provided access policies are compatible, the CAS server delegates credentials corresponding to the community's access rights. Clients use the delegated credentials to authenticate to the resources. GSCs with certificates from one Certificate Authority can access resources from another administrative domain. Only the CAS server needs to recognise all CAs.

The CAS server maintains a database of resource consumers (usually people) and resource providers (machines). Each resource has a list of authorised users associated with it. When a resource consumer requests delegation credentials to access a resource, the list associated with the resource is consulted to determine whether the consumer is authorised. This approach works for Virtual Organisations with a few tens of users, but does not satisfy the needs of larger organisations with thousands of employees and compute nodes.

CAS has been recently extended to provide sub-VO (finer granularity) role-based access control. Authentication involves verifying an individual's (role's) identity and their membership in the VO. The two extremes, the individual and the organisation, distinguish two intermediate granularities of access rights called *roles* and *groups*, respectively. Each group might belong to different physical (actual) organisation that has a Grid with its own Certificate Authority. These sub-VO groups can be thought of as VO individuals and are aggregated into a larger VO, which uses the extended CAS to manage access control. A role is assigned to an individual within a group, and each group is assigned a role within the larger VO. A hierarchy of group roles can be composed this way [CCOT03].

## 2.5 Economic Models for Resource Sharing

Traditional models for exchange of goods and services following well-established Economics practices have been successfully applied to management of computational resources [B02a, BAG98, SALH02, MD98]. They include:

- Barter
- Bonds
- Currency (cash)
- Accounts

The barter model concludes the deal with both parties exchanging services of equal value. The value has to be agreed upon by both participants. This model does not suit commerce on the Grid because the relationship between the parties is asymmetric – the GSC consumes the service without providing any service in return to the GSP.

Bonds are written statements issued by service consumers that promise to deliver a certain value of services upon request. They are given to service providers in return for consumed services. These also do not quite suit the commerce on the World Wide Grid since the service provider does not necessarily trust the service consumer to honour the bond and only that particular service consumer can repay the bond.

Currency is identical to bonds but is freely accepted by everyone. It can be spent at any service provider and is issued by trusted authorities with sufficient resources to honour the value of the currency. Cash is a suitable model for exchange of Grid services but can be too anonymous if there is a need for resource access audit.

The accounts model is the most appropriate model. All transactional activity is documented and can be audited for security and other purposes. Resource consumption is recorded against accounts and, at the end of a financial period, accounts are reconciled by compensating providers in some way, for example, using currency. Cheques are signed authorisations that are issued against accounts and are redeemed by the banking system.

Another aspect of resource trade is whether it is on a co-operative or competitive basis. A co-operative service sharing could imply that the value of some resources is either underestimated or overestimated for the sake of co-operation. In a competitive environment resource value is determined by market trends. The final price in some (stable) currency is determined by various marketing models examined in the following section.

### 2.5.1 Grid Architecture for Computational Economy

The GRid Architecture for Computational Economy (GRACE) developed at Monash University in Melbourne [B02a] applies concepts from Economics to Grid resource management. It introduces trading models such as commodity market (supply-and-demand driven pricing), posted price, bargaining, tender or contract, auction, bid-based, co-operative bartering, monopoly or oligopoly, and other influences on market prices. In an actual free market economy stockbrokers monitor the prices and decide on when and what stock to buy or sell. In a similar fashion, a Grid resource broker automates Grid resource trading and hides the complexities involved in selecting the right resource at the right price. Nimrod-G is an example of a resource broker and scheduler for parameterised Grid applications (examined in section 2.2.7) that is driven by user QoS requirements and accomplishes these goals [B02a, BAG00].

The inter-disciplinary approach to Grid resource management and scheduling taken by GRACE strives to answer the following questions:

What comprises the Grid and who owns its resources?
What motivates resource owners to contribute their resources?

Is it possible to have access to all resources in the Grid by contributing our resource?

How does one have access to all resources if contributing our resource is insufficient?

Do resource providers charge the same or different price for different clients if economic principles are introduced?

Is the cost identical for peak and off-peak hours?

How can resource owners maximise their profit?

How can users solve their problems within a minimum cost and shortest deadline?

How can a user get priority over others?

Can the solution cost be reduced if users relax their deadline?

The solution depicted in figure 2.4 presents various components that resolve the aforementioned issues. A GSC uses services of a Grid Resource Broker to solve a problem as cheaply as possible and within the required timeframe. A GSP runs Trade and Accounting servers to maximise the resource utilisation by offering a competitive service access cost in order to attract consumers. Resource allocation and access is achieved by leveraging Globus tools [B02a].

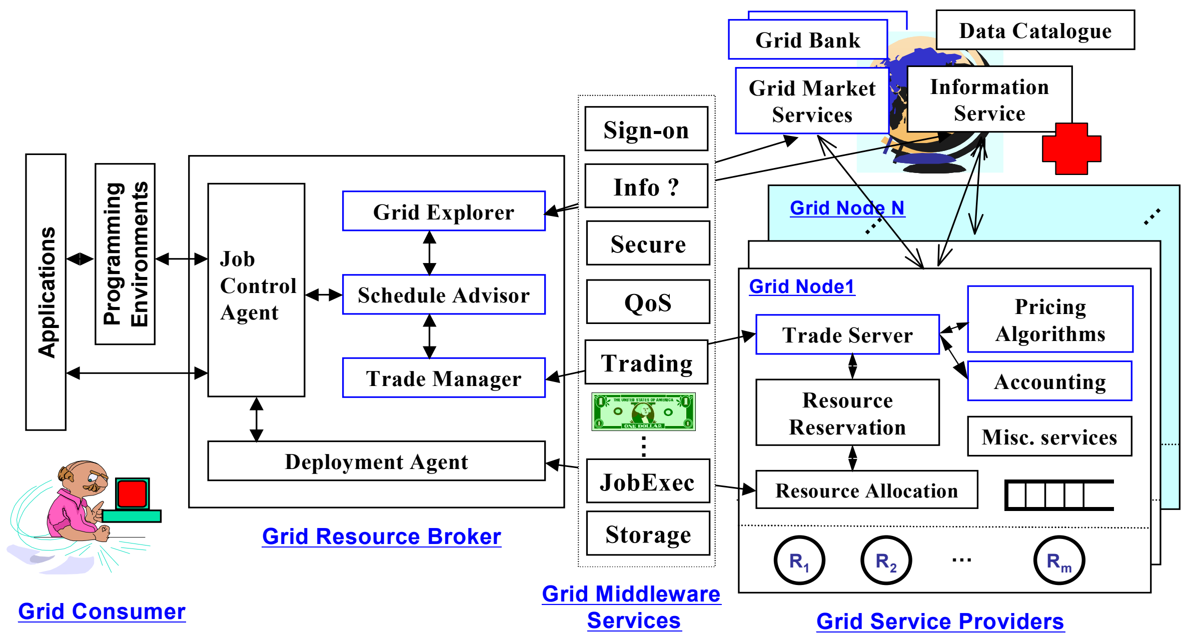


**Figure 2.4: GRid Architecture for Computational Economy (GRACE)**

GRACE presents Trade and Accounting Servers and GridBank as abstract entities required to establish market-based Grids. In particular, it emphasises computational resources that should be accounted for and payment mechanisms that need to be in place to achieve a trustworthy exchange medium. Although there are numerous electronic payment systems, no standard Grid accounting systems are in use today. Grid administrators tend to manually generate accounting data and charge using traditional general-purpose accounting packages such as MYOB [MYOB03].

## 2.5.2 Gridbus Project

The Gridbus (Grid business) project [GBP02] at the University of Melbourne has been undertaking research and development and leading the creation of a number of cluster and Grid technologies to realise service-oriented Grid and utility computing. The project focuses on the design and development of (i) tools that transform existing applications into master-worker style applications, (ii) high-level services that enable publication of application services in a market-like environment, (iii) Grid economy paradigm for distributed resource and user management, and (iv) resource aggregators that discover distributed data and applications services at runtime and map application tasks to resources based on their cost, capability, performance, and user's QoS demands such as the deadline and budget limits [GBP02, B02b].

The GridBus tools include:

- Nimrod-G resource broker/scheduler for parameterised application [B02a, BAG00]
- GridSim simulation toolkit for testing Grid scheduling algorithms [B02a]
- Libra scheduler for Portable Batch System (section 2.2.5)
- GridBus resource broker and scheduler for generic Grid applications (no articles have been published yet)
- Grid Market Directory [YVB02, B02a]
- Meta Search Engine for Web services [DB02]
- G-Monitor Web-based Grid portal [PB03]
- GridScape tool for creating dynamic and interactive Grid portals [GB03]
- Peer-to-Peer Compute Power Market [BV01]
- Visual Parametric Modeller for distributed application composition [BMBB02]
- Alchemi: A .NET-based Windows Desktop Grid System [LBRV04]

- GridBank for resource usage accounting and payment (chapter 4) [BB03]

Grid Resource Brokers and schedulers such as Nimrod-G use GridBank to perform financial transactions that keep track of consumed resources. Transactions also represent the resource access authorisation audit for accountability purposes. The continuing trend towards service-oriented Grid and P2P computing [B02a, BV01, HLMS03] ensures usability and portability of accounting systems and promotes global e-commerce.

## 2.6 Payment Systems

There is a wide variety of electronic payment systems in the literature; NetCash [MN93], NetCheque [MN95], Paypal [PP03], CyberCash, CyberCoin [W97], *i*KP [BGHH00, W97], Millicent [GMAG95, W97], Agora [GS96], PayWord, MicroMint [RS96, W97], NetPay [DL99], MPTP [H03], PMTP [PO95, P00], VarietyCash [BGJY98] and others [PJ97, W97]. These systems offer different payment mechanisms and, instead of surveying all of them, we outline three basic methods by which funds are transferred from a client to a merchant: digital cash, micropayment, and digital cheque.

### 2.6.1 Electronic Cash

Similar to actual cash, which has the main advantage of being anonymous, electronic cash systems [MN93, W97] provide anonymous money transfers that do not require accounts. The degree of anonymity varies from system to system: some systems are highly anonymous (e.g. [MN93]) while in others it is possible to determine the payor if the payee and the bank conspire (e.g. [PP03]).

A digital coin is an electronic document stating the issuing bank, the value, the serial number, and the expiry date. The expiry date is an extra safeguard against brute force attacks and should be directly proportional to the average time it takes to break a particular key length. A coin is digitally signed by the bank. Double-spending is a major issue in electronic cash systems since a digital copy can be easily replicated and be spent multiple times. A note presented to a merchant for payment must be verified "on-line" with the issuing bank. "On-line" verification means that there is extra real-time communication between the vendor and the bank. The bank uses the serial number to search its database, and if the number is present, then the coin is validated and the serial number is deleted from the database to prevent double-spending.

A generic algorithm for digital cash using Public Key Infrastructure assumes that all participants are issued standard X.509v3 digital certificates [FFW98]. The cash is generated in the following way:

A client performs mutual authentication with the bank and requests cash for the specified amount.

The bank issues a coin, which specifies the serial number, the expiry date, and the amount of the ledger. The serial number is entered into the database of valid notes, which is used to determine whether the coin has been already spent, thus preventing double-spending. The bank digitally signs the coin as follows [FFW98]:

The message digest of a coin is calculated by passing the whole coin into a hash function (e.g. MD5 [R92]).

Using the private key, the bank encrypts the message digest using a public key cryptography library (e.g. RSA BSAFE [RSA03]) to produce the bank's digital signature.

The resulting signature is appended to the coin.

The bank forwards the cash (coin + signature) to the client.

The merchant receives the cash from the client in return for the service and verifies its authenticity as follows:

Using bank's public key from the bank's certificate the merchant decrypts the appended signature, which produces the original message digest signed by the bank.

The message digest of the coin is calculated by passing it into the hash function.

The original message digest is compared to the one calculated in the previous step.

If they match, then the note is authentic, otherwise the note is counterfeit.

If the note is authentic, the merchant contacts the bank (mutually authenticating each other) and ensures that it has not been double-spent.

At this stage the merchant has two options:

Exchange the note for another one, or

Deposit the cash into the merchant's account.

All communication channels (between the client, the bank and the merchant) in this protocol must be authenticated and secure. Since the cash does not state the identity of either the client, or the merchant, anyone who can obtain a clear text copy of the cash can then spend it. Therefore the cash must be encrypted when passed over a public network. Once the two entities are authenticated, a Public Key Infrastructure can be used to exchange a secret key only valid for the session (for the duration of communication). Secret key cryptographic algorithms are much faster than public key encryption, and standard cryptographic libraries can be used for this purpose, such as Secure Socket Layer/Transport Layer Security (SSL/TLS) [SSLP96].

Scalability of digital cash systems is achieved by the establishment of multiple currency servers such a particular group of clients uses one server only. Coins issued by one currency server can be redeemed at another server, which must verify coin authenticity with the original server that issued the coin to prevent double-spending. Both, public key and secret key cryptographic techniques are employed in these systems to provide security [MN93].

Electronic cash schemes can be used in establishing Grid commerce. However, for accounting and security purposes, it is vital to identify the client. Therefore it is crucial that the cash scheme used should be capable of identifying the payor in case of a security breach.

## 2.6.2 Micropayments

Unlike electronic cash and cheques, micropayments [BIK01, MR02] have no analogy in the real world. The micropayment concept was formulated to overcome the problem of paying small amounts (fractions of a cent) when, for example, browsing a Web site, and the payment is required per Web page requested. Also, the scheme has been employed as an authorisation mechanism for frequent repetitive authentication in some systems. A hash chain consisting of individual hash tokens is constructed and each invoked system operation requires the next hash token from the hash chain in order for the call to proceed [RS96].

Micropayments are light-weight protocols that do not require as much computational power as needed to verify a single digitally signed cash or cheque note. However, micropayment's efficiency comes at the cost of reduced security. It takes much less computational effort to forge a hash token than to forge a digital signature [RS96]. In practice, security of micropayments depends on the number of bits used for each hash token relative to the average time it takes to calculate next token from the previous one using brute-force attack.

The generation of micropayment tokens entails the use of a secure one-way hash function such as MD5 [R92] to construct a hash chain. Such a function produces a result that is easy to calculate but is extremely difficult to obtain the inverse of the function; given an output bit string it is computationally infeasible to produce the input string.

The algorithm for generating a hash chain is as follows.

The number of *paywords* (hash tokens), **n**, is chosen. The n-th payword, $\mathbf{w_n}$, is randomly constructed.

The chain of paywords is calculated using **h** (the secure one-way function), by applying the function to each resulting payword in decreasing order, i.e. $\mathbf{w_{i-1} = h(w_i)}$ for i = n, n-1, … , 1. Neither $\mathbf{w_0}$ nor $\mathbf{w_n}$ are used as paywords. $\mathbf{w_0}$ is signed by the payor with the private key.

The micropayment is initiated by sending the payee $\mathbf{w_0}$.

When the next payment is required, the payor sends a payword, $\mathbf{w_i}$, which is hashed, using **h**, i times and the result is then compared to the signed $\mathbf{w_0}$.

The public key decryption of $\mathbf{w_0}$ signature is performed only once – during the initialisation. All subsequent payments only require hashing, which is approximately 10,000 times faster than verification of a digital signature [MR02].

The payor, or the issuer of micropayments, can be the bank or the client. A client generating its own hash chain only needs the bank to sign the root ($w_0$) hash value. The hash function is known by all parties and any hash token ($w_i$) can be verified by hashing the token (i times) to produce the root hash ($w_0$) and compare it with the signed one. It is argued that micropayments are more secure when the bank generates hash chains [RS96].

A micropayment protocol is as follows:

A client requests a hash chain from the bank by specifying the total amount and the number of tokens, thus assigning a particular value to each token. The merchant's identity must also be specified because otherwise the merchant must verify with the bank that the hash chain has not been double-spent at another service provider.

The bank generates a hash chain as described in the algorithm above, constructs an electronic document (the contract) stating identities of the client and the merchant, the amount assigned to each hash token, and the root hash ($w_0$) used to verify valid paywords (note that the total amount or the number of hashes is not stipulated for privacy and security reasons). The message digest (which is a hash) of the document is obtained by passing the contract into a hash function and the message digest is digitally signed with the bank's private key.

The client receives and forwards the contract to the merchant, who verifies the validity of the root payword ($w_0$) by verifying the bank's signature on the contract.

When the next payment is required, the client sends hash tokens in decreasing order (i.e. starting from $w_{n-1}$ and ending with $w_1$) from the chain, and the merchant verifies it by hashing it i times and comparing it to the root hash ($w_0$) in the contract.

At the end of a transaction the merchant redeems the micropayment by sending to the bank only the last hash token received from the client since the other tokens can be derived by hashing the last token until the root hash is produced.

Although micropayments remove the burden of computationally intensive public cryptographic processing from the banking and trading (service provider) servers, the scheme has been criticised for its relatively low security as well as other aspects such as inadequate anonymity [O00]. Since computation of hash tokens is a lot faster than generation of PKI signatures, it is also computationally easier to break the scheme using a brute force attack. Given a hash token in the chain, forging the next hash token involves

Constructing a bit string containing 0 in all bit positions,

Sequentially incrementing bits (i.e. performing bit-wise addition by adding 1 on each iteration),

Passing each string into the hash function until the result matches the previous hash token.

This process would be a lot faster than a brute force attack on PKI signatures. Certainly the security of a micropayment scheme depends on how many bits are used for each hash token. Increasing the number of bits decreases the probability of a successful attack exponentially.

### 2.6.3 Electronic Cheque

In the real world, bank cheques are an acceptable means of trade, but are seldom used by the general public for their relatively high transaction fees. The digital version, on the other hand, has the advantage of reduced banking fees since most of the processing involved in a transaction is computer automated instead of being performed by human bankers.

An electronic cheque is a digitally signed document stating the payor, the payee, the amount and the details of the bank. Unlike electronic cash, digital cheques do not have to be verified in real-time with the bank. A cheque can only be spent at the merchant whose identity appears on the cheque, therefore double-spending is prevented if the same cheque is presented more than once. Alternatively, the merchant can retain a history of cheques and prevent double-spending altogether. Cheques collected by service providers throughout a day and can be redeemed at the bank in batches at night (or off-peak periods).

A PKI-based generic cheque algorithm is as follows:

A client requests a cheque from the bank by specifying the drawer (itself), the recipient (the merchant), and the amount.

The bank issues the cheque specifying the client's and merchant's identities, the amount and the expiry date. The bank digitally signs the cheque with the bank's private key as described in section 2.6.1.

The client receives the cheque from the bank and forwards it to the merchant specified on the cheque for payment.

The merchant authenticates the cheque with the bank's public key from the bank's certificate as described in section 2.6.1.

The merchant redeems the cheque at the bank; cheques can be accumulated for a period of time and redeemed in batches – the banks can allocate time slots to each merchant for the cheque batch submissions if extra efficiency is desired.

Cheques suit Grid economies very well because the issuing bank need not be contacted ("off-line" transactions), and a cheque can be passed in clear text over Internet. A copy of a cheque is useless to an outsider because a cheque states client's and merchant's identities. An attacker has to forge the client's private key, which is computationally infeasible with a brute force attack. Cheques provide the highest payment reliability and assurance to both the service consumer and the service provider.

## 2.7 Related Works: Accounting Systems

Grid accounting is an emerging field in Grid computing that promises to cover a wide range of issues such as security, audit, accountability, authorisation and even scheduling. Accounting will be crucial for deployment of Grid applications, supporting payment or barter for the use of computing resources and providing incentives to the owners to make idle capacity available to others. A number of Grid and Peer-to-Peer (P2P) accounting systems have been proposed and we examine a few of them in the following sections to outline their main features.

### 2.7.1 SNUPI: A Grid accounting and performance system employing portal services and RDBMS back-end

The System, Network, Usage and Performance Interface (SNUPI) is an open source solution for resource utilisation reporting for heterogenous computer systems, including Linux clusters, developed by the NPACI/SDSC (National Partnership for Advanced Computing Infrastructure at San Diego Supercomputer Centre) [HBY01]. It provides tools for data collection, recommended Relational Database Management System (RDBMS) schema design, and Perl-DBI scripts suitable for portal services to deliver reports at the system, user, and job level across the enterprise. It can be classified as an accounting system at the Grid Fabric layer.

SNUPI considers production services such as performance monitoring and resource utilisation on various variants of UNIX. The system assumes that one or more of the following are provided by the operating system or the batch system:

Batch system accounting,  
System activity reporting,  
Process accounting,  
Project accounting (if available).

Using SNUPI's scripts the accounting data is collected from the operating system, validated and stored in a RDBMS database. The recommended schema includes all the necessary accounting attributes, but some of them are rather specific to NPACI requirements. For example, the database stores the NPACI ID number, which is meaningless outside NPACI. The SITE ID number, which is the UNIX user identification number (associated with the login name), is not globally unique. The ACCOUNT attribute is the user's account number for the job and is site-specific. The SU attribute, which is the total charge for the job, does not specify what currency (i.e. local currency only). APP_NAME is the application name, which is a text string with no constraints.

In general, some of the fields in SNUPI RDBMS tables are specific to the site and do not satisfy global accounting requirements. No account of the security in this system is presented. The system cannot be deployed on the World Wide Grid because it is specifically designed for cluster computing and would not be easily extended.

### 2.7.2 A Resource Accounting and Charging System in the Condor Environment

The authors of this system aim to create a resource accounting and charging system for the Grid environment that has been integrated with the Condor workload management system (section 2.2.3). The resulting application has been deployed at the Department of Control Engineering and Information Technology, BUTE University, Hungary [SLS03].

The system focuses on the development of the base level of accounting and charging services and does not provide economic services such as payment. It defines data flow, entity-relationship and dynamic behaviour models. The data flow model presents a generic architecture of the data traffic between distributed system components, which are responsible for job creation, metering, accounting, processing accounting queries, pricing, charging, processing charging queries, and billing. The entity-relationship model outlines relational database table schemas and their relationship to each other. The dynamic behaviour model introduces the algorithm elaborating on the data flow and entity-relationship models.

The system has been specifically developed for the Condor workload management system. The entity-relationship model does not elaborate on the accounting attributes currently stored, but rather discusses mediation – a way of reducing the mass of information by filtering relevant resource consumption items. The pricing rules are stored in the database and presumably are manually updated by administrators, implying static pricing. This system is not suitable for the World Wide Grid because it is specific to the Condor system.

### 2.7.3 Token-based Accounting and Distributed Pricing to Introduce Market Mechanisms in a Peer-to-Peer File Sharing

The system presents a token-based accounting mechanism that moderates the *free riding* problem present in Peer-to-Peer (P2P) networks (e.g. KaZaA [KAZA03]) where the majority of users consume the content and never provide any. A typical P2P file-sharing system treats all users the same and users do not have incentives to share their content. As a consequence a few peers offer most of the content. To encourage users to share valuable content this system [HLMS03] introduces a token-based accounting system with a quorum of elected trusted super-peers signing the tokens using a shared private key (with corresponding public key for verification). Apart from a certificate authority it is intended to avoid any central element.

Each peer's account consists of a token chain clearly issued to it. Peers cannot spend (foreign) tokens paid by other peers. *The token aggregation process* exchanges the collected foreign tokens to new ones issued to the peer. Upon receiving a token in exchange for service (file content) the provider peer checks for double-spending. The secure approach avoids double-spending by requiring an additional account on a provider peer that keeps track of last token received. The scalable approach detects double-spending (i.e. after fraud has occurred) by exchanging information about aggregated tokens among super-peers; if a peer presents a token for exchange that was subject of a token aggregation process before, double spending is detected.

Service delivery is guaranteed by providing payment in the following way. Before the service is provided, a token is sent without the client's signature. When the service is delivered and the requestor is satisfied with the quality, the token is signed and sent. Unsigned received tokens can be used as non-refutable evidence of the service request when resolving disputes. The pricing algorithm concurs with standard price negotiation and determination methods as presented in [B02a].

The token-based accounting system eliminates the central point of failure and promotes content sharing in P2P systems. The system is secure and scalable to potentially thousands of participants. However, the collection of accounting data about the whole system is tedious and would involve contacting all peers, some of which might be "off-line". The demand-based and market-based pricing schemes discussed (in [HLMS03]) would suffer from such a deficiency. In order to determine market value of resources and perform load balancing in the most efficient way the accounting information must be readily available and easily accessible.

### 2.7.4 Global Grid Forum Accounting Working Group

The GGF Accounting Working Group has been formed as part of the Global Grid Forum to overcome billing and authorisation limitations present in distributed Grid systems. The goal of the group is to collaborate research and development in the accounting field of Grid computing. Grid Economic Services Architecture (GESA) [GESA03] concentrates on providing the enabling infrastructure by defining additional service data and ports (interfaces) compliant with the Open Grid Services Architecture. OGSA builds on top of Web services; therefore GESA's design is based on OGSA's standards, namely Java and XML, and describes the architecture in terms of concepts such as inheritance.

The GRid Architecture for Computational Economy concentrates on pricing mechanisms independent of middleware implementation, whereas GESA provides an extension of the OGSA's Grid service definition. A third-party trusted Grid banking service and a resource usage service are considered as separate components and are not defined. Service Data Elements, which describe potential chargeable resource items, provided by the "Chargable Grid Service" abstraction are categorised but are not defined syntactically. The work is in progress and promises to improve future releases of the Globus Toolkit by providing integrated accounting services [GESA03].

## 2.8 Summary

The World Wide Grid computing paradigm is composed of four layers: Grid Fabric, Grid Middleware, Grid Application Development Tools, and Grid Applications. The separation of concepts of each layer allows independent researchers to develop their own components without affecting components from other layers. A number of other Grid systems (such as NetSolve [CD97] and Legion [CKKG99]) are monolithic and assume a vertical integration of components, which is often too inflexible. The success of the Globus Toolkit, which is an implementation of the Open Grid Services Architecture, is partially due to integration of Grid Fabric Resource Management Systems such as PBS, LSF, Condor, and SGE, which are more general and used more widely than the others.

The Globus Toolkit provides resource management middleware that unifies all of the RMSs and presents them as one powerful meta-computing resource. Security in such an environment must scale well to potentially millions of clients and services as well as being able to interface with the security system of a RMS. It achieves local security policy independence by providing the Grid Security Infrastructure based on a Public Key Infrastructure. A Grid accounting service should use the GSI in order to preserve the OGSA’s philosophy of single sign-on: once an entity is authenticated, it acquires access to all authorised Grid services.

Once authenticated, Grid authorisation is performed by individual Grid service providers. The scalability of such an approach to authorisation is poor because when a new user joins the Grid all resources that the user is allowed to access must be notified. The Community Authorisation Service offers a centralised role-based access control. Only the administrator of the CAS server needs to be contacted instead of all resource owners. Another typical authentication problem the CAS solves is the ability to recognise multiple Certificate Authorities such that an entity with the certificate from one CA can access a resource that recognises another CA. Only the CAS needs to recognise both. An authorisation and accounting system should offer similar infrastructure.

Existing payment and accounting systems do not use the GSI. Therefore they cannot be classified as Grid services. It is possible to design a Grid service interface for the systems, but such an approach would result in inefficiencies. Also most existing Grid accounting systems provide services at the Grid Fabric layer, which means that the data they collect is local to the site. There is a need for an integrated accounting and payment infrastructure at the Grid Middleware layer that uses the GSI for authentication. Next chapter presents such architecture.

# CHAPTER 3

# Grid Accounting Services Architecture

## 3.1 Introduction

Grids and Peer-to-Peer (P2P) network computer systems are emerging as a new computing paradigm for Internet-based parallel and distributed application processing [BFPT00, CKKG99, FK97, FK98b, FK99, FKNT02, FKT01, TCFF02]. They enable the sharing and aggregation of heterogenous computing resources distributed geographically as well as across multiple administrative domains and organisations. Traditional resource management systems (e.g. Portable Batch System, Load Sharing Facility, Sun Grid Engine) are system-centric, that is, they optimise for system performance metrics such as throughput and response time. Such an approach does not scale well to systems with potentially millions of compute and data nodes, for example the SETI@Home [SWBC96] project, typical of Grid systems because resource consumers and providers might have different objectives for being part of the Grid and, for example, might not be interested in maximising throughput.

The GRid Architecture for Computational Economy (GRACE) [BAG98, B02a] overcomes this limitation by proposing a user-centric approach to resource scheduling. In such economic settings Grid Service Providers and Grid Service Consumers have different goals, objectives, strategies, and supply-and-demand patterns. The economic approach provides a fair basis in successfully managing decentralisation and heterogeneity present in human economies. In the context of compute power markets it is extremely important to provide accounting and payment mechanisms that allow GSPs to exchange Grid services in return for currency, which can be actual, such as Australian dollars, or virtual, such as Grid dollars [B02a].

This chapter presents an accounting framework named Grid Accounting Services Architecture (GASA) [BB03] that extends GRACE to provide Grid resource access authorisation, accounting and payment protocols needed for successful construction of free market economies of scale on computational Grids. In order to encourage resource owners to contribute their services to the Grid, there is a need for a system that records resource usage and compensates for it in some way.

The main contribution of this chapter is to define accounting requirements and provide a generic architecture and algorithms for resource access authorisation, accounting and payment in the context of computational Grids. The GridBus project undertaken at the University of Melbourne simplifies the management of Grid resources by developing economy-driven concepts and tools for computational and data Grids. An implementation of GASA, named GridBank [BB03, B02a], is part of the project and is being used in the construction of the Grid for the Belle high-energy physics simulation [BADG03].

## 3.2 Computational Accounting and its Applications

Accounting for traditional end-system computing devices such as PCs has not been considered significant as these systems are typically owned and used by one person or organisation. Computational Grids provide distributed and parallel Internet-based application processing, which assume the data and execution hosts are unknown until run-time. During application deployment computational jobs are mapped onto selected resources, which might belong to different administrative domains owned and managed by independent Virtual Organisations.

The need for an accounting infrastructure, given the economic principles are adopted, comes from the answers to the following questions:

- Can resource access authorisation be achieved through payment provision?
- If so, are the resource providers concerned about who is authorised and how much resources they consume?
- If unauthorised or unintentional access occurs, how can security and accountability audits be conducted?
- Is it possible to improve Quality-of-Service provision for applications and reduce the cost by making scheduling decisions based on historical analysis of accounting data?

- Can resource providers improve the utility attained and improve QoS by experimenting with different pricing strategies and judging the best one based on transaction history?
- Can Grid load balancing and determining future hardware and software requirements be achieved by examining the previous usage?
- If an advanced reservation and allocation system is used, how easy is it to determine whether the reservation and the allocation have been utilised?

Accounting is crucial for the deployment of Grid applications, supporting payment or barter for the use of computing resources and providing incentives to the resource owners to enhance the utility of resources and the ability to exchange or share each others resources. Audit associates each service request with the corresponding service consumer, service provider, and the consumed resources. An audit is valuable in determining faults or tracking breaches of security [FK99]. The CAS system (section 2.4.1) mentions the need for accounting in authorisation systems [PWFK02].

A Grid resource broker can evaluate application performance on a particular platform and make future scheduling decisions based on capability of the previously used resources compared with the capacity of resources under consideration. For example, computationally intensive applications would perform better on resources with faster CPUs, whereas data intensive applications perform better on resources with larger and faster secondary and tertiary storage systems and network capacities. New scheduling algorithms would further improve the utility of Grid resources and avoid providers that are reluctant to enforce the promised Quality-of-Service [B02a].

## 3.3 Global Computational Accounting Requirements

Local Resource Management Systems (RMS), such as those surveyed in chapter 2, typically have a local accounting infrastructure to collect resource usage statistics from each execution node involved in serving a local user, and all user expenditures are recorded against a local account. Section 3.5.2.1 explains how global user identities such as digital certificate subjects are temporarily “mapped” to local accounts. The GASA accounting framework assumes that some accounting data is generated by the local RMS systems and is available ideally on request after job submission for monitoring purposes, or at least after execution for billing or accounting. Grid services are primarily concerned with utilising hardware, thus the performance metrics reflect hardware parameters in contemporary computer architectures. Since hardware and software are constantly being improved and new paradigms emerge, a resource usage record must be extensible, that is, the record format can be extended to the new format, while still being compatible with the old format.

The Grid Resource Usage Record (GRUR) must reflect accounting principles involved in any kind of financial transaction. A typical goods or services proof of purchase has the following details:

- Name or identification of the client,
- Name or identification of the merchant,
- Description of goods or services traded, and
- Amounts paid for individual items and the total.

Based on these principles GSC details include:

Email address,
Digital certificate subject name,
Certificate Authority that issued the certificate,
URL of compute or data node, and
Optionally:
Static IP address or IP address of the Internet Service Provider if IP address is dynamic,
Local RMS account name, and
The whole GSC's certificate.

GSP details include:

Email address of resource owner or administrator,
Digital certificate subject name,
URL and IP address of submission node, and
Resource performance metrics, whichever are relevant:

Number and speed of CPU units,
Main memory size,
Secondary storage size,
Tertiary storage size (for massive storage systems),
Network bandwidth of resource network interface (In/Out), and
Other specialised metrics; for example:
Pixels resolution on digital scopes,
Printer metrics (e.g. pages per minute, dots per inch),
Graphics adaptor metrics (e.g. on-board memory size).

Job information includes:

- User and system CPU time,
- Number of bytes of main memory allocated and actually used,
- Number of bytes of secondary storage allocated and actually used,
- Number of bytes of tertiary storage allocated and actually used,
- Number of bytes of raw network traffic (In/Out),
- Other specialised consumption; for example:
    - Number of pages printed,
    - Number of digital images captured.
- Software consumption (if relevant):
    - Application name,
    - Application version,
    - Application type (locally installed, user stage-in, ASP),
    - Application implementation language,
    - Dynamic link libraries used (names, versions),
    - Operating system (name, version).

Service consumption can also be charged for on the per Application Operation basis [YVB02]. For example, a DNA sequencing application might be charged for each DNA sequence computed.

Most of these data are present in local RMS systems, and allow creation of a World Wide Grid accounting infrastructure. Although this information can be provided in a plain text file, with attribute-value pair per line, the data should be structured for the ease of navigation (i.e. querying) and ability to extend the record in a consistent way.

Common hardware performance metrics can be generalised according to the granularity of modern computer hardware architectures:

CPU (user-mode and system-mode) time: seconds, milliseconds
Main Memory: MB*second (MBs)
Secondary Memory: GB*hour (GBh)
Tertiary Storage: TB*day (TBday)
Massive Storage Facility: PB*month (PBmonth)
I/O Devices: MB

The units (MB, GB, TB, PB, second, hour, day) can be adjusted according to the scale of the Grid in such a way that a smaller unit is redundant. The important point here is that in a Grid, not only does the memory size (e.g. MB) matter, but also the time that the memory was used (e.g. an hour) Therefore computational power is defined by the metric that is in terms of processor speed and memory size multiplied by time (e.g. MB per second, GB per hour). By analogy, the electricity Grid has units of power*time (e.g. kWh). The exception to this rule is the Input/Output metric, which only measures the total traffic (e.g. MB). Depending on the bandwidth and latency of an I/O channel, different tariffs will apply.

## 3.4 Grid Accounting Services Architecture (GASA)

A distributed Grid Accounting Services Architecture is shown in figure 3.1. It incorporates GRACE's (GRid Architecture for Computational Economy) components as well as introducing the new concepts. The key components are:

- The Accounting Server,
- The Grid Service Consumer Payment and Administration Module (with Graphical User Interface),
- Grid Service Provider Resource Access Authorisation Module,

- Grid Resource Meter, and
- Grid Service Charge Calculator.

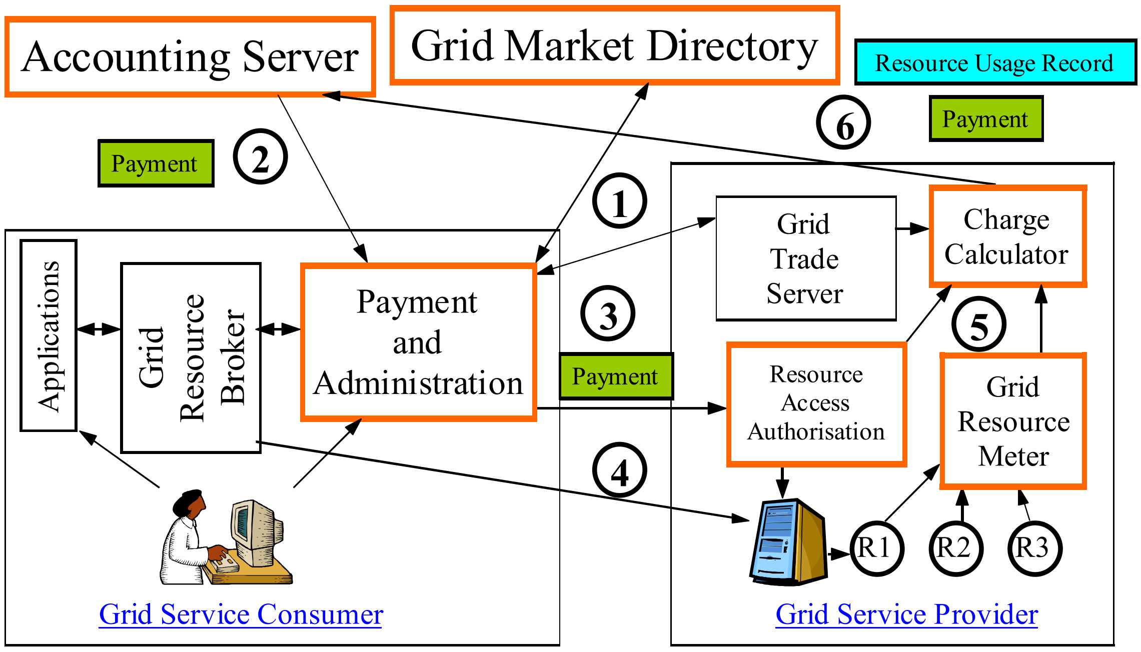


**Figure 3.1: Grid Accounting Services Architecture (GASA)**

Grid Service Consumers and Providers open accounts with the Accounting Server. The user submits the application processing requirements along with the QoS specification (e.g. a deadline and budget) to the Grid Resource Broker (GRB). The GRB interacts with GSP's Grid Trade Service (GTS) or Grid Market Directory (GMD) [YVB02, B02a] to establish the cost of services (i.e. service rates) in step 1, and then selects a suitable GSP. Grid Market Directory simply aggregates information collected from several GTSs. In step 2, the Payment module acts on behalf of the GRB, which acts on behalf of the client, and requests a payment instrument (e.g. GridCheque [BB03]) for the amount it would cost to execute the application. The payment is forwarded to the Resource Access Authorisation module in step 3, which authenticates the payment and allocates a local account for the purposes of process creation and file access (section 3.5.2.1 covers this aspect in more detail). Successful authorisation permits the GRB to submit application job(s) to the resource in step 4. In step 5, the Grid Resource Meter (GRM) collects resource usage data from all resources involved in the computation. The Charge Calculator attains the payment token(s) from the Resource Access Authorisation module and the Resource Usage Record [BB03] from the GRM. The payment instrument and the Resource Usage Record are submitted to the Accounting Server for reconciliation in step 6.

The three distributed key entities in the architecture are the Grid Service Consumer (GSC), the Grid Service Provider (GSP) and the Accounting Server. GSCs and GSPs have their own strategies and goals for being part of the Grid. In an economic Grid environment GSCs strive to minimise the resource cost while obtaining best resource utility. On the other hand, GSPs maximise the cost to obtain greatest possible returns on their hardware and software investments. GSC and GSP fraud can emerge. A GSC can accuse a GSP of failing to provide the service for the supplied payment. A GSP can claim the service was provided, but the payment was never received. Another situation is when a GSP advertises one set of service rates, but at the end of the GSC's computation uses higher tariffs to calculate the charge. Also a GSC and a GSP need to agree upon what happens if an executing job exceeds allocated budget.

The Accounting Server acts as a trusted third party that mediates financial transactions and enforces collection of resource usage data; in order to redeem a payment a GSP must present the usage record. It also acts as an authorisation entity either if payment is considered as credentials for resource access or if interfaced to another authorisation system such as CAS (section 2.4.1). The cental accounting and authorisation server, although representing a central point of failure, greatly reduces the risk of deceit by the Grid participants similar to actual banks. Anonymity provided by some payment systems [MN93, W97] is not preferred in a Grid environment for security and accountability reasons.

The three resource access concepts in the context of computational economies, namely authorisation, payment and accounting, are tied together by the need for a trusted third party. Authorisation by itself can provide sophisticated access control mechanisms and record every resource access event. An accounting system alone can record resource usage. Payment systems can provide some particular payment mechanism (cheque, coins, micro-payments) and generate transaction information. If the three systems are introduced separately to the Wide World Grid, they would produce a lot of redundant data since each system stores at the bare minimum the identification of both parties involved in the transaction, as well as time, place (physical or virtual), and in the case of accounting and payment – the bill. Another potential problem is creating three different identifications for each Grid participant. GASA eliminates the need for the three systems and provides a distributed accounting data model with unique Grid-wide identities.

## 3.4.1 Accounting Server Architecture

The Accounting Server architecture is shown in figure 3.2. It is divided into three layers:

1. Security Layer,
2. Protocol Layer, and
3. Database Layer.

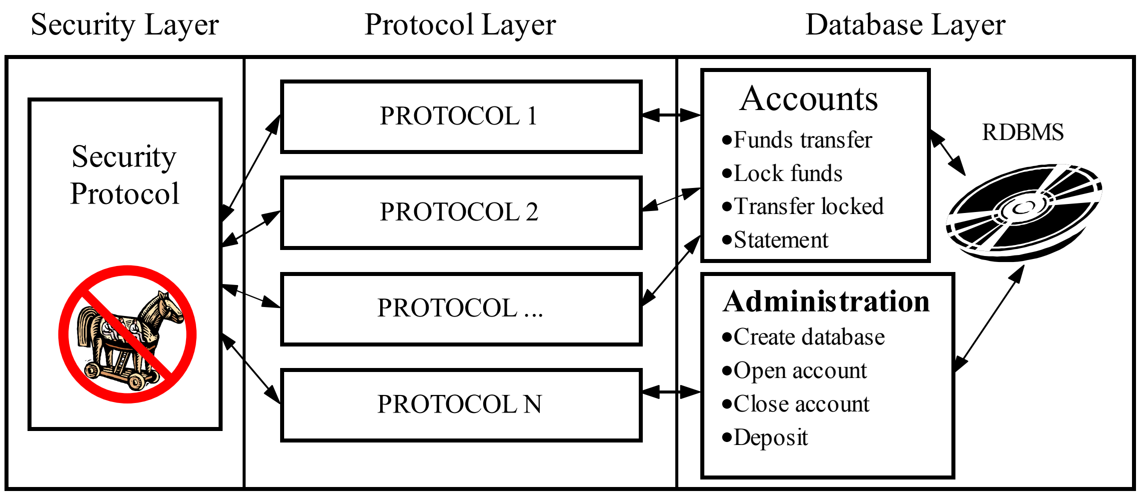


**Figure 3.2: GASA Server Architecture**

The Security layer is responsible for authenticating clients and enforcing data privacy and integrity of transactions traversing public networks. Authenticity of the GSC and the GSP ensures that both parties do not exhibit malicious behaviour such as masquerading as another identity. It should also provide guards against other well-known network attacks. Denial-Of-Service is one common attack that can generate millions of phoney messages, thus flooding the server request queue. Replaying messages, man-in-the-middle, message tampering, eavesdropping and other forms of vandalism must be isolated and must not affect system behaviour.

The (Accounting) Protocol layer builds on top of the Security layer, assuming that any messages passed to the protocols are authentic and have not been tampered with during public network transmission. The purpose of the Protocol layer is to incorporate different payment and accounting protocols without the need to alter the Security or the Database layers. Identity of the authenticated client must be passed to each protocol, and the protocol determines whether the client is authorised to perform the operation by searching for the client's identity in the database. Once an entity is affiliated with an account, protocols perform transactions via the Database layer.

The Database layer provides the interface to the database of accounts and administrators. Two modules abstract the database from protocol functionality. The ACCOUNTS module manipulates client accounts. The "Funds transfer" operation provides the means to withdraw money from one account and immediately deposit the amount into another account. Each account has available and "lock" balances. The "lock" balance is a repository of funds reserved for deferred transactions. Whereas "funds transfer" operates on available balance, the "lock funds" operation debits the available balance and creates a deferred payment record detailing the GSP, the amount, and when it expires. If the specified GSP has not claimed the amount within the valid period through "transfer from locked" operation, the transaction is assumed to fail and the amount is transferred back to the available balance. The "Lock funds" and "transfer from locked" operations are useful for many payment protocols such as digital cheques and micro-payments.

The ACCOUNTS module also provides basic account audit inquires such as account balances and statements. The ADMINISTRATION module manages user accounts as well as other administrators. It provides functions to create an empty database, create and delete other administrators, open and close accounts, cancel transactions, deposit and withdraw money from the accounts. The deposit and withdrawal operations are to be performed by authorised administrators only or can be interfaced with other electronic payment systems such as those presented in section 2.6

## 3.4.2 Grid Service Provider Architecture

The Grid Service Provider architecture is shown in figure 3.3. It introduces the new components, namely, the Grid Resource Meter, the Grid Resource Access Authorisation/Template Accounts and the Charge Calculator.

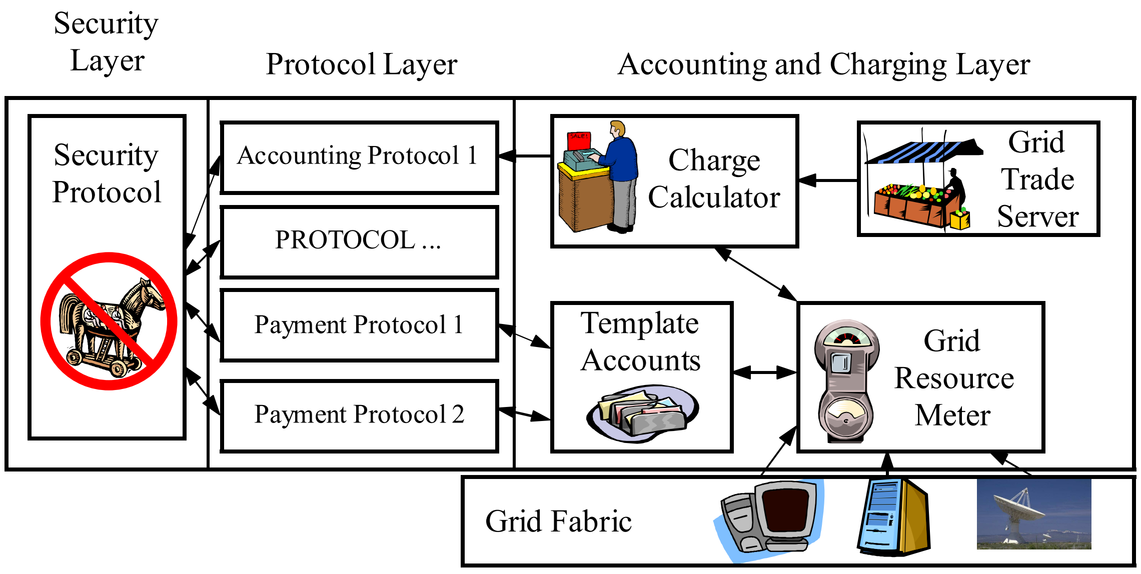


**Figure 3.3: Grid Service Provider Architecture**

The Security and Protocol layers play identical role as in the Server architecture. The Accounting and Charging layer is responsible for the allocation of local resource accounts, monitoring resource consumption and calculating the total charge after the service has been consumed.

Payment issued by the Accounting Server represents the resource access authorisation and is received via the Security and Protocol layers. The Template Accounts module allocates a local account on the resource as the means of resource access. This concept is covered in section 3.5.2.1. The Grid Resource Meter obtains the budget from the Protocol layer and the service rates from the Grid Trade Service for the particular GSC. Different rates can be offered to different GSCs according to pricing strategies [B02a].

The service rates are based on time and specify how much currency to charge per unit of time (e.g. G$ per second [B02a]). The service could charge a flat fee or no fee at all, in which case the Grid Resource Meter merely collects the resource usage. Once a job is submitted for execution, the GRM monitors resource consumption, translates it into budget consumption via the service rates, and ensures that the budget is not overspent. When the job execution completes, Grid Resource Meter de-allocates the local account (section 3.5.2.1) and forwards the Resource Usage Record to the Charge Calculator, which computes the total charge and adds it to the record. It is a requirement of payment protocols to present the Resource Usage Record along with the payment token in order to redeem the charge at the Accounting Server. The requirement ensures collection of up-to-date resource usage information and forces GSPs to produce such audit data.

#### 3.4.2.1 Grid Template Accounts

Every Grid resource (PC, cluster, supercomputer, specialised device such as electronic microscope) has a local system account for the purposes of process creation and local file system access [FK97, FK99, FK98b, HKFG00]. The resource administrator creates this account in the local Operating System or RMS (such as cluster systems). Only users who have an account are authorised to use the resource. The implication of this traditional system management is that each resource owner has to create an account for every authorised user, and each user has to contact the administrator of every resource. Clearly, this approach to local account management is not scalable in a Grid environment.

Instead of creating one account for every user (there can be millions of them), a GSP maintains a small pool of template accounts [HA01]. A template account is an ordinary local system account, which is not permanently associated with any global (Grid) user. When a GSC contacts a GSP to execute an application, the GSP dynamically assigns one from the pool of available (free) template accounts. The GSC's identity is temporarily associated with the local account. Once the application completes execution, the template account is dissociated from the GSC and is returned to the pool of available accounts.

The GASA Server or any other Grid resource authorisation system such as the Community Authorisation Service manages the coarse-grained access control to the resources by stipulating which GSCs are allowed to access which GSPs, while each GSP retains the fine-grained control over its resource by specifying local account permissions. The number of template accounts that a GSP needs to maintain is estimated based on the peak resource access [HA01]. The Grid Template Accounts component of GASA records and notifies the resource owner when all template accounts are exhausted, in which case the resource administrator should create more template accounts. The maximum number of accounts depends on the system capacity, i.e. aggregated CPU speed, main memory (RAM, virtual) and secondary storage sizes, network bandwidth and latency. The slower the system, the fewer users should be allowed to access the resource simultaneously.

### 3.4.3 Grid Service Consumer Architecture

The Grid Service Consumer architecture is shown in figure 3.4.

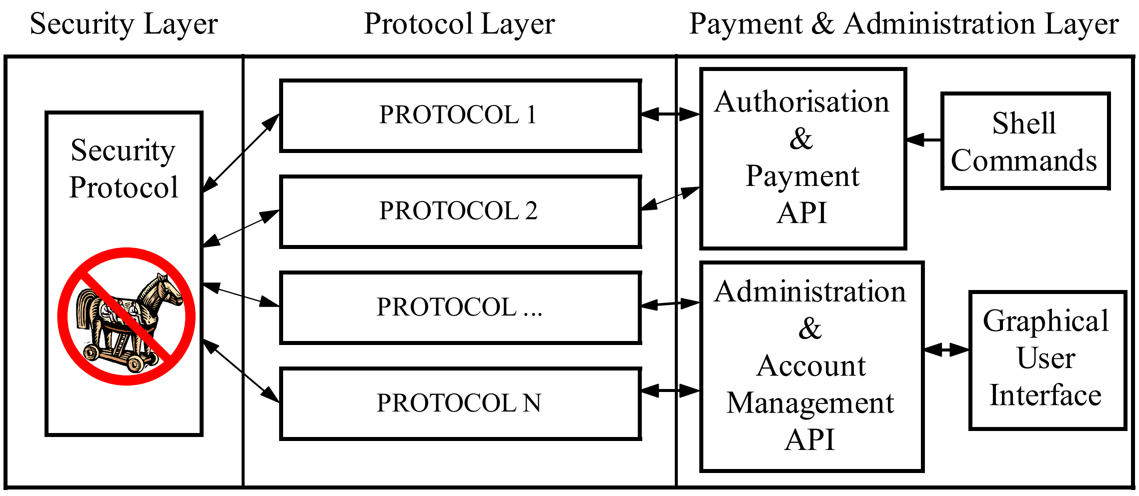


**Figure 3.4: Grid Resource Consumer Architecture**

The Security and Protocol layers have equivalent functionality as in the Accounting Server and GSP counter parts. The Payment (Authorisation) and Administration layer provides APIs (Application Programming Interfaces) to perform authorisation and payment operations when submitting jobs for execution on remote resources and for managing the Accounting Server's administrator and user accounts.

The Authorisation and Payment API offers three different payment methodologies:

- Pre-pay,
- Pay-as-you-go, and
- Pay-after.

These payment tactics allow GSCs and GSPs to manage the risk associated with financial transactions by selecting the appropriate method. The Pre-pay operation can be configured to merely perform resource access authorisation as described in section 3.5.2.1 if a Grid service is free of charge for some particular GSC.  The Pre-pay option transfers the specified amount from the GSC's account immediately into the GSP's account and is suitable when the GSP has a minimum of trust in the GSC and the GSC has a maximum of trust in the GSP for delivering service, or when the GSP has a flat fee charge. The Pay-as-you-go issues micro-payment tokens, which are periodically released by the GSC to the GSP according to resource consumption. This choice is appropriate when the service total cost is unknown beforehand and both the GSC and the GSP have average trust for each other. The money is transferred when GSP presents the Resource Usage Record and the last micro-payment token received. The Pay-after models the credit card or the cheque payment mechanism from actual banking. As in Pay-as-you-go, the requested amount is reserved, and is transferred from the GSC's account to the GSP's account when the computation completes, provided the GSP presents the Resource Usage Record.

The Administration and Account Management API (figure 3.4) provides the means to remotely perform user accounting and administration operations. The Graphical User Interface (GUI) allows users to inquire about their account details such as the balance, as well as request account statements and perform transfers. Administrators use the GUI to manage other administrators and deposit and withdraw funds from user accounts as described in section 3.5.1. The GUI should have an ability to be integrated into a Web browser such that all management functions can be invoked from the Web.

#### 3.4.3.1 Payment Policy

Regardless of the payment method used, if the service charge depends on execution time, the accounting architecture is faced with the problem of handling excessive resource consumption when a computation exhausts the allocated budget. We extended the architecture to grant the following scenarios. In the case that the allocated budget is overspent:

- Cancel the job immediately,
- Suspend the job and ask for more payment,
- Wait for a certain period for payment and then cancel, or
- Keep executing for a certain period or until job terminates and inquire payment.

Before the job is submitted for execution, the GSC and the GSP must agree on which policy will apply. This is necessary to minimise the probability of a dispute in case of budget exhaustion.

## 3.5 Grid Accounting Services Architecture Protocols

Given the generic architecture presented in section 3.4 the following protocols are defined to introduce payment management. The three protocols, GridCash, GridHash, GridCheque, correspond to the three payment methodologies (pre-pay, pay-as-you-go, pay-after) [BB03, B02a] essential for the provision of alternative payment methods.

### 3.5.1 GridCash

The GridCash protocol is based on the algorithm presented in section 2.6.1. It corresponds to the pre-pay methodology. The purpose of this protocol is to provide a payment instrument that does not specify the identity of the GSP. The GridCash can be spent at any provider, but the provider must verify in real-time the validity of the cash with the Accounting Server to prevent double-spending. The identity of the GSC is included in the GridCash since the Resource Access Authorisation/Template Accounts module (GSP architecture) requires it in order to authorise resource access (section 3.5.2.1). This protocol is useful when a GSC has a large number of jobs, service providers are unknown, and the GSC wants to avoid contacting the Accounting Server every time a payment is required.

The numbers in figure 3.5 correspond to the corresponding steps of the algorithm.

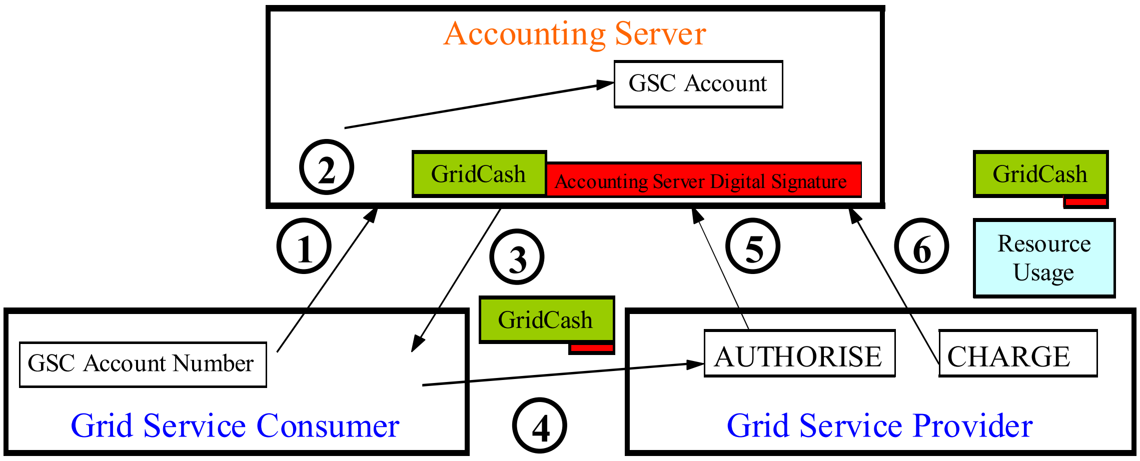


**Figure 3.5: GridCash Protocol**

- The GSC initiates a transaction by requesting cash. The GSC specifies the account number from which to withdraw the funds, and the amount to be withdrawn. The GSP is presumed unknown and is not specified. The Grid Security Architecture (section 2.3.1) is employed to mutually authenticate the GSC and the Accounting Server.
- The Accounting Server ensures there are sufficient funds in the GSC's account, and reserves the amount by transferring it from the available balance into the "lock" balance (section 3.5.1). Upon success the server issues an electronic cash note stating identities of the GSC and the issuer (itself), the amount, the expiry date, and signs it with the Accounting Server's private key to produce the digital signature.
- The Accounting Server forwards GridCash to the GSC.
- The GSC selects suitable GSP and presents the cash.
- The GSP contacts the Accounting Server to ensure that the cash has not been double-spent and allocates a template account.
- When the application processing is completed, the GSP redeems the GridCash by sending it along with the resource usage to the Accounting Server. The GSP must ensure during job execution that the cash has not expired.

Since the GridCash protocol does not state the GSP, anyone who obtains a copy of a GridCash note can unlawfully disburse it. Therefore it is utterly important to secure all communication channels between the Accounting Server, the GSC and the GSP. Steps 1 and 3 (figure 3.5) must mutually authenticate the Accounting Server and the GSC. The GSC and the GSP must mutually authenticate each other in step 4. In steps 5 and 6 - the

Accounting Server and the GSP. Once authenticated, the GridCash must be strongly encrypted to avoid eavesdropping attacks that would result in theft.

### 3.5.2 GridHash

The GridHash protocol's name reflects the nature of the payment; the hash chain is the mechanism used in this scheme. The protocol is based on the micropayment algorithm presented in section 2.6.2. It corresponds to the pay-as-you-go payment methodology and provides the means to pay in increments. This protocol is practical when the application execution time is unknown (e.g. never executed before, non-deterministic run-time). Figure 3.6 illustrates the protocol.

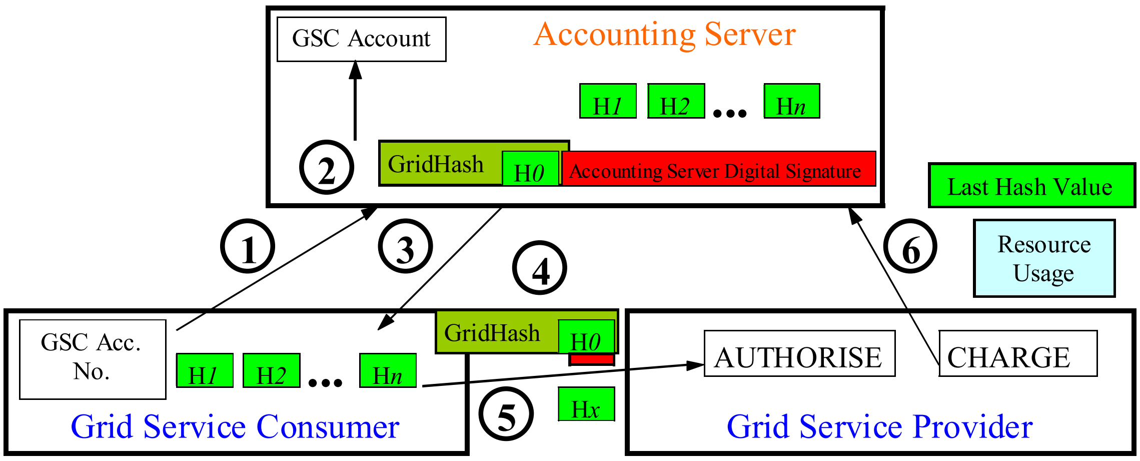


**Figure 3.6: GridHash Protocol**

The numbers in figure 3.6 correspond to the following steps of the algorithm:

The GSC requests the hash chain by specifying the account numbers of the GSC and the GSP, the total amount of the whole chain, and the number of hashes (implying the amount assigned to each hash token).

The Accounting Server "locks" the funds in the GSC's account, generates the hash chain using a secure one-way hash function such as MD5 (as described in section 2.6.3), digitally signs the contract containing the initial hash token ($h_0$), identities of the GSC, the GSP, the issuer, the expiry date, and the value assigned to each token. The total (or the number of tokens) is not stated for privacy reasons.

The Accounting Server forwards the contract and the hash chain to the GSC.

The GSP receives the contract from the GSC, validates the Accounting Server's signature, and authorises the resource access.

The GSC periodically dispenses payments by releasing the next hash token ($h_i$) to the GSP.

The GSP redeems the micropayments by transferring the last hash token received from the GSC together with the resource usage. The GSP must redeem the last hash token before the expiry date.

The GSC and the GSP are both identified in the GridHash contract. The Accounting Server and the GSC must mutually authenticate each other in step 1 (figure 3.6). The GridHash contract and the hash chain passed in step 3 must be encrypted to avoid theft; the GSP can eavesdrop on the transmission and redeem the whole chain illegally. In step 4 the contract is in clear text since it is useless to an attacker because it stipulates identities of the GSC and the GSP. The hash tokens released in step 5 are in clear text, but the CSC and the GSP must mutually authenticate each other. The GSP must ensure that any hash tokens received were from the GSC stipulated in the contract; if a hash token is not from that GSC, then it is assumed unauthentic and is ignored. Optionally, the GSP can report the attacker to the appropriate authorities (e.g. the Accounting Server). In step 6 the Accounting Server and the GSP must mutually authenticate each other for non-repudiation of the Resource Usage Record, and also because the resource usage must be encrypted for privacy reasons.

## 3.5.3 GridCheque

The GridCheque protocol is based on the algorithm described in section 2.6.3. It offers the pay-after payment methodology and by far the most secure method of all since both parties are identified and mutually authenticated. Therefore a cheque is useless to an attacker. This protocol is most useful when there is minimal trust between the service consumer and provider. Figure 3.7 depicts the protocol.

The numbers in figure 3.7 correspond to the corresponding steps of the algorithm.

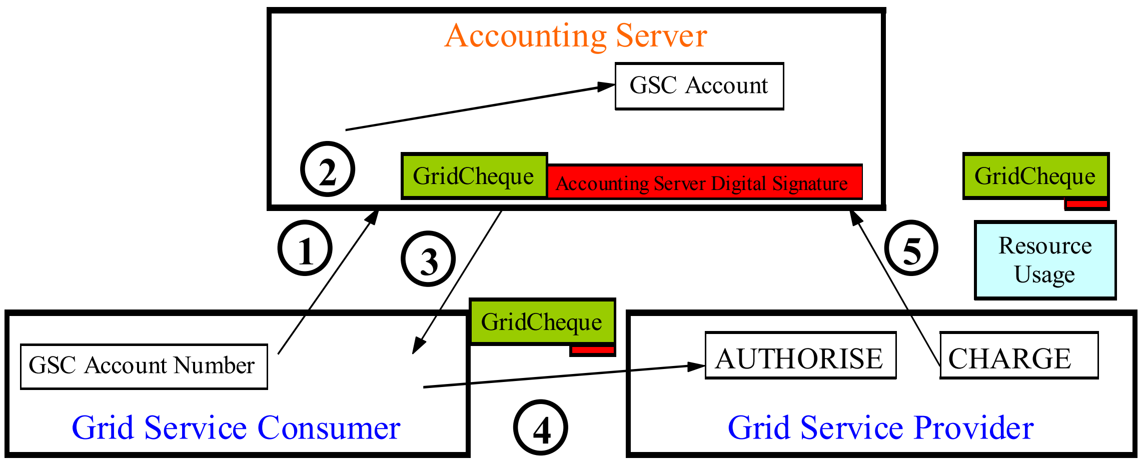


**Figure 3.7: GridCheque Protocol**

The GSC requests a cheque by specifying the GSP and the amount

The Accounting Server reserves the amount in the GSC’s account and issues an electronic document stating identities of the GSC, the GSP, the issuer, the amount and the expiry date. It signs the cheque with the private key.

The Accounting Server passes the GridCheque to the GSC.

The GSC forwards the cheque to the GSP who authorises the resource access by validating the GridCheque’s signature by using the public key from the Accounting Server’s certificate (as described in section 2.6.3).

The GSP redeems the payment by sending the GridCheque and the resource usage record to the Accounting Server. The GridCheque must be current, that is, not expired.

A GridCheque is meaningless to an attacker because identities of the GSC and the GSP are stated. Therefore a cheque is transmitted in clear text in steps 3 and 4 (figure 3.7). An attacker cannot gain access to the resource because it requires authentication as the GSC’s identity stated on the cheque. As in the GridCash and GridHash protocols, in step 1 the Accounting Server and the GSC must mutually authenticate each other, and step 5 must ensure non-repudiation and prevent disclosure of private information by encrypting the Resource Usage Record.

## 3.5.4 Multi-Party Payments

The next scenario considered is when a GSC requires services of several GSPs. The previously defined protocols are slightly modified to provide multi-party payments. Issuing individual payments for each selected GSP can be a performance issue. In such a case, the Accounting Server produces one document listing identities of all requested

GSPs and the corresponding amounts assigned to each GSP. A GSC only needs to contact the Accounting Server once and multicast a copy of the document to all GSPs.

Such an optimisation is elementarily applied to the GridCheque protocol. The GridCash does not state the GSP's identity and separate notes must be issued for different GSPs. In the case of micropayments a multi-party micropayment scheme can be employed, such as the one designed by Peirce [P00].

A transaction is initiated by a client generating a hash chain consisting of paywords $P_0$, $P_1$, $P_2$, … , $P_N$ by applying a secure one-way hash function to the root token $P_N$ (as described in section 2.6.2). The client forwards the $P_0$ token, the length of the chain (N), the desired total value of the chain, a nominated specific service provider called *enforcer*, and details of the micropayment, which is the GSC's account number in the context of Grid Accounting Services Architecture. All details are encrypted with the broker's public key. The chain has no monetary value until *committed* by the broker.

The broker commits to the hash chain by digitally signing the *commitment* consisting of the $P_0$ token, length (number of tokens), the value of a single token, which is fixed later, the identity of the enforcer, and the expiry date. The user selects suitable service providers and informs the broker of their identities as well as the Quality-of-Service requirements. The broker then signs a *pricing contract* consisting of the transaction identifier, the list of service providers, the charging mechanism and individual service rates (tariffs) for each service provider, the broker commitment, the value assigned to each hash token, an endorsement chain commitment created and signed by the enforcer, and the list of redeeming brokers. The value assigned to each token is calculated based on each service provider's tariff rate and the chosen payment period (e.g. every minute) such that the value is the addition of individual tariffs of each service provider. The enforcer generates the second hash chain in order to prevent double-spending (explained below).

Once the contract is composed, every service provider (SP) mentioned in the contract must verify details pertaining to the SP and sign it, therefore agreeing to the corresponding tariff and the QoS assurance. Once the contract is signed by all implicated service providers, the client can initiate micropayments. Figure 3.8 illustrates the protocol.

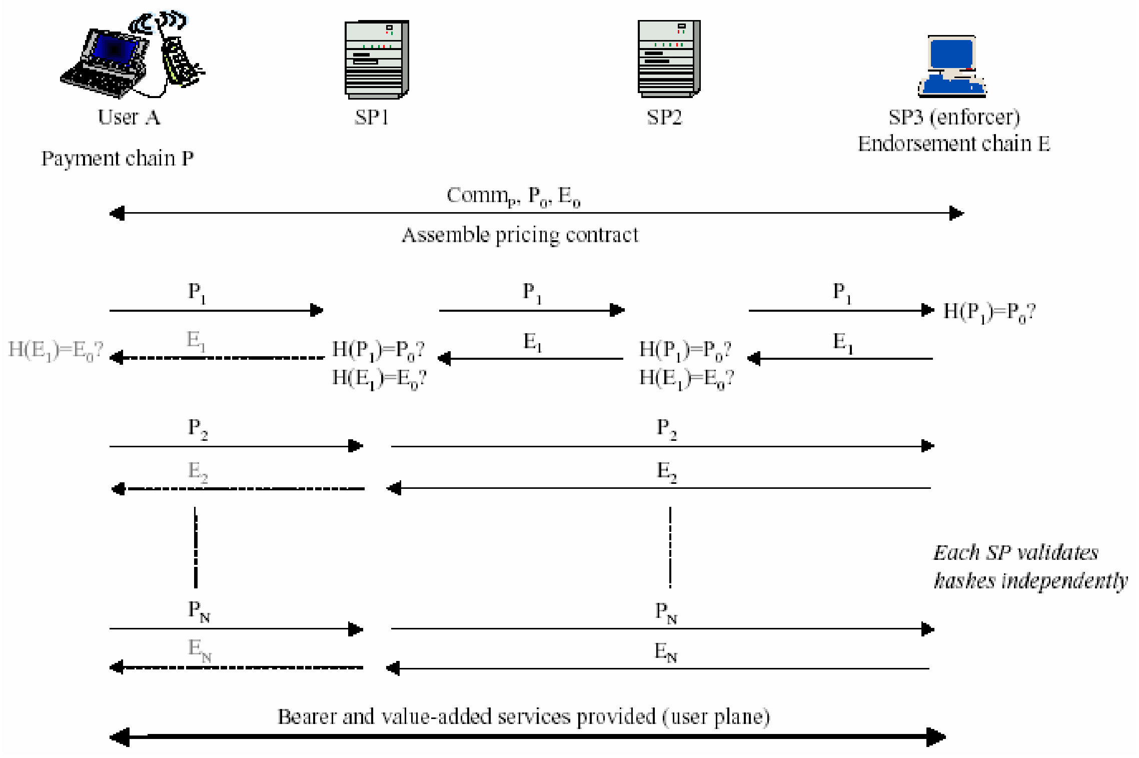


**Figure 3.8: Multi-Party Micropayment Scheme. Courtesy of M. Peirce [P00].**

Each payment period the user releases a hash token, $P_x$, from its own chain endorsed by the broker. Every SP keeps a copy of the token and authenticates the token by hashing it $x$ times to produce $P_0$, which must match the one in the pricing contract. To validate the token ($P_x$) a corresponding endorsement token ($E_x$) must be received from the enforcer to prevent double-spending. Upon receiving $P_x$ the enforcer authenticates it, and forwards $E_x$ to all other service providers implicated in the contract. Every service provider validates the endorsement token by hashing it $x$ times to produce $E_0$, which must match the one in the pricing contract.

The role of the enforcer is to prevent double-spending, thus counterfeiting is possible if the client and the enforcer conspire. Therefore it is important to delegate the task of enforcer to a trusted service provider. Ideally it should be one of the accounting servers, but would overload them. Instead, a lightly loaded GSP would be nominated and must be trusted and compensated for providing enforcer services. When the enforcer fraud is detected (section 5.9 of [P00]), the Accounting Server can re-nominate a trust worthier GSP.

## 3.6 Multi Branch Architecture

Following the initial architecture described in section 3.5 we came to realise next that in order to successfully deploy the accounting infrastructure on the World Wide Grid it must scale well to potentially millions of GSCs and Providers participating in Grid markets. The Accounting Server represents the central point of failure and the bottleneck in the whole system unless the architecture is improved. The common approach to tackle such problem is to institute multiple accounting branches, such as the currency servers in the NetCash system [MN93].

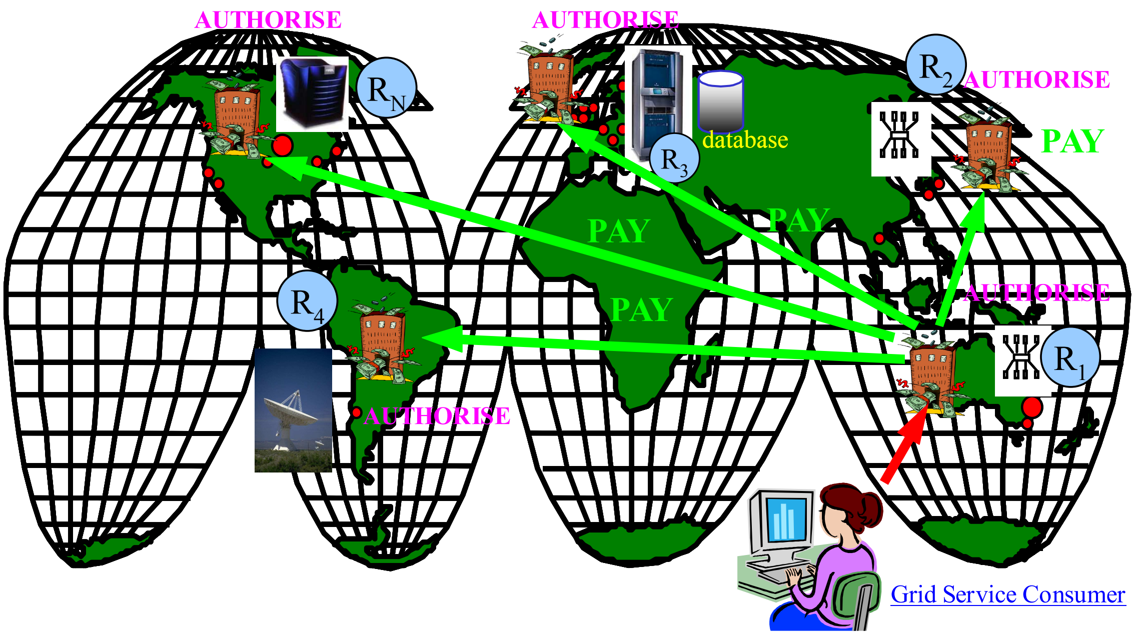


**Figure 3.9: World Wide Grid Accounting Architecture**

Figure 3.9 depicts geographically distributed resources under the control of different administrative domains managed by the branches. Each branch is responsible for transferring funds and sharing accounting information with other branches, authorising access to resources within its domain, which would have been off-limits without the computational economy. Ideally, GSCs should have an account at every branch (or at those whose resources are frequently used) to avoid a branch contacting another one "on-line" (in real-time) to confirm funds transfer and access authorisation, thus saving on transaction fees. However, in cases when a GSC does not have an account at the branch whose resource is needed immediately the Grid Accounting Services Architecture provides inter-branch protocols. Using the same currency for all branches is desirable, but not realistic. That is why there is a need for the currency exchange rate infrastructure. Competitive businesses would prefer actual money instead of virtual currency. Resources in Australia might charge in Australian dollars, in the United States – in American dollars, in Europe – in Euro. Section 3.9 elaborates on the issue of virtual versus actual currencies.

### 3.6.1 Account Numbers

The obvious question is how do branches authorise each other and how do they route payments to other authorised branches? For this purpose we use the account number infrastructure similar to the one used by actual banks. A GASA account number consists of two-character country code, four-digit branch number, and eight-digit local account number separated by dashes (e.g. AU-0001-00000001). The two-character country code is the International Standards Organisation's country code [ISO3166]. Country codes provide a good indication of the currency used by the branch when charging actual currencies.

The branch and local account numbers are in numeric form to be easily incremented to generate next unique number. The first six characters (excluding the dash) are used to identify a particular branch (e.g. AU-0001). Each accounting bank-branch code must be unique. We present a simple algorithm for assignment of bank-branch numbers. The protocol assumes that the URL of the central branch that allocates branch numbers is known by the new branch. An establishment of a new Accounting Server is not expected to happen often (figure 3.10).

We envision the World Wide Grid consisting of national Grids, which are composed of regional Grids that aggregate local Grids. Each bank code is assigned to a national Grid, implying the country assigned to the bank code. It is expected that a branch in a particular country will use the national currency of that country, and decisions regarding access to Grid resources managed by the branch are made without contacting the branch.

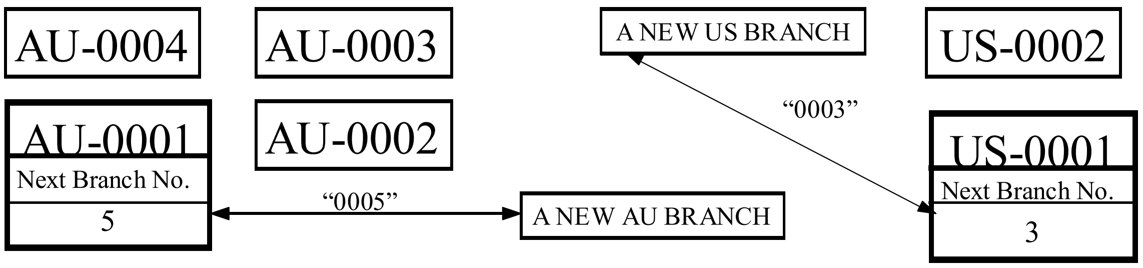


**Figure 3.10: Branch Number Assignment**

### 3.6.2 Inter Branch Payment Protocol

The multi-branch protocols should take into account the fact that branches can operate on different currencies. The exchange rate infrastructure must exist in order to satisfy this requirement. Exchange rates must be negotiated by branch administrators before transactions of such type can proceed. Using actual world currencies implies that the exchange rate used by a branch depends on the actual exchange rate provided by actual banks since reconciliation between branches with different currencies is expected to be completed manually by administrators, who transfer actual money between actual bank accounts. GASA merely simplifies Grid accounting by aggregating GASA transactions and allowing bulk transfers between actual banks to save on fees.

Considering a GSC from one branch and a GSP from another branch, the resource usage information is kept at the GSP's branch, and the GSC's workstation can optionally cache it for scheduling purposes. The reason for keeping Resource Usage Records at the GSP's site, and not at the GSC's, is because such data should be local (closer, faster access) to the GSP's site in order to predict the site's future hardware and software requirements based on previous usage and current demand. Replication of resource usage at the GSC's branch is unnecessary since the data is only useful to the GSC and is not expected to be used by any one else from the same site.

A generic inter-branch payment (cash, cheque, micropayment) protocol is illustrated in figure 3.11. This is a deferred payment model since the GSP receives the money after the branches reconcile accounts in step 5, and such operation is performed less often than inter-branch transactions.

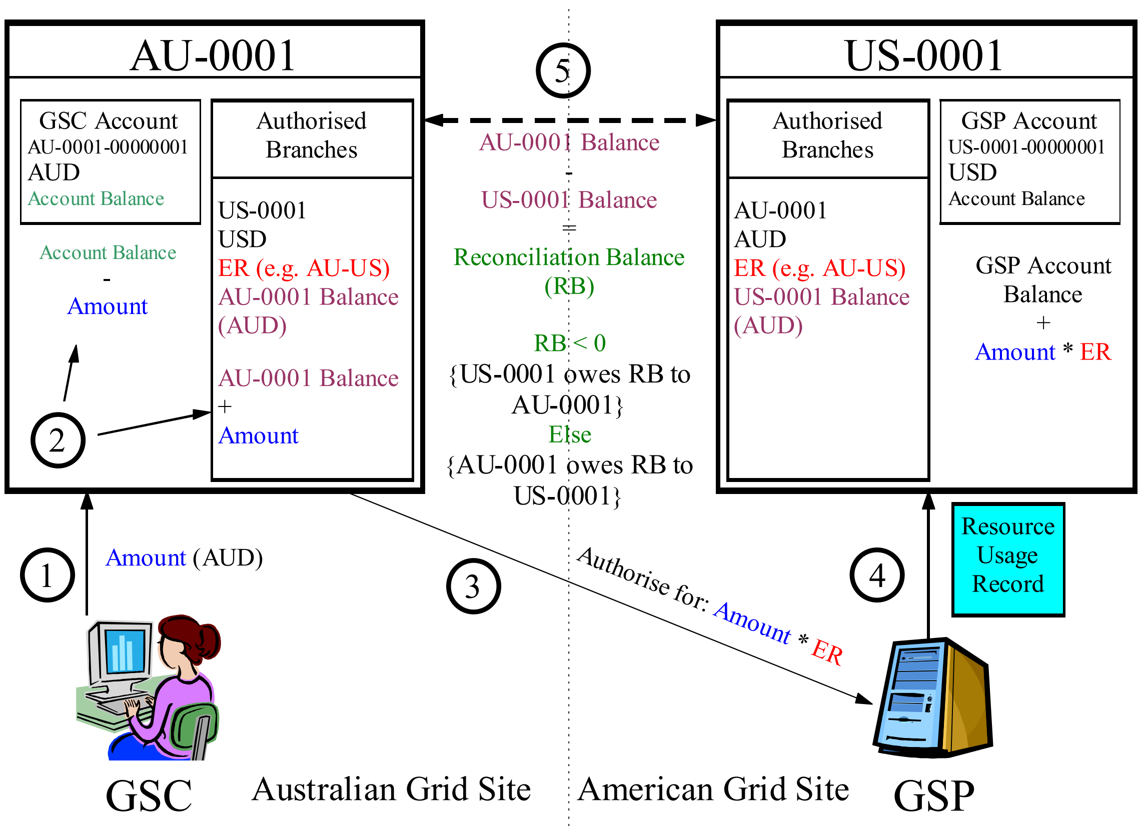


**Figure 3.11: Inter Branch Payment Protocol**

The numbers in figure 3.11 correspond to the following steps of the algorithm:

The GSC pays for resource access in local currency (e.g. AUD).

The local Accounting Server (AU-0001) subtracts the local currency amount from the GSC's account balance (places it into the "lock" balance) and adds it to the balance owed to the other branch (AU-0001 Balance). This is how actual banks operate [CSHM02].

The local Accounting Server (AU-0001) authorises access to the resource at the foreign site for the amount in the foreign site's local currency (e.g. USD = Amount(AUD) * ExchangeRate(AUD-to-USD)).

Upon computation completion, the GSP transfers the Resource Usage Record to the foreign Accounting Server, which stores the record for future reconciliation.

At the end of the financial period (i.e. when the exchange rate is renegotiated) the branches reconcile their balances by calculating how much each one owes each other. The corresponding balances must be in common currency (e.g. AUD). The balance of the foreign branch (e.g. US-0001 Balance) is subtracted from the balance of the local (e.g. AU-0001 Balance) to produce the *Reconciliation Balance*. If the Reconciliation Balance (RB) is negative, then the local branch (AU-0001) owes the RB to the foreign branch (US-0001), and the RB is converted into the currency used by the foreign branch (e.g. USD). Otherwise the foreign branch owes the RB to the local branch.

The above protocol assumes that the charge is known beforehand. However, in reality the run-time of a particular execution of an application is unpredictable due to various factors such as the current load (e.g. CPU, network). In such a case transactions must be reconciled based on the actual resource usage (i.e. after the GSP produces the Resource Usage Record). Section 3.8.1 deals with such situation.

### 3.6.3 Certificate Authorities Across Grid Sites

Figure 3.11 represents a scenario when GSPs at the foreign site (US-0001) recognise Certificate Authority of the local site (AU-0001). Another case is when sites have different Certificate Authorities as depicted in figure 3.12.

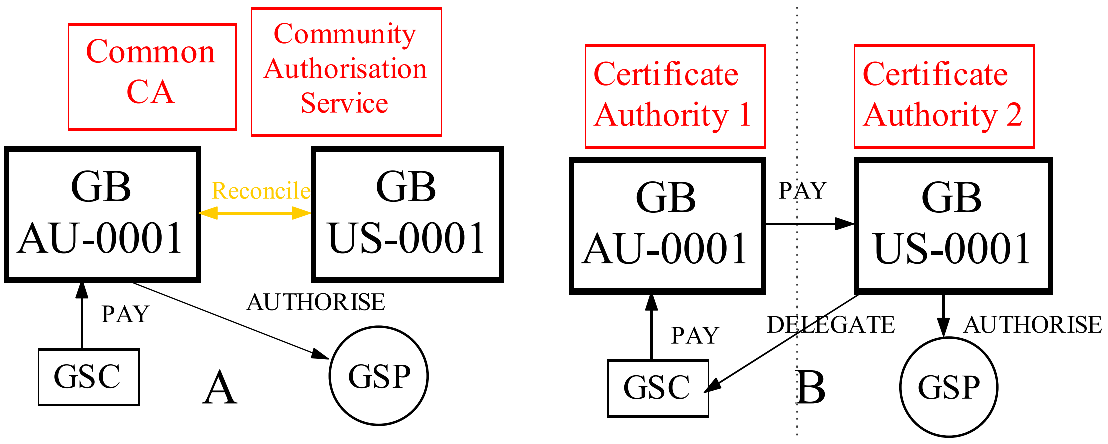


**Figure 3.12: Computational Economy Resource Access Authorisation Models**

Figure 3.12a shows the scenario represented by figure 3.11 and assumes that the two branches have at lease one common certificate authority. If the branches are assigned to different CAs or if advanced authorisation is required, then the Community Authorisation Service (section 2.4.1) can be set up to manage authorisation procedure. CAS offers advanced role-based access control; for example, some resources might require clients to be qualified to use the resource (e.g. a radio telescope). Figure 3.12b depicts the situation when neither a common CA nor a CAS exists and both sites have different CAs. In such a case authorisation is performed by the foreign branch, and the

GSC is delegated access credentials for the resource at the foreign site. The delegated credentials are then used to access the service normally (e.g. using the Globus Toolkit).

## 3.7 Currency

Traditional E-commerce and Internet-based financial transactions involve actual money. Countries have their own currency and when a transaction spans international borders, a currency of one country is converted via an exchange rate into currency of another. Banks use common currency between themselves to perform transactions. Natural resources such as gold, platinum, oil and others serve as the bond against which the money is issued. The calculation of the exchange rate between two countries is very complicated and depends on various political and economic factors such as the imports and the exports of goods and services relative to each other. The importance of universal money has been recognised by the European Union, which introduced one currency for Europe in 2001, the Euro (€).

The Grid Accounting Services Architecture proposes one standard Grid currency – the Grid dollar (G$). The Grid dollar poses as a standard middleware resource token, redeemable at any GridBank branch, or any other Grid payment system to emerge. This currency represents the market value of Grid resources, implying that two identical Grid resources might have different prices. By analogy, a desktop PC might cost 3000 € in Germany, but exactly the same architecture (perhaps assembled locally) would cost 2000 € in Poland. The precise correspondence of the Grid dollar to actual money is irrelevant and can be decided by people who manage Virtual Organisations.

The Grid dollar can be thought of as the trade token to barter Grid resources. Grid dollars earned at one site can be used to purchase services at other sites without worrying about actual money. In fact, *Virtual Private Grids* (VPG), by analogy with Virtual Private Networks, can be formed on the basis of one common currency recognised by all branches participating in the VPG. However, banks in the real world use some common denominator, which is usually the Reserve Bank. Central bank model implies central point of failure, and some countries experienced what is known as an economic default: bankruptcy of the Reserve Bank implying bankruptcy of all other commercial banks and total devaluation of the national currency [P03]. Instead of a central accounting server the Grid dollar can serve as the common denominator.

### 3.7.1 Exchange Rates

The exchange rate infrastructure described in section 3.7 is facilitated by the Grid Accounting Services Architecture. The reconciliation process, however, remains the responsibility of a *Grid Financial Administrator (GFA)* - the person responsible for the transfer of actual local currency from the VO's bank account by exchanging it into the receiving VO's currency.

An actual bank of one country might have different exchange rates from another country's bank. Exchange rates can be provided by the Reserve Bank, national and international commercial banks, and whatever rate is used, it must be agreed upon by both Accounting Servers if the barter (figure 3.11) is to take place. Each accounting branch may use different exchange rate to charge more local currency to compensate for the exchange rate fluctuations. As long as each branch pays the other branch in the other branch's currency, the inter-branch transactions can be reconciled. An Accounting Server must authorise other Accounting Servers explicitly as to prevent sudden unwanted demand for resources from other sites.

The accounting services can be extended by providing an authorised software agent (with administrator permissions) that observes the actual exchange rate variations and adjusts the exchange rate at the Accounting Server accordingly. The algorithm for such an agent must consider the exchange rate margin, *ERM*, which is the difference between the exchange rate from the actual bank, *AER,* and the Grid Accounting Server Exchange Rate, *GASER*. The ERM should reflect current AER fluctuations at the actual bank. If the AER "jumps" frequently by relatively large values, the ERM should be larger whereas a stable AER would result in a smaller ERM. The branches can optionally charge additional banking service fees. Figure 3.13 demonstrates an example.

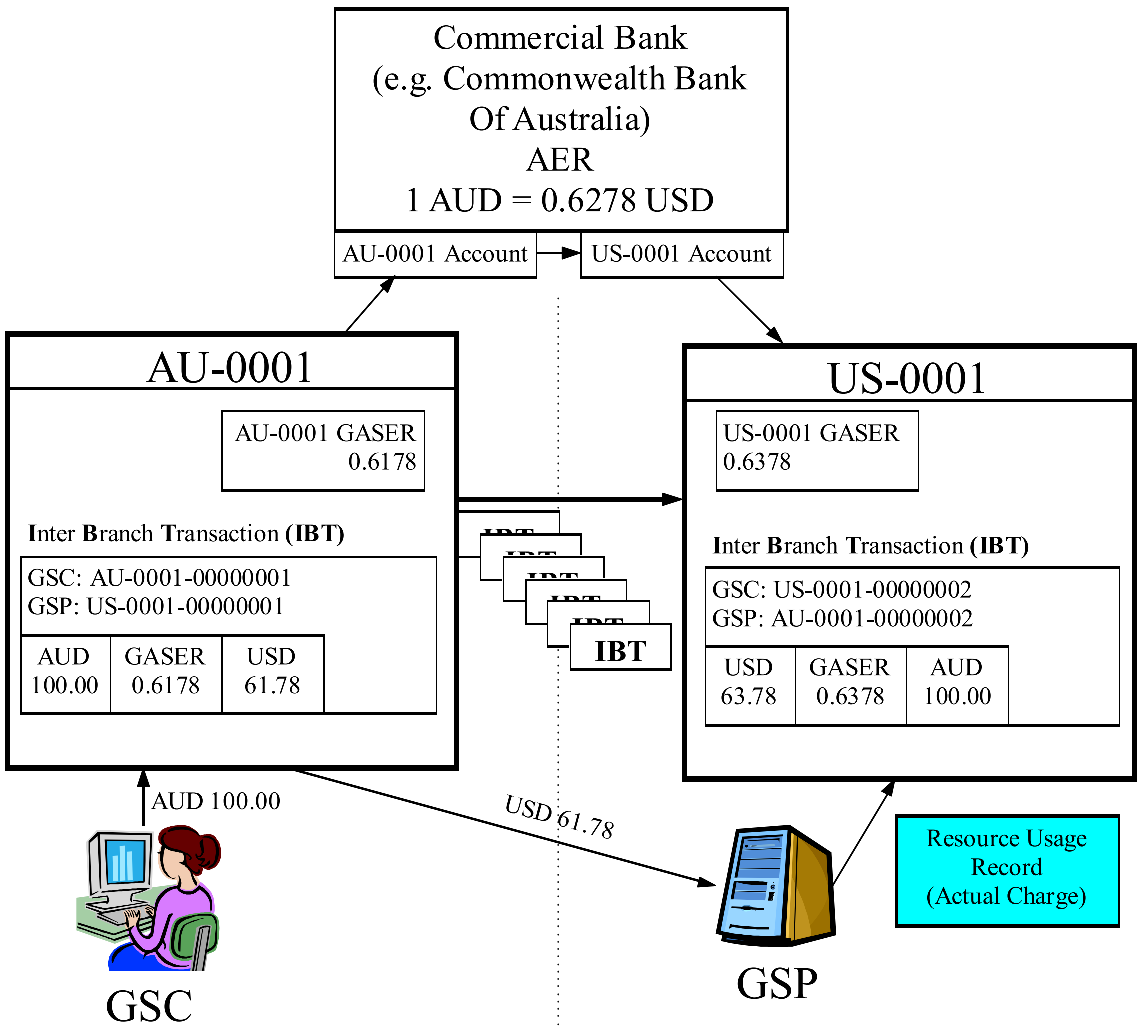


**Figure 3.13: Exchange Rate Margin Calculation**

There are three exchange rates used in the reconciliation process. One is provided by any actual commercial bank at which the two branches have accounts. This exchange rate must be agreed upon by both branches if the branches operate in barter mode – only compensate for the difference between amounts owed to each other. Otherwise the exchange rates are subject to each branch's discretion and can be freely solicited. Each branch is obliged to pay the total amount owed to the other branch. For example, figure 3.13 shows the Australian branch offering 1 AUD for 0.6178 USD while the commercial bank's rate is 0.6278.

Inter-Branch Transactions (IBT) are processed less frequently than actual bank transactions, and the commercial bank's rate (Actual Exchange Rate) might drop. Therefore offering a lower rate to consumers allows compensating for the loss (e.g. AUD100 is exchanged for USD61.78 to the consumer, whereas in fact the AU-0001 branch can obtain AUD100 * 0.6278 = USD62.78 from the actual bank). A drop of the

AUD-to-USD rate, on the other hand, would benefit the American branch. Vice versa, if the AUD-to-USD rate increases, the Australian branch would benefit and the American branch will lose. The Grid Accounting Services Exchange Rate is adjusted accordingly. The accounting branches can then calculate the Exchange Rate Margin (ERM = absolute_value(AER – GASER)) based on statistical analysis of AER fluctuations in the past. Each time the AER changes it is logged in the Accounting Server database along with the date and time.

Some resources require restricted access even in a free market economy. The accounting branches are allowed to issue their own currency for such resources, but the amount of currency issued must directly reflect resource reservations associated with it. The currency must be guaranteed by allocations in the resource pool. A separate exchange rate for this currency to actual monetary value must be set. The question is when should a site use an actual currency, and when can it operate with a virtual currency? Section 3.9 outlines a solution.

## 3.8 Economic Models for Grid Sharing in Market-based Environment

We envision a World Wide Grid with many local, regional, and national Grids. An Accounting Server can be set up at any level as long as it reflects organisational aspects. Each Grid site (local, regional, national) is expected to collaborate on a particular problem. Scientific problems include computational research in biochemisty, pharmacology, physics, astrophysics and a lot of other inter-disciplinary sciences. Entertainment Grids (aka interactive Grids) extend Web services by providing multi-media content that is easily shared between the participants by providing extra functions such as real-time (streaming) updates. As long as the Grid constructed for the purposes of a particular community a co-operative economic model should be adapted.

On the other hand, resource requirements of modern supercomputer applications sometimes exceed capacity of a single Grid site. It would be preferable to be able to solve such problems in less time. The GSC of such application might employ the services provided by other sites anywhere in the world. Since the resources of foreign sites belong to different Virtual Organisations with their own access policies and control, as long as there are middleware components that can negotiate access authorisation by providing payment and submit the task, the GSC can execute the application by leasing computational facilities in other Grids. Such environment precludes competition, since the Grid sites of different Virtual Organisations would sell off their resource surplus by offering competitive prices to obtain a greater return on their investments in the infrastructure. If demand exceeds available resources, then service rates are raised, and GSCs might select alternative sites. A competitive economic model should be adopted when sites do not trust each other as much as participants of their own Grid. Figure 3.14 demonstrates these concepts.

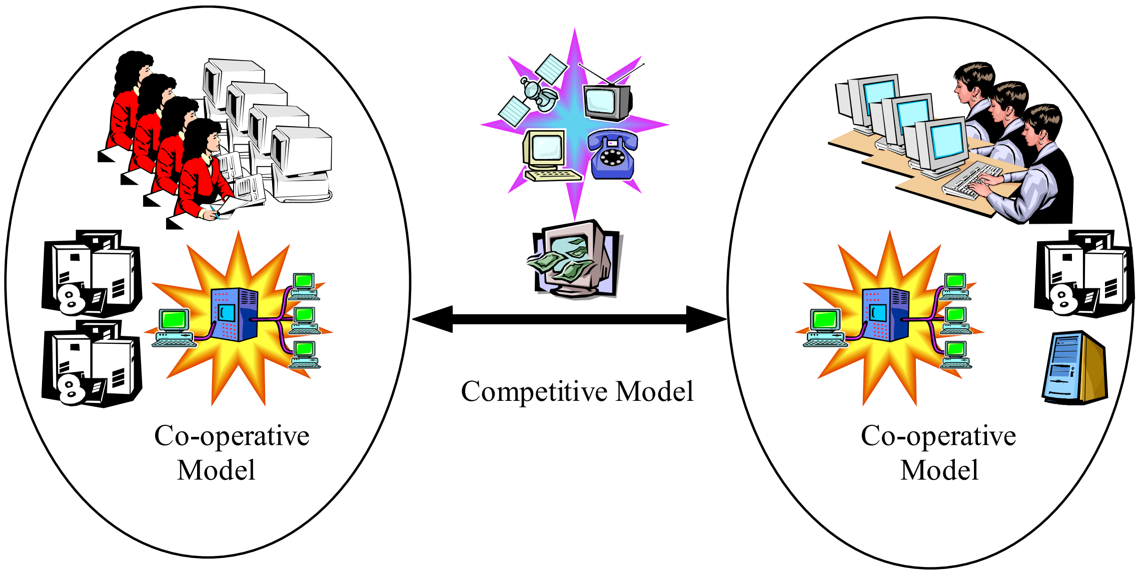


**Figure 3.14: Co-operative and Competitive Community Models**

### 3.8.1 Co-operative Model

In co-operative computing environments, all participants both consume and provide services; when participants provide services, they earn credits. They can use those credits to purchase access to other resources within the community when needed [B02a, BAG98]. Grid Accounting Services Architecture supports this form of resource barter. When new participants join the community, a certain number of credits are allocated to them. The allocation could depend on the value of the resource the participant is contributing to the community. The issue of what value is assigned to each resource needs to be resolved by the community and is outside the scope of GASA.

To achieve price equilibrium, supply and demand need to be carefully regulated in such a way so that GSPs are paid approximately as much currency as they will use to access other Grid services. Otherwise the whole economic environment will end up in a state where some participants, who do not require any services, have all the money while others who want to access GSPs have none. A community-based resource evaluation and pricing authority is needed to control the market equilibrium.

A co-operative community model essentially constitutes resource barter. Virtual currencies are a good alternative for this purpose. However, some co-operative environments might choose actual currency; a Grid operating with its own currency can be thought of as a co-operative environment. Trusted third-party financial authorities must perform Price regulations. Pre-pay payment mechanism is most appropriate since it optimises payment procedure by minimising communication with the Accounting Server. A direct payment protocol, which is similar to the GridCheque, but transfers the funds straight into the GSP's account instead of "locking" it, should be adopted if the application processing cost is known. Commodity Market, Posted Price or Contract/Tender models [B02a, BAG98] are most appropriate for the pre-pay payment methodology since the charge is a flat or a subscription fee, or is based on usage rates that are fixed.

### 3.8.2 Competitive Model

In competitive computing environments GSPs are allowed to solicit any price [B02a, BAG98]. However, to attract customers they need estimates of market value of their resources. The Accounting Server's transaction history can assist in deciding how much a computational service is worth. Such audit information is confidential and cannot be disclosed as is. Therefore an Accounting Server would receive a description of the resource, process the information in its database regarding prices paid for resources of similar type, and then produce an estimate. The simplest approach to compare resources is to consider hardware parameters (e.g. processor speed, number of processors, amount of main memory, secondary storage, network bandwidth). Another approach would be to use some common Grid application benchmark capable of indicating relative application performance.

Market prices can be adjusted in any way to regulate supply and demand. However, in order to maximise profits on their investments GSPs need good pricing strategies to gain competitive advantage. It has been recognised by marketing academics that the way organisations develop performance critical resources, including intangible resources such as tacit knowledge, internal networks and the creation of new intellectual capital, affects organisation's performance and ability to adapt to ever-changing business environment. Also emphasis is placed on sharing of resources and skills in co-operative strategies such as strategic alliances, and the allocation of resources to produce innovation [HCNC99].

Deferred payment (the funds are reserved, but not transferred) is most appropriate in a competitive business environment. GridHash and GridCheque are more appropriate for this payment model. Bargaining, Auction or Bid-based trading models [B02a, BAG98] allow to negotiate the price at application run-time. Resource monopoly can be applied to some resources (i.e. higher price for an outsider, lower for an insider) such that outside (competitive) demand is reduced to sustainable level. Resources offered on both, co-operative and competitive basis would have two price tags – one for the collaboration and one for those times when the resource is not used (i.e. offered to the World Wide Grid).

## 3.9 Summary

The Grid Accounting Services Architecture that extends GRid Architecture for Computational Economy has been developed to provide authorisation, payment, accounting, and audit services for the World Wide Grid. The architecture is sufficiently generic to be applied to any computational device with the concept of local account for the purposes of local process creation and file access control. The protocols presented incorporate authorisation, payment and accounting techniques to achieve efficiency. Using three separate sub-systems (i.e. authorisation system, payment system, accounting system) is undesirable and can lead to increased application response times gratuitously. Even worse, using existing payment systems implies that the user must authenticate twice: the first time to sign-on to Grid, and the second time to gain access to the account. GASA avoids such inefficiencies and can be easily integrated with other

Grid components and tools, such as the Nimrod-G resource broker (section 2.2.7) or the Community Authorisation Service (section 2.4.1).

GASA can be employed on the scale of Internet, or more precisely, on the scale of the World Wide Grid. A central accounting branch must be established in order to create other national and regional branches. It is recommended that one central accounting branch is created in each country, and the international English country code is assigned to it to represent the bank number. Subsequent branch numbers are assigned to other branches in incremental fashion. Bank-branch numbers ensure that payments are world-routable, and if any two branches authorise each other, then the resources of the two Grid sites can be shared and accounted for.

The choice of currency depends entirely on Grid Financial Administrators who must decide whether to use virtual or actual currency. Using virtual money in a co-operative manner has the advantage of being common, thus there is no need for currency exchange. The danger of using virtual funds is that inflation can occur if the GFAs do not have sufficient knowledge of economics. Therefore it is highly recommended to use actual currencies, even though GSCs might lose money when exchanging currencies.

Competing for computational resources inevitably reduces market prices as some providers sell their resources at a cheaper price. The competition between businesses in the real world ensures a healthy market growth and in general advances technological progress. Monopolies tend to slow down overall progress because they do not take as much risk to diversify their products. The Open Grid Services Architecture aims at diversifying Grid services and new lower-level Gird services might emerge because of the demands of higher-level services. GASA promotes e-commerce by providing the essential financial and accounting services.

The next chapter presents an implementation of GASA. The GridBank system utilises the services of the Globus Toolkit to provide a Grid accounting service. It proves the credibility of GASA while optimising Grid authorisation, payment, accounting and audit by integrating the concepts into one system.

# CHAPTER 4

# The GridBank System

## 4.1 Introduction

Computational Grids enable uniform and coordinated access to various types of geographically distributed heterogenous computing resources, owned by autonomous organisations, with diverse administration policies. Resource access authorisation in such complex Grid environments remains one of the key barriers in creating Grid communities on a global scale. Chapter 3 outlined the basic economic mechanism (i.e. user pays) that achieves World Wide Grid and Web authorisation for any kind of computing service, which has the abstraction of a local account for the purposes of process creation and file access control. As a proof of the Grid Accounting Services Architecture concept, GridBank is a World Wide Grid authorisation, payment, accounting and audit system leveraging Globus Toolkit middleware technologies. It offers monetary incentives for service providers to contribute their resources to the World Wide Grid and for service consumers to select their required resources considering current supply and demand, thus avoiding unnecessary resource load. GridBank records resource usage and the associated cost in a secure way that produces non-refutable evidence that a resource access occurred, what compute resources were engaged, and how much of each was consumed.

GridBank maintains the accounts of service consumers and providers and resource usage records in its database. The resource brokers and schedulers of GSCs gain access to the resource traders of GSPs using GridBank's protocols. It has been primarily envisioned to provide accounting services for establishing Grid computational economy. However, GridBank can be applied in other e-commerce applications that span international borders.

This chapter outlines the protocols and various external data representations used in the construction of the GridBank system. Essential GASA components are realised by the selected technologies and the reasons for the choices are presented. Authorisation, payment and accounting protocols are outlined first, followed by the multi-branch protocols that realise worldwide computational economy. GridBank applications are presented next along with other suggested uses. The complete GridBank and GSP APIs, external data representation XML-based records and the GridBank database tables are presented in the appendixes.

## 4.2 System Architecture

The GridBank communication protocol stack is shown in figure 4.1. The service consumer, the GridBank server, and the service provider are the three distributed system entities communicating with each other over public Internet.

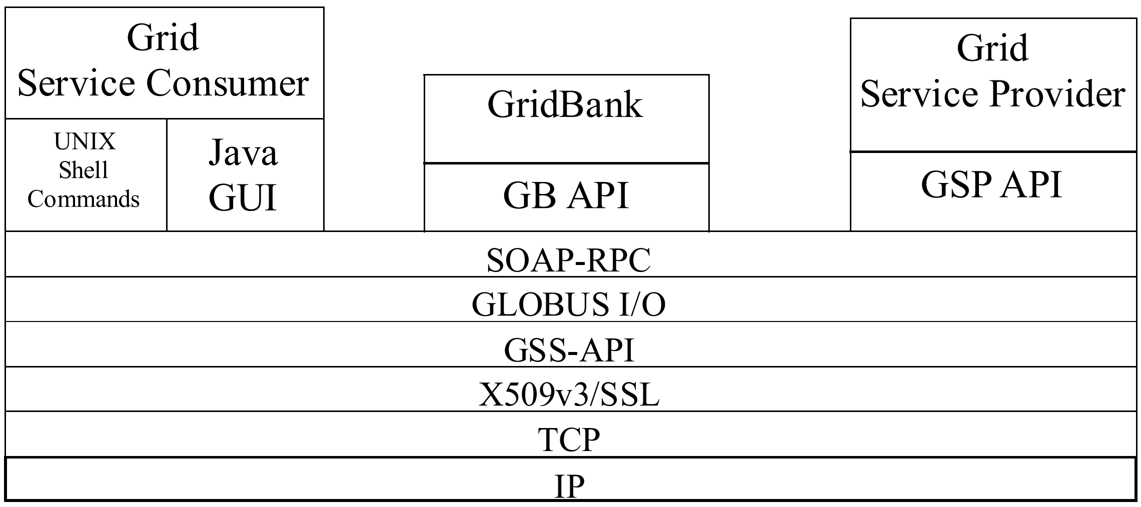


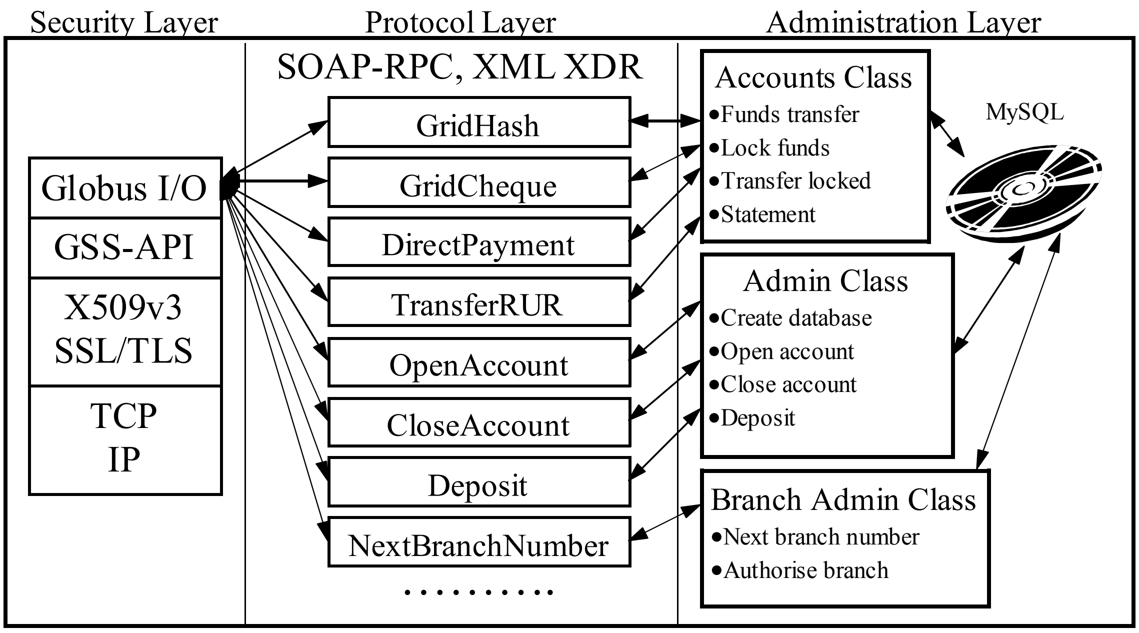
**Figure 4.1: GridBank Communication Protocol Stack**

The GridBank system employs the Globus I/O library, which is part of the Globus Toolkit, in order to be Grid-compatible; GridBank is just another Grid service. GSCs use the same Grid proxy certificate to access the GridBank service as they use to authenticate to other services on the Grid. A proxy certificate is a temporary certificate issued for the duration of a computation by the user and it prevents a malicious process, to which proxy credentials are delegated, causing prolonged damage. The user's identity is verified by checking the signature of the proxy certificate using the public key from the user's certificate. The user's private key would be safe even if the proxy's private key were compromised. Using Globus I/O is one of the reasons for developing GridBank instead of using existing payment systems discussed in chapter 2, thus preserving the Grid's single sign-in policy of avoiding repeated entry of the user password each time a GSC requires to access another Grid service.

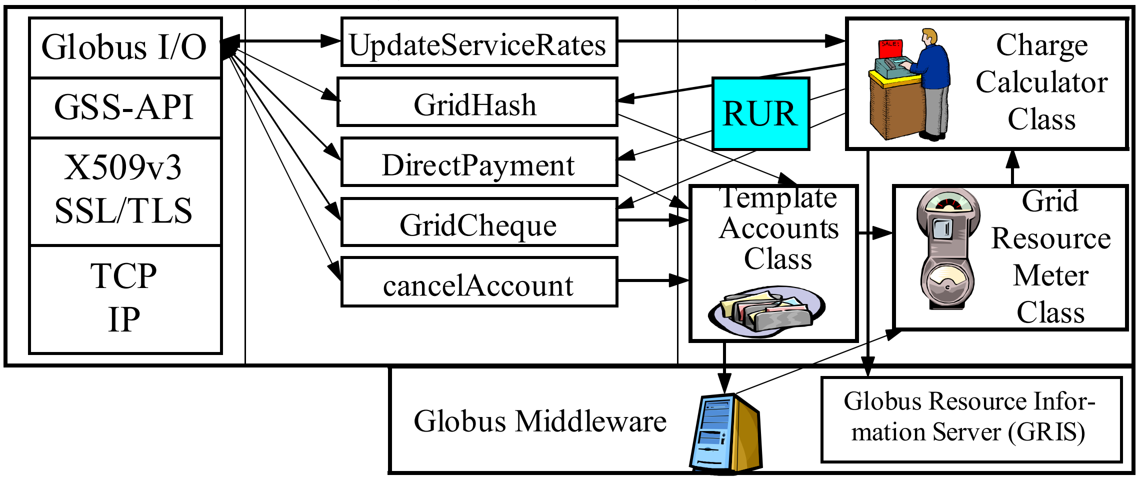
The Globus I/O library builds on top of the Generic Security Services interface (i.e. the GSS-API). GSS-API separates the security operations such as authentication from the underlying implementation as described in section 2.3.1. Globus I/O implements GSS-API with X.509v3 digital certificates (Public Key Infrastructure) and the SSL/TLS protocol. PKI serves the purpose of secure exchange of a secret (session) key for the SSL protocol, which uses TCP/IP as a reliable transport protocol. A secret key is randomly generated by the server process for each accepted (authenticated) connection and the session key is encrypted with the client's public key and is transferred to the client over public Internet. Since only the client can decrypt the message with the private key to obtain the session key, the communication channel is secure.

Globus I/O provides the security abstraction, which ensures authenticity, integrity and confidentiality of any received messages, thus it constitutes the Security layer of Grid Accounting Services Architecture (figure 2.1). Building on top of Globus I/O is the Simple Object Access Protocol (SOAP). SOAP was chosen as the means to invoke remote API methods because it allows creation of loosely coupled systems. The GridBank service can be dynamically discovered by resource brokers that wish to acquire other Grid resources managed by a GridBank branch. Therefore, if one of the GridBank, GSC or GSP system entities is re-implemented in some other programming language (e.g. Java), then the rest of the system can remain unchanged and retain its functionality. Another reason is the current advance towards Web service implementation of Grid middleware components using Web service technologies (i.e. XML, SOAP and IPv6) [FKNT02, TCFF02].

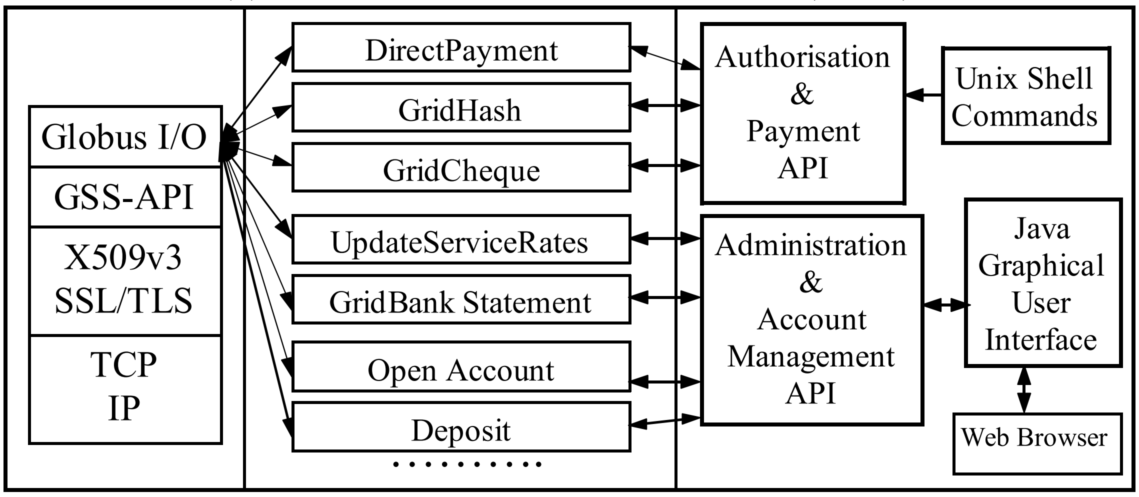
It is expected that Globus I/O is sufficiently secure for security purposes by providing different Quality-of-Protection (QoP): a choice of 512-bit, 1024-bit, 2048-bit and 4096-bit public-private key length. As Grids become more powerful, brute force attacks on PKI become more practical and threatening. Increasing the number of bits used in a key pair counteracts this form of attack.



**(A) GRIDBANK BRANCH (GBB)**



**(B) GRIDBANK SERVICE PROVIDER SERVER (GBSPS)**



**(C) GRIDBANK SERVICE CONSUMER CLIENT (GBSCC)**

**Figure 4.2: GridBank System Architecture**

Figure 4.2a depicts the GridBank server as GridBank Branch (GBB), figure 4.2b – the service consumer as GridBank Service Consumer Client (GBSCC), and figure 4.2c – the service provider as GridBank Service Provider Server (GBSPS) entities that realise Grid Accounting Services Architecture. The modules within the entities are implemented as C++ classes, APIs are implemented by C libraries, and a client implemented in Java programming language is interfaced with C libraries via Java Native Interface (JNI) to provide a Web interface. C and C++ programming languages were chosen for the API implementations for efficiency's sake, whereas the client GUI is implemented in Java programming language for portability and to be capable of presentation in a web browser (as an applet).

The authorisation, payment, accounting and audit protocols use the SOAP protocol to define GridBank protocol's method signatures. Web Services Definition Language (WSDL) documents describing GridBank and GridBank Service Provider APIs are presented in appendixes B.2 and C.2 respectively. WSDL is the Web service standard for self-describing services that can be dynamically discovered and invoked. A GridBank service consumer is loosely coupled with the GridBank server and provider. All external data representations are in XML. An arbitrary data structure can be defined in XML and can be easily extended to include extra elements in the new version without affecting previous versions. XML's extensibility is one of the reasons for choosing it as the external data representation. The current trend towards the Web services implementation of Grid services is another reason. However, the GridBank server's database is implemented by the MySQL relational database management system rather than an XML database for efficiency's sake and easy deployment on Linux platforms.

## 4.3 Establishment of GridBank Branches and Account Management

A Grid administrator creates a GridBank Branch by executing the create branch shell command that is part of the GridBank software. The central branch that keeps the next unique branch number must be known beforehand. A central branch for each country must be created in order to create other branches. When a new branch is being created, whether central or not, the certificate subject name of the initial (default) administrator must be supplied. That administrator will be the only one capable of accessing the branch initially and can use the GridBank Service Consumer Client Graphical User Interface to create other administrators to manage user accounts. In the following sub-section the Globus Toolkit's authentication and authorisation procedure is closely examined in order to prove that the GridBank protocols are secure.

### 4.3.1 Globus Authentication and Authorisation Procedure

The Globus client and server perform mutual authentication (figure 4.3) as described in section 2.3.1. Once authenticated, Globus I/O calls a server-defined authorisation callback function, which is implemented by the server programmer and is responsible for determining whether the authenticated entity is authorised. The default Globus Toolkit (version 1 and 2) authorisation callback function simply searches the authorisation file. On UNIX-like file systems this file is located in */etc/grid-security/grid-mapfile*. This authorisation file contains mappings from authorised certificate names to local system account names (i.e. the UNIX log-in name). The client is treated as if the client is logged in locally. The purpose of the local account is to determine and constrain local file access for all local processes started by the remote client.

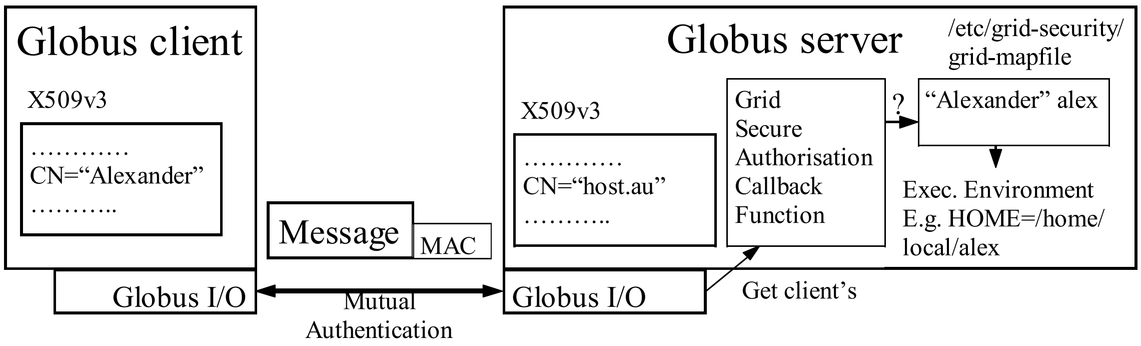


**Figure 4.3: Globus Authorisation Procedure**

Once the client and the server mutually authenticate, a secret (session) key is created and exchanged using PKI; the server encrypts the key with the public key of the client, thus only the client can decrypt and obtain the key because only the client possesses the private key. The session key is subsequently used by the SSL protocol to authenticate every message sent. By default, SSL in the Globus Toolkit generates a Message

Authentication Code (MAC) [B00, FKTT98], which is used to detect if the message has been tampered with. Optionally, SSL can symmetrically encrypt messages for maximum security. Essentially the MAC mimics digital signatures of the client and the server since they are mutually authenticated.

### 4.3.2 GridBank Branch Creation and Management

The branch creation program requires the certificate name of the initial administrator (e.g. “\O=Grid\OU=cs.uwa.edu.au\CN=Alexander Barmouta”). This creates an entry in the “administrators” table (appendix A.4) with “CertName” and “CreatedBy” fields both set to the certificate name string. This means that only the initial administrator will be able to access that branch. Connections from any other clients will be refused. GridBank server implements its secure authorisation callback function by searching the GridBank’s database (i.e. “administrators” table) instead of the authorisation file (i.e. */etc/grid-security/grid-mapfile*).

Once a branch is created, the initial administrator has to sign-in to the Grid and then use the GridBank Service Consumer Client’s Graphical User Interface to perform required operations. Another administrator is created by clicking on the “create administrator” button and then entering administrator’s certificate name. Other operations include “authorise branch”, “open account”, “change credit limit”, “deposit”, “withdraw”, “close account” and “cancel transaction”. Only a legitimate administrator can invoke these functions since GridBank will search “CertName” fields in the “administrators” table (appendix A.4) in its database to match against the certificate name extracted from Globus I/O during the connection.

### 4.3.3 GridBank Account Creation and Management

Any authorised administrator can open an account. An administrator must enter the certificate name of the user, the currency name, which is virtual (e.g. G$) or actual (e.g. AU$), and, optionally, any account details (e.g. client’s email address). The account’s available and “lock” balances and the credit limit are initialised to zero. The account can also be used in authorisation and accounting mode only in case a GSP chooses to provide its service for free. The GridBank Service Provider Server can be configured to disable charging. Therefore a client can request a payment with the amount specified at zero, which serves as resource access authorisation.

Only an administrator can deposit and withdraw funds. It is expected that the actual money kept with GridBank would be transferred by actual banks. Depositing virtual currencies into accounts signifies relative resource allocations among co-operating GridBank clients. It is expected that one account be opened for each GSC, and another one for a GSP. Once an administrator deposits currency, a GSC uses payment protocols to gain access to Grid resources. A funds transfer operation without authorisation procedure can be used in case same physical person is a GSC who administrates a resource in some collaboration. The person's computer earns credits during the times when idle, and the person uses those credits to access other people's machines when the machines are unoccupied.

## 4.4 GridBank Protocols

Grid Accounting Services Architecture protocols required slight modification in order to integrate them with the Globus I/O library. The library does not provide a function to digitally sign (with the private key of GridBank) an arbitrary document such that the signature can be later verified with the public key from the GridBank's certificate. Therefore it is not possible to generate a cash, cheque or micropayment document for storage on the GSC's computer for later use. Instead, Globus I/O is used to establish secure communication channels, which are used as real-time authorisation, payment and accounting data transporters that provide means of non-repudiation. Community Authorisation Service (section 2.4.1) has similar functionality; when a client contacts a CAS server, the server notifies the GSP of the client's certificate name to authorise. It is not possible to implement the GridCash protocol using current Globus I/O functionality (more specifically the Grid Security Infrastructure), and consequently it is replaced by the DirectPayment protocol.

## 4.4.1 DirectPayment

The first payment protocol implemented is named DirectPayment. The protocol withdraws the requested amount from the GSC's account and immediately deposits it into the GSP's account. This protocol is similar to an actual bank funds transfer when the amount to be paid is known. It is based on the pre-pay methodology and is useful among participants who charge a fixed (subscription) fee. The protocol is illustrated in figure 4.4.

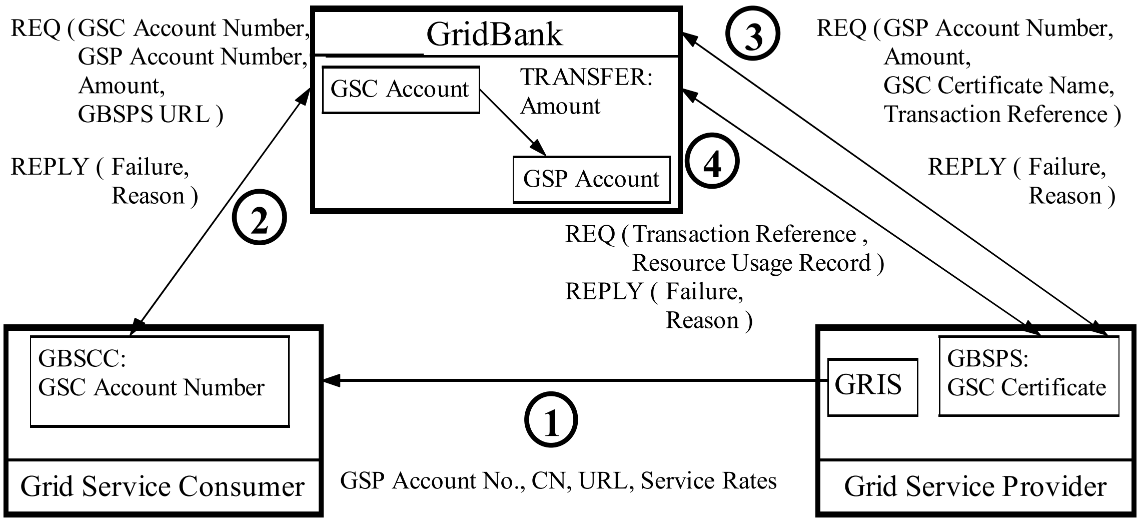


**Figure 4.4: Direct Payment Protocol**

In step 1 a GSC contacts a GSP's Globus Resource Information Service (GRIS) [CFFK01, FK99], which is part of the Globus Toolkit, to obtain service data. Service data for a computational resource includes hardware specifications (e.g. CPU speed, memory size) as well as the software available. This is part of Globus service discovery process that a GSC typically performs. In order to optimise GridBank protocols the Grid Trade Server is implemented by the Globus Resource Information Service. The optimisation comes from the fact that a service consumer contacts GRIS normally and does not need to contact a Grid Trade Server if it was implemented as a separate service. Service rates, the GSP's account number, the certificate name, and the URL of the authorisation server (i.e. GridBank Service Provider Server) are stored on the GRIS along with other service data. GRIS is based on the Light-weight Directory Access Protocol (LDAP) [CFFK01].

In step 2 the GSC invokes the direct payment (appendix B.1.2) by sending to GridBank a request (denoted as REQ in figure 4.3) with a 4-tuple containing the GSC's account number, the GSP's account number, the GridBank Service Provider Server's URL, and the amount. GridBank transfers from the GSC's account to the GSP's account and creates a transaction entry (in "transfers" table; appendix A.14) with a transaction identification number, which is stored in the table "nexttrans" (appendix A.13), and is positive integer for the ease of incrementing it. Other transaction details include the date of entry, the GSC's account number, the amount in local currency, and the GSP's account number. The transaction is also entered into the account history table (appendix A.9). The transaction number, the date and the amount are the same as in "transfers", the type is "transfer", and the "DoneBy" field contains the certificate subject name of the transaction initiator to ensure non-repudiation. The transaction history table is needed to record deposits and withdrawals, which are performed by an administrator.

In step 3, GridBank makes an authorisation request (appendix C.1.2) by addressing to the specified GridBank Service Provider Server URL a 4-tuple containing the GSP's account number, the amount, the GSC's certificate name to authorise and the transaction reference that will be used to associate the resource usage record with the transaction. The GBSPS allocates a template account and the result of the operation (denoted as REPLY in figure 4.3) is returned to GridBank. All GridBank protocols return the standard reply (operation result) consisting of an integer, which is the error number, and a string, which is the reason for the error. A successful operation has zero as the error code and the string is context-dependent, that is, it would contain the result of the operation. For example, in a request statement operation reply there would be the statement in XML format.

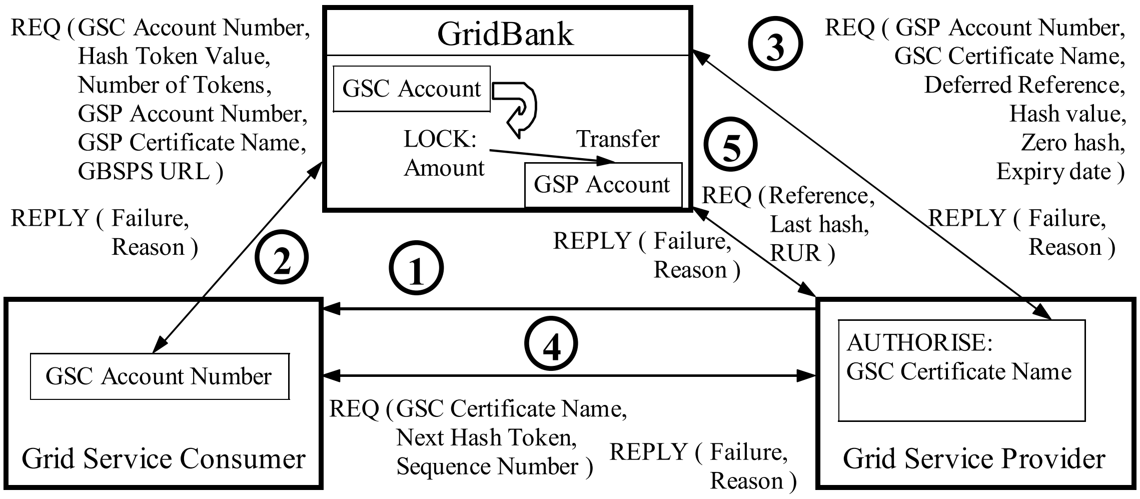
When GridBank receives the reply of the authorisation request (step 3), it forwards the reply to the client. In case of an error, for example, if the GBSPS runs out of template accounts, GridBank cancels the transaction. Upon successful response the GSC submits jobs normally using the modified Globus Toolkit version 2.0. The Globus Resource Allocation and Management (GRAM) protocol (section 2.2.6) was modified to record resource usage of submitted tasks by writing them to a text file. Grid Resource Meter reads the file and converts the data into a standard XML-based resource usage record. In step 4 the record is transferred to the GridBank server by sending a 2-tuple containing the transaction reference and the record. GridBank parses the record and converts it into an entry in the RUR table (appendix A1). Only the GSP can invoke the transfer resource usage record operation as the GSP's certificate name from the connection is matched against the one recorded in the transaction entry; the entry (in the "transfers" table; appendix A.14) contains the GSP's account number, which is used to look up the certificate name in the "accounts" table (appendix A.2).

The communication channel in step 1 is in clear text. Account numbers, certificate subject names and locations of authorisation servers do not pose a security problem because access to accounts and resources is protected by Globus I/O (i.e. SSL/TLS). In order to gain access to an account a client has to possess the private key associated with the certificate subject name recorded in the account details (i.e. "accounts" table). Submission of computational tasks requires the GSC to be authorised (i.e. entry in */etc/grid-security/grid-mapfile*). The communication channels in steps 2, 3 and 4 are mutually authenticated and encrypted using Globus I/O. In step 2 GridBank authorises a connection from a client by extracting the client's certificate name from Globus I/O and matching it against the account referred to by the GSC's account number specified in the request. For step 3 to succeed, each GSP must authorise the trusted GridBank server by placing GridBank's certificate subject name into the authorisation file (i.e. */etc/grid-security/grid-mapfile*). The authorisation procedure is elaborated upon in section 4.4.4.

## 4.4.2 GridHash

GridHash is a micropayment protocol for participants with medium trust in situations when the service cost is unknown. Authorisation is performed once at the start of a transaction and subsequent payments are made without contacting the GridBank. We chose the MD5 secure one-way hash function [R92] to generate hash tokens, as it is freely available on most platforms. The protocol is depicted in figure 4.5.

Step 1 is as in the DirectPayment protocol. In step 2 the GSC requests a GridHash chain by sending a request with a 6-tuple containing the GSC's and the GSP's account numbers, the GSP's certificate name, the GridBank Service Provider Server's URL, number of hashes required and a monetary value assigned to each hash. GridBank "locks" the funds by transferring the amount from the available balance to the lock balance and creates a deferred payment transaction (in "deferred" table; A1) with a deferred reference number obtained from "deferred_id" table, the GSC's account number and certificate name, the GSP's account number and certificate name, the total amount, the number of hash tokens, the timestamp and the expiry date. By default, the expiry date is one year more than the date of payment issue.



GridBank
GSC Account
LOCK: Amount
Transfer
GSP Account
REQ ( GSC Account Number, Hash Token Value, Number of Tokens, GSP Account Number, GSP Certificate Name, GBSPS URL )
REPLY ( Failure, Reason )
2
1
3
REQ ( GSP Account Number, GSC Certificate Name, Deferred Reference, Hash value, Zero hash, Expiry date )
REPLY ( Failure, Reason )
5
REQ ( Reference, Last hash, RUR )
REPLY ( Failure, Reason )
4
GSC Account Number
Grid Service Consumer
REQ ( GSC Certificate Name, Next Hash Token, Sequence Number )
REPLY ( Failure, Reason )
AUTHORISE: GSC Certificate Name
Grid Service Provider

**Figure 4.5: GridHash Protocol (based on figure 3.6)**

In step 3 GridBank sends the GBSPS a 6-tuple containing the GSP's account number, the GSC's certificate name, the deferred transaction reference, the monetary value of one hash token, the zero hash (used to authenticate subsequent hash tokens) and the expiry date. Every payment period when the next payment token is required the GSC sends the next hash to the GSP in step 4. When the Grid application job completes execution, in step 5 the GSP updates GridBank by sending a 3-tuple containing the deferred transaction reference, the last hash token received from the GSC, and the resource usage record.

### 4.4.3 GridCheque

The GridCheque protocol is an electronic cheque protocol for participants with maximum trust and when the service cost is unknown. It is more secure than the GridHash protocol due to the fact that hash tokens are faster to break than the private key of GridBank. The protocol is depicted in figure 4.6.

Step 1 is as in the DirectPayment. In step 2 the GSC sends a cheque request with a 5-tuple containing the GSC's account number, the maximum amount, the GSP's account number and certificate name, and the authorisation server URL. GridBank creates a deferred transaction as in the GridHash protocol, but the number of payments is fixed at one. The amount is reserved by subtracting it from the available balance and adding it to the “lock” balance. In step 3 GridBank invokes the authorisation call (appendix C.1.5) with a 6-tuple containing the deferred transaction reference used to redeem the cheque, the GSP’s account number to which the amount will be credited upon presenting the resource usage record, the GSP’s certificate name to confirm (since only that GSP would be able to redeem the cheque), the maximum amount (budget), the expiry date and the GSC’s certificate name to authorise. Provided all the details are correct the GSP returns a successful reply, which is forwarded by GridBank to the GSC, who submits jobs via Globus middleware.

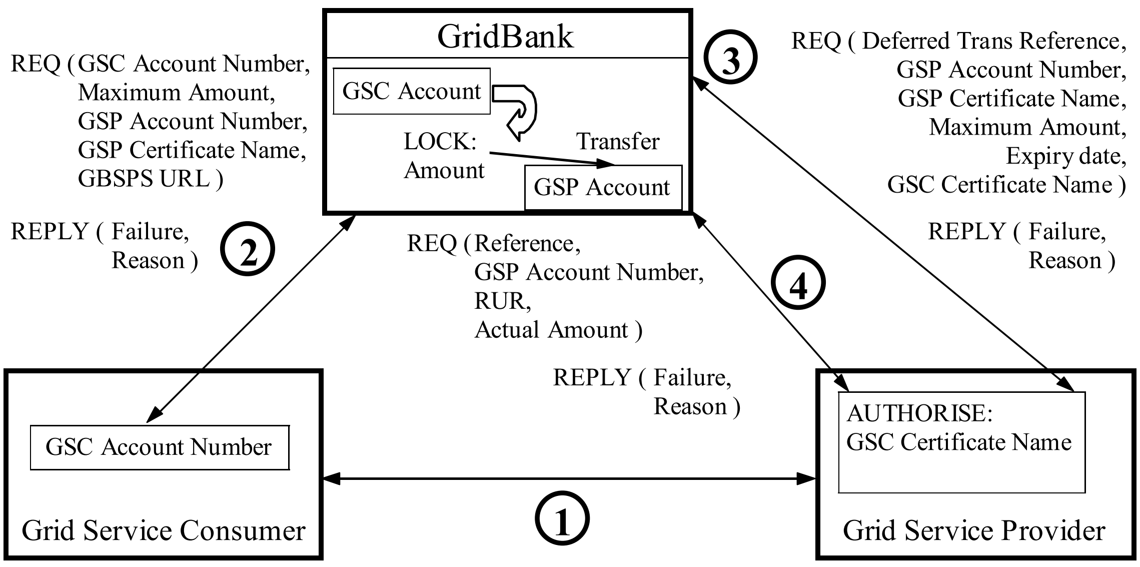

GridBank
GSC Account
LOCK: Amount
Transfer
GSP Account
REQ (GSC Account Number, Maximum Amount, GSP Account Number, GSP Certificate Name, GBSPS URL )
REPLY ( Failure, Reason )
2
3
REQ ( Deferred Trans Reference, GSP Account Number, GSP Certificate Name, Maximum Amount, Expiry date, GSC Certificate Name )
REPLY ( Failure, Reason )
REQ (Reference, GSP Account Number, RUR, Actual Amount )
REPLY ( Failure, Reason )
4
GSC Account Number
Grid Service Consumer
1
AUTHORISE: GSC Certificate Name
Grid Service Provider

**Figure 4.6: GridCheque Protocol (based on figure 3.7)**

When jobs are complete, the GSP reports the resource consumption by sending a 4-tuple containing the deferred transaction reference number (for the ease of searching the database), the account number, the resource usage record and the actual amount, which must be less than or equal to the maximum amount originally allocated. The GSP’s certificate name is extracted from the connection and is matched against the one recorded in the deferred payment entry. Upon successful authentication and authorisation, GridBank subtracts the actual amount from the maximum amount stored in “lock” balance and deposits the difference into the GSC’s available balance. The actual amount is deposited into the GSP's account. The deferred payment entry is deleted from the database.

### 4.4.4 General Authorisation, Payment and Accounting Protocol

After examination of the three protocols closely, the following generalisation was drawn from them. A general authorisation, payment and accounting procedure starts with a GSC requesting the service data from a GSP. The service data for a computational service includes resource characteristics such as CPU speed, main memory and secondary storage capacities and other. Such information is maintained by the Grid Resource Information Server (GRIS), which is part of the Globus Toolkit. Since every resource provider already runs GRIS in order to advertise the service to clients, the provider's account number, certificate name, URL of the GridBank Service Provider (authorisation) Server and the service rates are stored on the GRIS as part of the service data for efficiency's sake. Implementing a separate Grid Trade Server implies inefficiency because once a particular resource is chosen by a resource consumer, another network request must be made to retrieve the trade information.

The next step is to obtain resource access authorisation through GridBank. A client, using GridBank Service Consumer Client module, contacts the GridBank server by invoking a SOAP call corresponding to the desired protocol. The parameters of the call include the account number of the client (and the client's certificate name is obtained from the Globus I/O connection), the account number of the service provider, the certificate name of the provider, the URL of the GridBank Service Provider Server, the amount and, optionally, other protocol-specific data (e.g. number of hash tokens).

Upon receiving a client's request, GridBank extracts the certificate name of the client from the Globus I/O library. GridBank validates the request by ensuring that the account number supplied by the client and the certificate name extracted from the connection match the corresponding account details (i.e. account number and certificate name) in the GridBank's database (i.e. "accounts" table; appendix A.2). The next step is to allocate sufficient funds from the available balance of the client's account. We have generalised the payment modes used in GridBank protocols to the following:

Direct payment, or

Deferred payment.

In the direct payment mode (i.e. DirectPayment protocol) GridBank withdraws the specified amount from the client's account and immediately deposits it into the provider's account. In case the provider's account is at another branch, the funds are transferred into the other branch instantaneously, exchanging money if the branches operate on different currencies. The exchange rate is fixed in this payment mode because once a direct payment transaction is processed "on-line", it is considered completed. Section 4.5 elaborates on inter-branch protocols. The direct payment is only useful when the service total cost is known beforehand and the trust between a client and a provider is minimal. The direct payment mode guarantees payment as funds are transferred immediately, but has the disadvantage of being a heavyweight protocol – there is an extra communication between the two branches involved in a transaction.

In the deferred payment mode GridBank withdraws the requested amount from the client's available balance and deposits it into the client's "lock" balance. Effectively, it is not a withdrawal, but rather a mechanism that assures availability of funds for later reconciliation. An entry in the "deferred" table (appendix A.7) is added, which states the reserved amount, the client's and provider's account numbers as well as provider's certificate name. Only the GSP would be able to redeem the payment by authenticating to GridBank. The number of payments indicates how many tokens were issued for the "locked" amount. Cash and cheque payments have one, whereas hash chains typically contain thousands of tokens. The expiry date is also entered as an extra security precaution to counter-act brute force attacks on digital signatures (chapter 2). The deferred payment reference is issued, which is later used by the provider to claim the payment. The actual amount claimed must be less than or equal to the reserved amount. The Resource Usage Record is the evidence of the actual amount. The deferred payment mode is useful when the application processing cost is unknown and there is a medium to high trust among the participants.

Once the payment is guaranteed by GridBank, the next step is to authorise the resource access. Using the specified URL, the GridBank server contacts the GridBank Service Provider Server and notifies the provider of the account number where funds will be transferred upon job completion (i.e. provider's account number), the amount, the certificate name of the client to authorise, the transaction reference, and other protocol-specific data (e.g. number of hashes). The GridBank Service Provider Server only accepts connections from the GridBank; it maintains an entry in the authorisation file that contains the certificate name of GridBank. The GSP ensures that the recipient's account number and certificate name are its own and performs authorisation. As mentioned, GridBank was implemented to be used in conjunction with the Globus Toolkit version 1 and 2 on a UNIX-based platform (e.g. Linux). The GridBank Service Provider Server process simply creates an entry in the authorisation file (i.e. */etc/grid-security/grid-mapfile*) containing the specified (by GridBank) client's certificate name followed by an allocated local template account name (as described in section 3.5.2.1). A GSP maintains a pool of free template accounts (i.e. the */etc/grid-security/template-accounts* file). When a template account is de-allocated, it is removed from the authorisation file together with the client's certificate name, and the template account name is written back to the template accounts file. Only the resource administrator has the authority to create the template accounts.

Upon successful allocation of a template account the GridBank Service Provider Server sends back a confirmation to GridBank, which forwards it to the client. The GridBank cancels the transaction in case of an error (e.g. template accounts were exhausted). The GSC that receives a successful reply from the payment request can submit jobs normally using Globus middleware (i.e. the Globus Resource Allocation and Management protocol).

The GridBank Service Provider Server instantiates a Grid Resource Meter by supplying it the client's certificate name. Grid Resource Meter's functionality relies on the modified Globus Toolkit's GRAM (Globus Resource Allocation and Management) protocol. The modification provides the means of obtaining resource usage information extracted from the operating system. Each time a client submits a job via Globus, the modified GRAM server software writes the job request into the job requests file. The job requests file name is stored in the */tmp* directory and its file name consists of the client's certificate name with all '\' characters replaced by '_' characters followed by a dot and "requests" file extension (e.g. *"\O=Grid\OU=cs.uwa.edu.au\CN=Alexander Barmouta*" becomes
*_O=Grid_OU=cs.uwa.edu.au_CN=Alexander_Barmouta.requests*) because the UNIX

file system recognises the '/' as the directory separator. The requests file ensures that all jobs submitted via the GRAM protocol are accounted for. An entry in the requests file contains the job sequence number, GRAM job identification, and the date (and time) of entry. When a job completes, its resource usage data is written by the modified GRAM into the resource usage file, whose name is similar to requests file but followed by the extension "rur" and the job sequence number (e.g. *_O=Grid_OU=cs.uwa.edu.au_CN=Alexander_Barmouta.rur.1*). Job sequence numbers start at one and are incremented by one.

The Grid Resource Meter is spawned by the GridBank Service Provider Server process during authorisation, and it starts monitoring resource consumption. A job submitted via the modified Globus job manager process is acknowledged by writing the job request to the job requests file. Each job is allocated a sequence number. Globus job manager was modified to simply write the job's identification number (used internally by the job manager), the date and time when it was submitted, and the sequence number to the temporary requests file. The other modification to the job manager consists of code that calls a resource usage function (e.g. rusage(…) on Unix-based platforms) provided by the operating system to extract resource consumption of the job. Since the job manager of the Globus Toolkit version 2.0 forks a child process for the job, the resource usage is collected for the child process.

When the job completes, the GRAM job manager writes the job's resource consumption to the appropriate RUR file (corresponding to the sequence number), the Grid Resource Meter reads the file, converts the data into the XML-based Grid Resource Usage Record format, and forwards the RUR to the Charge Calculator module. The Charge Calculator uses the service rates to compute individual resource charges and the total, appends them to the RUR, and sends the record to the GridBank server for storage and subsequent audit. When all requests have been processed and resource usage has been reported to GridBank, the requests file and all resource usage files are deleted.

The Resource Usage Record must be supplied to GridBank regardless of the payment type (i.e. direct or deferred). The record presents a means of non-repudiation and serves as the usage and financial record for accounting purposes. Deferred payment type forces GSPs to produce the record because they cannot redeem a payment otherwise.

Therefore direct payment should be used only when the service providers are trusted to produce the RUR.

## 4.5 GridBank Branches

The inter-branch protocols are also divided into the two categories: direct and deferred. The direct inter-branch payment protocol assumes that the two branches authorise each other and operate using some common currency. As long as the exchange rate is fixed, the branches can operate using different actual currencies. A floating exchange rate for actual money poses a problem for direct payments because an actual exchange rate can fluctuate in such a way that creates a loss for GridBank (as described in section 3.8.1). The implication of such a methodology is that the direct payment model essentially moves currencies between branches as opposed to mere accounting in the case of the deferred payment mode (i.e. currencies are actual and moved by actual banks). The initiating service consumer's branch has to contact the service provider's branch in the direct inter-branch payment mode.

The deferred payment protocols include GridHash and GridCheque. The assumption is that every resource at the foreign Grid site authorises the local GridBank server. The local initiating branch creates a deferred payment transaction and invokes authorisation call directed to the resource. There must be at least one common Certificate Authority between the sites. The deferred payment model also enforces collection of resource usage records. Deferred payments can be redeemed only by presenting the usage record. Inter-branch reconciliation procedure involves bulk inter-branch transactions processed during off-peak times decided by the administrators. Reconciliation between branches involves recording of actual usage and produces statements for administrators to reconcile with the actual bank.

### 4.5.1 Branch Authorisation

Before any inter-branch protocols can proceed, a branch must explicitly authorise another branch if both sites intend to share resources. By default a branch revokes access rights of all other branches. Only a legitimate GridBank administrator can authorise another branch by entering the branch's details using GridBank Service Consumer Client's GUI.

Using the GridBank Service Consumer Client module an administrator invokes a method (appendix B.1.13) of the GridBank server API that enters the foreign branch's details into the local branch's database (table "authorised_banks"; appendix A.5). The details include the bank-branch number, the branch's certificate subject name, the URL of the server, the foreign currency name, the local to foreign currency exchange rate, and the foreign currency balance. The bank-branch number is used for payment routing and there is a one-to-one mapping between the bank-branch number and the GridBank server URL. The foreign branch's certificate subject name is used for authentication and authorisation when the foreign branch contacts the local branch. The exchange rate is negotiated by the administrators and, if common currency is used (e.g. Grid$), the exchange rate is assigned to 1.0.

The foreign currency balance is a safeguard against unlimited resource consumption at the foreign site. This balance represents a rough resource allocation (in terms of money) at the foreign site. Administrators of two branches decide on the exchange rate, and each administrator assigns the foreign currency balance such that both balances are equal via the exchange rate. For example, branch A operates on Grid$, branch B on AU$, and the exchange rate is fixed at 0.5 (i.e. 1 Grid$ = 0.5 AU$). Branch A sets the balance at 500 AU$ (i.e. AU$500 credit at branch B), and branch B sets its balance at 1000 Grid$ (i.e. Grid$1000 credit at branch A).

Such an arrangement essentially ensures that the branches do not owe each other money until the next reconciliation. Depending on how many resources were consumed at both sites by the foreign consumers, each branch calculates an outstanding amount by subtracting the balance of the other branch from its own balance (converted via exchange rate into local currency). If the amount is negative, then the local branch owes the amount. Vice versa, if the amount is positive, the foreign branch owes the amount.

### 4.5.2 Inter-Branch Direct Payment

The inter-branch direct payment protocol is illustrated in figure 4.7.

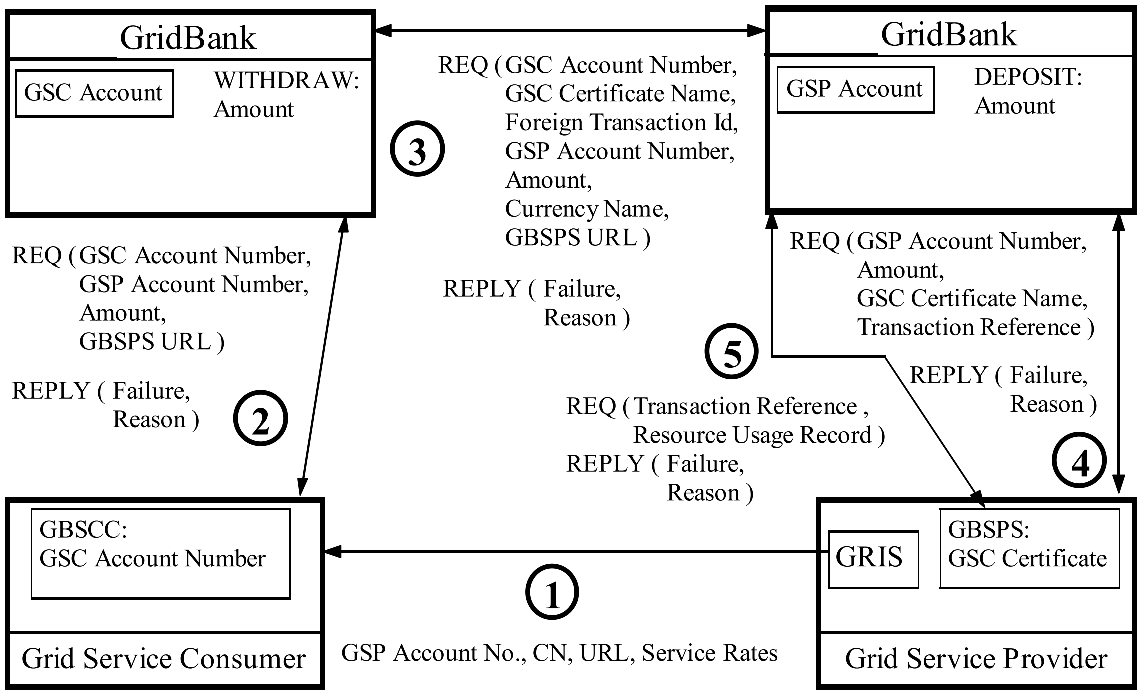

GridBank
GSC Account
WITHDRAW: Amount
GridBank
GSP Account
DEPOSIT: Amount
REQ (GSC Account Number, GSC Certificate Name, Foreign Transaction Id, GSP Account Number, Amount, Currency Name, GBSPS URL )
REPLY ( Failure, Reason )
3
REQ (GSC Account Number, GSP Account Number, Amount, GBSPS URL )
REPLY ( Failure, Reason )
2
REQ (GSP Account Number, Amount, GSC Certificate Name, Transaction Reference )
REPLY ( Failure, Reason )
4
5
REQ (Transaction Reference , Resource Usage Record )
REPLY ( Failure, Reason )
GBSCC: GSC Account Number
Grid Service Consumer
1
GSP Account No., CN, URL, Service Rates
GRIS
GBSPS: GSC Certificate
Grid Service Provider

**Figure 4.7: Inter-Branch Direct Payment**

Steps 1 is as described in the DirectPayment protocol. Step 2 involves invoking the same method (appendix B.1.2), but the protocol determines whether the GSP account number is from another branch. Provided that the foreign branch is authorised, GridBank withdraws the requested amount in local currency from the GSC's account, converts the amount into the foreign currency via the exchange rate, and in step 3 invokes the inter-branch direct payment method (appendix B.1.10), which succeeds only if the certificate subject name of the calling branch matches the one recorded in the database of the serving (foreign) branch. The currency name must be that of the foreign branch. The foreign transaction reference is the transaction number from the calling branch. Steps 4 and 5 are identical to the DirectPayment described in section 4.4.1.

### 4.5.3 Inter-Branch Deferred Payment

The GridHash and GridCheque protocols both use the deferred payment model. Requesting a hash chain or a cheque for a GSP from another site is identical to step 2 in sections 4.4.2 and 4.4.3 respectively. The local initiating branch directly contacts the foreign resource and authorises the access. The resource at the foreign site must recognise the certificate authority of the local site. The authorisation call is also identical to step 3 in sections 4.4.2 and 4.4.3. However, the transaction details are different.

Between the step 2 and the step 3 (figures 4.5 and 4.6) GridBank ensures there are sufficient funds in the GSC's account, withdraws the requested maximum amount from the available balance and deposits it into the "lock" balance, implying that the money remains at the local branch at this stage. The transaction is created in the "inter_branch_trans" table (appendix A.10) with the (local) transaction identification number, the entry timestamp, the expiry date, the GSC's account number and certificate name, the GSP's account number and certificate name, the local currency name, the local amount, which is the requested maximum amount, the GridBank Exchange Rate, which is the Grid Accounting Server Exchange Rate (section 3.8.1), the foreign currency name, the foreign currency amount calculated by multiplying the local amount by the GridBank Exchange Rate, the number of payments (e.g. one for GridCheque), and the reconciled field is assigned to false (i.e. 0). The foreign transaction identification is assigned to null, as it is unknown until the reconciliation between the two branches.

The amount in the authorisation call in step 3 (figures 4.5 and 4.6) specifies the budget in foreign currency. When the computation is completed, the GSP at the foreign site transfers the resource usage record to the foreign branch by sending a 3-tuple containing the transaction reference from the local branch, the last hash token received, which is null in the GridCheque protocol, and the resource usage record. The foreign GridBank branch creates the inter-branch transaction entry with the foreign transaction reference number, which will be entered into the "ForeignTransID" field ("inter_branch_trans" table; appendix A.10) at the local branch, the timestamp of the entry, the GSC's account number and the certificate name, the GSP's account number and the certificate name, the currency (i.e. the foreign branch currency name), the actual amount (in foreign currency), and the "Reconciled" value assigned to false. The rest of the fields are null, but if the protocol is GridHash, then the value of the last hash token received is assigned to the "LastHash" field.

The inter-branch transactions reconciliation is initiated manually by a GridBank administrator from either branch. Using the GridBank Service Consumer Client module, an administrator specifies the bank-branch number. GridBank searches the database for all inter-branch transactions that have the specified bank-branch number and are not reconciled (i.e. “Reconciled” field is set to false). Once those transactions are retrieved, GridBank constructs an XML document listing all transactions and the available details within individual transaction (appendix D.4.1). It invokes the reconciliation method (appendix B.1.11) using the URL of the branch, which is retrieved from the “authorised_bank” table (appendix A.5).

Upon receiving the inter-branch transactions document the foreign branch ensures that the calling branch is authorised (i.e. entry in “authorised_banks” table; appendix A.5). For each tranaction it completes the “ForeignTransID” field (table “inter_branch_trans”; appendix A.10), and sends back the transaction reference number of this branch (to be entered into the “ForeignTransID” field at the other branch), and the actual amount, which must be less than or equal to the maximum amount. The calling branch receives an XML document listing only the transaction numbers (local and foreign) and the actual amounts (appendix D.4.2). The calling branch replaces the maximum foreign amount with the actual total calculated via the exchange rate, subtracts the maximum local amount from the locked balance, calculates the difference between the maximum and the actual local amounts and deposits the difference into the available balance. The “Reconciled” fields of each transaction, at the local and at the foreign branches, are assigned to true. Both branches generate reconciliation reports that state how much actual currency each branch owes to the other.

## 4.6 GridBank Applications

GridBank’s primary objective is to provide Grid resource access authorisation and accounting. In a Virtual Organisation there would be a GridBank branch present in which all VO users have accounts and credits associated with each account. The Grid accounts (i.e. X.509v3 digital certificate subject names) would be linked to GridBank accounts. Therefore, the twin problems of authentication (account entry in GridBank) and authorisation (enough account credit in GridBank for a resource provider) are solved.

The GridBank system can be used to restrict access to resources that are otherwise heavily subscribed. For example, in the High Energy Physics Grid [BADG03] at the department of Physics, Melbourne University, the data files are stored on different servers. A storage server (data host) might be better than the others due to superior network connectivity or due to the presence of most requested data files of common interest to the community. It is possible that this storage server may be heavily overloaded. In order to reduce the demand on the resource and reduce congestion, the cost of accessing it is raised. Only those users with serious intent and a high enough balance with the GridBank will be able to access it. This is an example of how GridBank can be used to enforce economic principles. Therefore, before any data request operation, the GridBus scheduler (section 3.2) will check GridBank for user account balance.

Based on prediction of resource prices for a certain time period, the resource broker and scheduler can issue advanced reservation requests for resources against the balance in the GridBank account.

In general, GridBank can be established at a VO and a GridBank Service Provider Server installed on every resource that is contributed to the Grid. Service consumers can then open GridBank accounts by mailing an actual cheque to a GridBank administrator who would create an account, deposit the amount, and email back the account number, which is entered into the configuration file of the GridBank Service Consumer Client module.

## 4.7 Summary

The GridBank system is an implementation of the Grid Accounting Services Architecture. It allows a GSC to access a GSP located anywhere in the world by providing a payment. Once a computation is completed, the Resource Usage Record [BB03] is produced by the service provider and stored by GridBank. GSCs and GSPs conduct reliable audits by requesting GridBank statements listing transactions that have their identities (i.e. certificate subject names). GridBank administrators have access to all transactions for the purpose of estimating future hardware and software requirements.

The continuing trend towards Web services implementation of the OGSA motivated the use of XML and SOAP technologies in the construction of the GridBank system. Building on top of the Grid Security Infrastructure provided by the Globus Toolkit preserves Grid's single sign-on policy. The GSS-API implemented by the SSL protocol with PKI for session key exchange ensures that GridBank is secure and preserves accounting data integrity.

Two payment modes are provided: direct and deferred. Direct mode implies that the funds are transferred immediately into the GSP's account. Such a method is suitable for service providers with minimal trust in service consumers. The disadvantage of such a method is inefficiency of inter-branch transactions – the local branch has to contact the foreign branch in real-time in order to authorise resource access. Deferred payment mode avoids "on-line" transactions by contacting the foreign GSP directly. Inter-branch transactions are reconciled later during off-peak periods.

GridBank provides the DirectPayment protocol that operates in the direct mode. The protocol is suitable for a GSP that has a fixed (subscription) fee. A GSC requests a payment, GridBank transfers requested amount into the GSP's account, contacts and notifies the GSP of the amount. The GSC then submits the job. DirectPayment handles infra-branch and inter-branch transactions.

There are two protocols that operate in the deferred mode: GridHash and GridCheque. GridHash is based on a micropayment scheme and uses a hash token chain to pay in increments. A GSC requests a payment. GridBank generates a hash chain, which is forwarded to the GSC, contacts and notifies the GSP of the root hash and the amount assigned to each hash token. The GSC releases a token every payment period. The GridCheque protocol is similar to DirectPayment. A GSC contacts GridBank, which reserves the amount and notifies the GSP. To redeem the payment, a GSP must present the Resource Usage Record [BB03] to GridBank.

The GridBank system is expected to regulate the supply for and demand of Grid services. When a resource is overloaded, its price is raised. When it is idle, the price is lowered. Economic load balancing ensures that the World Wide Grid is utilised efficiently without enforcing any complex scheduling policies. The scheduling policy is simple: the one who pays more is assigned a higher priority. The next chapter summarises and concludes the contribution to Grid Economics by the Grid Accounting Services Architecture and the GridBank system.

# CHAPTER 5

# Conclusions and Future Directions

## 5.1 Summary

With the advent of Internet the researchers around the world realised that “the network is the computer”. In the mid-nineties the term Grid was formulated to mean exactly that – the network and compute farm infrastructure for the next generation supercomputing. In the World Wide Grid environment, computing resources are heterogenous and geographically distributed, as well as logically across multiple administrative domains with various usage and cost policies. Application data and applications themselves are disseminated around the world and typically reside on massive tertiary storage facilities. Resource access control and authorisation remain the key obstacles of achieving a global meta-computer.

To overcome this limitation we have designed and developed a system that achieves the following:

Allows World Wide Grid resource access authorisation through payment provision,

GSPs have the ultimate control of their resources; they can disable GridBank’s authorisation control and they can maintain their own policies by specifying local (template) account permissions,

GSCs, GSPs and GridBank administrators can conduct reliable accountability audits by requesting statements from GridBank,

Audits can be used by Grid resource brokers and schedulers to assess application performance and adapt to availability and cost of resources dynamically,

Offers monetary incentives to resource owners to contribute their resources to Grids, and

Prevents resource users from unnecessarily overloading popular resources.

## 5.2 Conclusion

World Wide Grid resource management and allocation is a complex task. The resources vary dynamically in availability, cost and usage policies. The Grid Architecture for Computational Economy (GRACE) [BAG98, B02a] proposed an economy-based approach to management of Grid resources. In the conclusion, the thesis [B02a] suggested an accounting infrastructure that would enable real-time resource usage collection and payment. This thesis extended the original concept of GridBank to also provide secure centralised resource access authorisation similar to the Community Authorisation Service [PWFK02]. Whereas CAS is adopted by co-operating communities, GridBank is expected to also operate in competitive environments.

The Grid Accounting Services Architecture extends GRACE by providing secure authorisation, payment and accounting services, which are scalable to be applied to the World Wide Grid. The architecture formally defines resource usage metrics analogous to the metrics used by electricity Grids – not only is the computational power metered, but also the time that power was supplied. It also defines the standard Resource Usage Record, which contains common computational service metrics to be used at the Grid middleware level.

GASA introduces new components as well as incorporating the previously defined ones, such as the Grid Market Directory and the Grid Trade Server. The Resource Access Authorisation module ensures that only those clients who have paid are authorised. Grid Resource Meter interfaces with heterogenous resources and produces the Resource Usage Record. The record is forwarded to the Grid Service Charge Calculator responsible for computing the total charge based on the Resource Usage Record and the service rates obtained from the Grid Trade Server. The RUR, updated with the service charge, is forwarded to the Accounting Server for storage and subsequent audit. Using the GSC Payment and Administration module, the clients are capable of accessing Grid resources by providing payment and assessing application performance, the resource owners can update service rates, and the administrators can estimate future hardware and software requirements based on the overall utilisation of the Grid site.

GridBank is an implementation of GASA that works in conjunction with the Globus Toolkit version 2. It simplifies management of computational Grids by providing a centralised resource access authorisation. Resource owners retain ultimate control of their resources and can decide to whom they grant the access on individual basis by modifying the access authorisation file on each resource they own. However, in a typical Grid GridBank accounts can be used not only to relieve resource owners from the burden of managing thousands of clients, but also to record resource usage and cost automatically against the accounts. Such information can be easily shared between Grid sites as it is in the common XML-based format.

GASA defines and the GridBank system implements the three payment protocols. These are classified as the two payment modes:

Direct payment, and
Deferred payment.

The direct payment method can be compared with cash in the real world and is implemented by the GridBank protocol named DirectPayment (pre-pay methodology). The deferred payment protocols are GridHash (pay-as-you-go), which is based on hash chains, and GridCheque (pay-after), which can be compared with cheques in the real world. The problem with deferred payments is that not every service provider would agree to sell its services for a postponed payment. Most actual retailers prefer cash and give discounts to those customers who pay cash. This is the reason for providing the DirectPayment protocol. However, if the service cost is unknown beforehand (i.e. if it depends on the application processing time), then the GridHash or the GridCheque protocols should be used. GridHash offers incremental payments, but is not as secure as GridCheque.

The realisation of GASA through the GridBank system, by utilising Globus Toolkit technologies and providing authorisation, payment, accounting and audit services for Grid computing, ensures a consistent resource access authorisation that is recorded against GridBank accounts and provides non-refutable information about the participants in the transaction as well as the details of resource consumption.

Using actual or virtual currencies allows organisations to lease services on demand instead of purchasing and maintaining expensive computing facilities. The GridBank system can be integrated with other access control systems, such as CAS, as well as any Grid application tools such as resource brokers and schedulers (e.g. Nimrod-G).

## 5.3 Future Research

This dissertation presented an authorisation, payment and accounting framework for service-oriented economy-based Grid computing. The GridBank system was implemented in C/C++ programming language. Given the current trend towards a Web services implementation of Grid services and the fact that the Globus Toolkit version 3 is implemented in Java programming language, the GridBank system should be re-implemented in Java to achieve platform independence and also leverage recent advances in Grid services.

The Resource Usage Record was developed for a computational service in the context of the GRAM protocol as part of the Globus Toolkit. A number of payment protocols that incorporate resource usage accounting have been developed to alleviate existing accounting systems developed by independent Grid sites. However, there are other areas in computational accounting that do require further development. Data storage accounting is one of them.

Due to time constraints the payment policy outlined in section 3.5.3.1 was not implemented in the GridBank system. Such a policy remains an important issue in handling authorisation and job execution exceptions and should be implemented in the future.

### 5.3.1 Generic Grid Resource Usage Record Format

In appendix D.1 a generalised Resource Usage Record for a computational service is presented, which is used by the GridBank system to exchange accounting information between Grid sites and to reliably conduct accountability audits. This record is only useful for a general computational service, and can be extended to include other site-specific information such as the number of times a job was migrated from one cluster node to another. However, there is a need for a Resource Usage Record for a data storage service, but more importantly, for any specialised (e.g. instrument) Grid service. Open Grid Services Architecture [FKNT02] and Grid Service Specification [TCFF02] outline the concept of service data – the information that describes what the service does (e.g. CPU frequency, size of main memory, network interface bandwidth).

A generic Grid Resource Usage Record can be formulated based on the service data for any kind of Grid hardware and software device. The GridBank system can be improved by developing a generic Grid Resource Meter that takes each attribute of the service data and records the corresponding resource consumption in units that apply to that attribute. For example, a digital telescope might have a service data element called “Resolution”, which is measured in gigapixels. GridBank can assemble the XML-based RUR by creating an element named “Resolution”, and create a sub-element “Units” that would have the text element “Gigapixels” and another sub-element “NumberOfUnits” that would have a positive double value indicating the number of gigapixels. I.e.:

```
<Resolution>
        <Units>
                Gigapixels
        </Units>
        <NumberOfUnits>
                1.45
        </NumberOfUnits>
</Resolution>
```

### 5.3.2 GridBank Payment Policy

As mentioned in the conclusion, the GridBank payment policy presented in section 3.5.3.1 was not implemented in the GridBank system. In order to introduce such policy into the Grid environment, the Grid and the GridBank system must be mature. The Grid must guarantee the promised Quality-of-Service requested by a GSC. In order to guarantee Grid QoS in a wide-area network, the Internet protocols must support QoS. This is not possible with the current IPv4 routing protocol as it has best-effort delivery policy. It is only when the much anticipated IPv6 is introduced GSPs would be able to guarantee the requested QoS.

Since Grid computing contemplates the use of public Internet to deliver Grid services, it is not practical to design and implement a payment policy. When a new routing protocol (e.g. IPv6) replaces IPv4, it will provide guaranteed bandwidth from a service consumer to a service provider. It is then the GridBank payment policy should be developed in case a service provider fails to deliver the promised service (e.g. the server crashes), and GridBank can automatically play the role of a mediator and an arbitrator.

Another scenario that has to be considered is when two GridBank branches are in conflict. For example, one branch might refuse to reconcile deferred transactions and the administrator does not settle accounts with the actual bank. Such a situation has to be resolved by humans, but the GridBank system should be able to produce non-disputable evidence of all transactions that can be confirmed with involved consumers and providers. This requires all service consumers and providers to digitally sign each transaction, which consists of the Resource Usage Record and other details. The digital signatures must be stored in GridBank database for later proof. At the moment GridBank uses Globus I/O library, which only provides non-repudiation at the time of entry.

### 5.3.3 Grid (Wide-Area) Load Balancing

Each Grid site inevitably has its own usage policies and pricing algorithms. Although each site will dynamically react to supply and demand, a generic pricing algorithm can be adopted by all sites. Such an algorithm perhaps should have site-specific policies as an input and have a resource administrator privileges such that it can adjust service rates automatically through the GridBank Service Consumer Client module (i.e. Payment and Administration module). The site-specific policy would include pricing thresholds, which govern whether the price should be lowered or raised. The thresholds can be in terms of currency amounts and current demand. The currency is easily measured, but the measure of demand can vary. Some measures of demand are:

Number of clients (requests) per unit time,
Percentage of CPU engaged for outside clients,
Percentage of secondary (tertiary) storage used, and
Percentage of network capacities used by outside clients.

The outside clients are those who are external to the organisation and who share the Grid resources through the GridBank system. It is expected that pricing algorithms will take into account the current demand for services by the users who are part of the organisation, which owns the site's resources. In such manner, the pricing algorithms will be able to achieve wide-area Grid load balancing similar to electricity Grids, which are managed by people.

### 5.3.4 GridBank Service Provider Reputation

A number of Web-based trade and Peer-to-Peer systems, such as e-Bay, KaZaA, eDonkey and others (e.g. [HLMS03]), have introduced the concept of service provider reputation. Another area of improvement in the GridBank system is to introduce the concept GSP reputation. Those providers who deliver exceptionally excellent quality of service will score highest points. Those who consistently crash or provide inconsistent quality of service are complained about and will score lowest points. This would effectively influence the kind of payment policy that GSCs will select. Again, this process can be automated to decouple Grid application developers from such low-level details and instead concentrate on the application itself.

### 5.3.5 Grid Resource Reservation and Allocation

Another system that can be developed and used in conjunction with GridBank is an economy-based advanced reservation and allocation scheme. The Globus Architecture for Reservation and Allocation (GARA) [FKLL99] and a QoS Architecture that combines Resource Reservation and Application Adaptation [FR00] are the few initiatives that have addressed this issue. However, similar to the Globus Toolkit's approach to Grid computing, they do not take an economic approach to management of Grid resources. Rather, they are designed for collaborations of scientists or scientific government agencies. In the context of GRACE and GASA an advanced reservation system would pre-allocate resources by making a payment through GridBank and presenting the payment token as a guarantee of an allocation.

# References

[ASGH95] Abramson, D., Sosic, R., Giddy, J., Hall, B. *Nimrod: A Tool for Performing Parameterised Simulations using Distributed Workstations.* Proceedings of the 4th IEEE Symposium on High Performance Distributed Computing (HPDC'95), August 2-4, 1995, Washington D.C., USA.

[BADG03] Belle Analysis Data Grid - http://roberts.ph.unimelb.edu.au/epp/grid/ (22/09/2003)

[BAG98] Buyya, R., Abramson, D., Giddy, J. *A Case for Economy Grid Architecture for Service Oriented Grid Computing*. Proceedings of IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18, 1998.

[BAG00] Buyya, R., Abramson, D., Giddy, J. *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*. Proceedings of the HPC ASIA '2000, the 4th International Conference on High Performance Computing in Asia-Pacific Region. Beijing, China, IEEE Computer Society Press, pp. 283-290, USA, 2000.

[BB03] Barmouta, A., Buyya, R., *GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration*, Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS2003), Workshop on Internet Computing and E-Commerce (ICEC2003), pg. 245, April 22-26, 2003, Nice, France.

[BFKT99] Bester, J., Foster, I., Kesselman, C., Tedesco, J., Tuecke, S. *GASS: A Data Movement and Access Service for Wide Area Computing Systems.* Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems (IOPADS'99), pp. 78-88, 1999, Atlanta, Georgia, USA.

[BFPT00] Brooke, J., Foster, M., Pickles, S., Taylor, K., Hewitt, T. *Mini-Grids: Effective Test-beds for GRID Application.* Proceedings of the 7th International Conference on High Performance Computing (HiPC2000), Workshop on Grid Computing (GRID2000), December 17-20, 2000, Bangalore, India.

[BGHH00] Bellare, M., Garay, J., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Tsudik, G., Van Herreweghen, E., Waidner, M. *Design, Implementation and Deployment of a Secure Account-Based Electronic Payment System.* IEEE Journal on Selected Areas in Communications, 2000. http://citeseer.nj.nec.com/99445.html (22/09/2000)

[BGJY98] Bellare, M., Garay, J., Jutla, C., Yung, M. *VarietyCash: a Multi-purpose Electronic Payment System.* Proceedings of the 3rd USENIX Workshop on Electronic Commerce, pp. 9-25, Boston, USA. August 31 – September 3, 1998.

[BIK01] Blaze, M., Ioannidis, J., Keromytis, A. *Offline Micropayments without Trusted Hardware.* Financial Cryptography 2001. Grand Cayman, February 2001.

[BMBB02] Burq, S., Melnikoff, S., Branson, K., Buyya, R. *Visual Parametric Modeller for Rapid Composition of Parameter-Sweep Applications for Processing on Global Grids.* Technical Report, 2002. GRIDS Laboratory, the University of Melbourne, Australia. http://www.gridbus.org (22/09/2003)

[B00] Black, J., *Message Authentication Codes*. PhD Thesis, University of California, USA, September, 2000.

[B02a] Buyya, R., *Economic-based Distributed Resource Management and Scheduling for Grid Computing*, PhD Thesis, Monash University, Melbourne, Australia, April 12, 2002.

[B02b] Buyya, R., *Grid Economy Comes of Age: Gridbus Technologies for Service-Oriented Cluster and Grid Computing.* Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing (P2P 2002), September 5-7, 2002, Linkoping, Sweden.

[BV01] Buyya, R., Vazhkudai, S. *Compute Power Market: Towards a Market-Oriented Grid.* Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2001), May 16-18, 2001, Brisbane, Queensland, Australia.

[BW03] Brumen, B., Welzer, T. *Internet Commerce Authorities and Digital Cash*. http://www.ggf1.nl/abstracts/ACCT/VUS.pdf (22/09/2003)

[CB02] Chetty, M., Buyya, R. *Weaving Computational Grids: How Analogous Are They with Electrical Grids?*. Computing in Science and Engineering (CiSE), ISSN 1521-9615, Volume 4, Issue 4, Pages: 61-71, IEEE Computer Society Press and American Institute of Physics, USA, July-August 2002.

[CCOT03] Cannon, S., Chan, S., Olson, C., Tull, C., Welch, V., Pearlman, L., *Using CAS to Manage Role-Based VO Sub-Groups.* Proceedings of the 2003 Conference for Computing in High-Energy and Nuclear Physics (CHEP03), March 24-28, 2003, La Jolla, California, USA.

[CD97] Casanova, H., Dongarra, J., *NetSolve: A Network-enabled Server for Solving Computational Science Problems*. The International Journal of Supercomputer Applications and High Performance Computing, Volume 11, number 3, pp. 212-223, Autumn 1997.

[CDK01] Coulouris, G., Dollimore, J., Kindberg, T., *Distributed Systems. Concepts and Design*. Third Edition, ISBN 0-201-61918-0, Addison-Wesley Publishers Ltd., Pearson Education Ltd., 2001.

[CE98] Czajkowski, G., Von Eicken, T. *Jres: A Resource Accounting Interface for Java.* Proceedings of the 1998 ACM OOPSLA Conference, October 1998, Vancouver, Canada.

[CFFK01] Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C. *Grid Information Services for Distributed Resource Sharing*. Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), August 7-9, 2001, San Francisco, California, USA.

[CKKG99] Chapin, S., Katramatos, D., Karpovich, J., Grimshaw, A. *The Legion Resource Management System*. Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 16, 1999, San Juan, Puerto Rico.

[CSHM02] Crompton, P., Swann, M., Hopkins, S., McEachern, W. *Macroeconomics: A Contemporary Introduction.* ISBN 0-170-10460-5, Nelson Australia Pty Limited trading as Nelson Thomson Learning, 2002.

[DB02] Ding, C., Buyya, R. *Guided Google: A Meta Search Engine and its Implementation using the Google Distributed Web Services.* Technical Report, 2002. GRIDS Laboratory, the University of Melbourne, Australia. http://www.gridbus.org (22/09/2003)

[DL99] Dai, X., Lo, B. *Netpay – an Efficient Protocol for Micropayments on the WWW,* Technical Report, 1999. Southern Cross University, Lismore, Australia. http://ausweb.scu.edu.au/aw99/papers/dai/paper.html (22/09/2003).

[FFW98] Feghhi, J., Feghhi, J., Williams, P. *Digital Certificates: Applied Internet Security.* ISBN 0-201-30980-7, Addison Wesley Longman Inc., 1998.

[FK97] Foster, I., Kesselman, C. *Globus: A Metacomputing Infrastructure Toolkit.* International Journal of Supercomputer Applications, 11(2): 115-128, 1997.

[FK98a] Foster, I., Karonis, N. *A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems.* Proceedings of High Performance Networking and Computing Conference (SC'98), November 7-13, 1998, Orlando, Florida, USA.

[FK98b] Foster, I., Kesselman, C. *The Globus Project: A Status Report.* Proceedings of IPPS/SPDP '98 Heterogenous Computing Workshop, pp. 4-18, 1998.

[FK99] Foster, I., Kesselman, C. (editors), *The Grid: Blueprint for a New Computing Infrastructure*, ISBN 1-55860-475-8, Morgan Kaufmann Publishers, Inc., 1999.

[FKLL99] Foster, I., Kesselman, C., Lee, C., Lindell, B. *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation.* Proceedings of the 7th International Workshop on Quality of Service, pp. 27-36, June 1-4, 1999, London, UK.

[FKNT02] Foster, I., Kesselman, C., Nick, J., Tuecke, S., *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, Open Grid Services Infrastructure Working Group, Global Grid Forum, June 22, 2002. http://www.globus.org/research/papers.html

[FKT01] Foster, I., Kesselman, C., Tuecke, S., *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International Journal of Supercomputer Applications, 15(3), Sage Publications. 2001.

[FKTT98] Foster, I., Kesselman, C., Tsudik, G., Tuecke, S., *A Security Architecture for Computational Grids*, Proceedings of the 5th ACM Conference on Computer and Communication Security, pp. 83-92, 1998.

[FL98] Foster, I., Von Laszewski, G., *Usage of LDAP in Globus*, Open Grid Services Infrastructure Working Group, Globus Grid Forum, April 21, 1998. http://www.globus.org/research/papers.html

[FR00] Foster, I., Roy, A. *A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation.* Proceedings of the 8th International Workshop on Quality of Service (IWQoS'00), June 5-7, 2000, Pittsburgh, Pennsylvania, USA.

[FTLF01] Frey, J., Tannenbaum, T., Livny, I., Foster, I., Tuecke, S., *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* Proceedings of the 10th International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE CS Press, August 2001.

[GB03] Gibbins, H., Buyya, R. *Gridscape: A Tool for the Creation of Interactive and Dynamic Grid Testbed Web Portals.* Technical Report, July 2003, GRIDS Laboratory, The University of Melbourne, Melbourne, Victoria, Australia. http://arxiv.org/ftp/cs/papers/0307/0307052.pdf (22/09/2003)

[GBP02] GridBus Project, 2002. GRId computing and Distributed Systems (GRIDS) Laboratory, the University of Melbourne. http://www.gridbus.org

[GESA03] Grid Economic Services Architecture, Working Group, http://www.doc.ic.ac.uk/~sjn5/GGF/gesa-wg.html (22/09/2003).

[GMAG95] Glassman, S., Manasse, M., Abadi, M., Gauthier, P., Sobalvarro, P. *The Millicent Protocol for Inexpensive Electronic Commerce.* World Wide Web Journal, p. 89, 1(1), December 1995.

[GS96] Gabber, E., Silberschatz, A. *Agora: A Minimal Distributed Protocol for Electronic Commerce.* Proceedings of the 2nd USENIX Workshop on Electronic Commerce, pp. 223-232, November 18-21, 1996.

[HA01] Hacker, T., Athey, B., *Account Allocations on the Grid*, 1st Global Grid Forum (GGF1) Working Draft, 4-9 March, 2001, Amsterdam, Netherlands.

[H03] Hallam-Baker, P. *Micro Payment Transfer Protocol (MPTP) Version 0.1.* http://www.w3.org/TR/WD-mptp (22/09/2003)

[HBY01] Hazlewood, V., Bean, R., Yoshimoto, K. *SNUPI: A Grid Accounting and Performance System employing Portal Services and RDBMS Back-end.* Proceedings of the 2001 Linux Revolution Conference, June 26-27, 2001, Urbana, Illinois, USA.

[HCNC99] Hitt, M., Clifford, P., Nixon, R., Coyne, K., *Dynamic Strategic Resources: Development, Diffusion and Integration.* The strategic management series, ISBN 0-471-62533-7, Wiley Publishers, 1999.

[HKFG00] Humphrey, M., Knabe, F., Ferrari, A., Grimshaw, A. *Accountability and Control of Process Creation in Metasystems.* Proceedings of the 2000 Network and Distributed System Security Symposium (HDSS2000), February 3-4, 2000, San Diego, California, USA.

[HLMS03] Hausheer, D., Liebau, N., Mauthe, A., Steinmetz, R., Stiller, B. *Token-based Accounting and Distributed Pricing to Introduce Market Mechanisms in a Peer-to-Peer File Sharing Scenario.* Proceedings of the 3rd IEEE International Conference on Peer-to-Peer Computing, September 1-3, 2003, Linkoping, Sweden.

[ISO3166] International Organisation for Standardization, *ISO3166 English Country Names and Code Elements.* http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html (22/09/2003).

[J03] Jackson, S. *Qbank 2.8: A CPU Allocations Bank.* http://www.emsl.pnl.gov:2080/docs/mscf/qbank-2.8 (22/09/2003)

[KAZA03] KaZaA file sharing application. http://www.kazaa.com (22/09/2003)

[KMW01] Kupczyk, M., Meyer, N., Wolniewicz, P. *Simplifying Administration and Management Processes in the Polish National Cluster.* 1st Global Grid Forum (GGF1), 4-9 March, 2001, Amsterdam, Netherlands.

[LBRV04] Luther, A., Buyya, R., Ranjan, R., Venugopal, S. *Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids.* Technical Report, 2004. GRIDS Laboratory, the University of Melbourne, Australia. http://www.gridbus.org/papers/Alchemi.pdf (22/04/2004)

[LSF03] Load Sharing Facility, http://accl.grc.nasa.gov/lsf/ (22/09/2003)

[M01] McGinnis, L. *Resource Accounting – Current Practices.* 1st Global Grid Forum (GGF1) Working Draft, 4-9 March, 2001, Amsterdam, Netherlands.

[MD98] Miller, M., Drexler, K. *Markets and Computation: Agoric Open Systems.* In "The Ecology of Computation" by B. Huberman (editor), pp. 133-176, Elsevier Science Publishers. The Netherlands, 1998.

[MN93] Medvinsky, G., Neuman, B. *NetCash: A design for practical electronic currency on the Internet.* Proceedings of the 1st ACM Conference on Computer and Communication Security, pp. 102-106, Fairfax, Virginia, USA. November 1993.

[MN95] Medvinsky, G., Neuman, B. *Requirements for Network Payment: The NetCheque Perspective.* Proceedings of IEEE COMPCON'95. San Francisco, USA. March 1995.

[MR02] MICALI, S., RIVEST, R. *Micropayments Revisited.* Proceedings of the Cryptographer's Track at the RSA Conference 2002, Bart Preneel (ed.), Springer Verlag CT-RSA 2002, LNCS 2271, pp. 149-163.

[MYOB03] MYOB GROUP, *MYOB Accounting Package.* http://www.myob.com/ (22/9/2003)

[O00] Odlyzko, A. *The Case Against Micropayments.* O'Reilly Network Press Release, 19th of December 2000. http://www.openp2p.com/pub/a/p2p/2000/12/19/micropayments.html (22/09/2003).

[PP03] PayPal Payment System, http://www.paypal.com/ (22/09/2003)

[PB03] Placek, M., Buyya, R. *G-Monitor: A Web Portal for Monitoring and Steering Application Execution on Global Grids.* Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments, 21st of June 2003, Seattle, Washington, USA.

[PBS03] Veridian Systems, *OpenPBS v2.3: The Portable Batch System Software*, Veridian Systems, Inc., Mountain View, CA, September 2000. http://www.openpbs.org/ (22/09/2003)

[P00] Peirce, M. *Multi-Party Electronic Payments for Mobile Communications.* PhD Thesis, University of Dublin, Trinity College, Dublin, Ireland. October 31, 2000.

[PJ97] Pagnia, H., Jansen, R. *Towards Multiple-payment Schemes for Digital Money.* Lecture notes in Computer Science (LNCS), Volume 1318, ISBN 3-540-63594-7, Springer, 1997.

[PKWF03] Pearlman, L., Kesselman, C., Welch, V., Foster, I., Tuecke, S. *The Community Authorization Service: Status and Future.* Proceedings of the 2003 Conference for Computing in High-Energy and Nuclear Physics (CHEP03), March 24-28, 2003, La Jolla, California, USA.

[PO95] Peirce, M., O'Mahony, D. *Scaleable, Secure Cash Payment for WWW Resources with the PayMe Protocol Set,* World Wide Web Journal, 1(1):587:602, O'Reilly & Associates, December 1995.

[P03] Pravda Newspaper. *1998 Default Was Inevitable, Expert Opinion.* On-line version last updated on 18 August 2003. http://english.pravda.ru/main/18/89/358/10736_default.html (22/09/2003)

[PWFK02] Pearlman, L., Welch, V., Foster, I., Kesselman, C., Tuecke, S. *A Community Authorization Service for Group Collaboration.* Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks, 2002.

[RFC1510] RFC 1510, *The Kerberos Network Authentication Service (Version 5).* Internet Request For Comment, Internet Engineering Taskforce, http://www.faqs.org/rfcs/rfc1510.html (22/09/2003)

[R92] Rivest, R. *The MD5 message-digest algorithm.* Internet Request for Comments, April 1992. RFC 1321.

[RS96] Rivest, R., Shamir, A. *PayWord and MicroMint: Two simple micropayment schemes.* CryptoBytes, volume 2, number 1 (RSA Laboratories, Spring 1996), pp. 7-11. Also in the proceedings of 1996 International Workshop on Security Protocols, April 1996, Cambridge, United Kingdom.

[RSA03] RSA BSafe Software Development Kit – *RSA Security.* http://www.rsasecurity.com/products/bsafe/bsafe.html (17/10/2003)

[RUSW03] Resource Usage Service Working Group, *Globus Grid Forum*. http://www.doc.ic.ac.uk/~sjn5/GGF/rus-wg.html (17/10/2003)

[SALH02] Sherwani, J., Ali, N., Lotia, N., Hayat, Z., Buyya, R., *Libra: An Economy driven Job Scheduling System for Clusters*, Proceedings of the 6th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2002), December 16-19, 2002, Bangalore, India.

[SGE03] Sun Grid Engine (SGE): A Cluster Resource Manager. Sun Microsystems Inc. (22/09/2003) http://gridengine.sunsource.net/project/gridengine/documentation.html

[SLS03] Somogyi, C., Laszlo, Z., Szeberenyi, I. *A Resource Accounting and Charging System in Condor Environment.* Proceedings of the International Conference on Parallel and Distributed Computing, August 26-29, 2003, Klagenfurt, Austria.

[SSLP96] Secure Socket Layer Protocol, Specification Draft, 1996. http://wp.netscape.com/eng/ssl3/draft302.txt (22/09/2003)

[SWBC96] Sullivan, W., Werthimer, D., Bowyer, S., Cobb, J., Gedye, D., Anderson, D. *A New Major SETI Project based on Project Serendip Data and 100,000 Personal Computers.* Proceedings of the 5th International Conference on Bioastronomy, July 1-5, 1996, Capri, Italy.

[TCFF02] Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., *Grid Service Specification*. Open Grid Service Infrastructure Working Group, Global Grid Forum. Draft 3 of 17th of July 2002.

[TH01] Thigpen, B., Hacker, T. *Distributed Accounting on the Grid.* 1st Global Grid Forum (GGF1) Working Draft, 4-9 March, 2001, Amsterdam, Netherlands.

[VL01] Vazhkudai, S., Laszewski, G. *A Greedy Grid – The Grid Economic Directive.* Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS2001), Workshop on Internet Computing and E-Commerce (ICEC2001), pg. 173, April 23-27, 2001, San Francisco, California, USA.

[W97] Wayner, P. *Digital Cash.* 2nd Edition. ISBN 0-409-31316-5. AP Professional, Academic Press Limited, 1997.

[YVB02] Yu, J., Venugopal, S., Buyya, R. *Grid Market Directory: A Web Services based Grid Service Publication Directory.* Technical Report, November 18, 2002. GRIDS Laboratory, the University of Melbourne, Australia. http://www.cs.mu.oz.au/~raj/grids/gmd/ (22/09/2003)

[ZZWD93] Zhou, S., Zheng, X., Wang, J., Delisle, P., *UTOPIA: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems.* Software – Practice and Experience, 23(12):1305-1336, December 1993.

# Appendix

# GridBank Database, APIs and External Data Representations

## Appendix A: GridBank MySQL Database Tables

This appendix presents database tables created on a GridBank server. These tables contain information about GSC's and GSP's accounts, past and outstanding infra and inter-branch transactions, Resource Usage Records, GridBank administrators, authorised branches (banks), as well as storing next account number, which is sequentially incremented when a new account is open. The central branch in each country also stores the next branch number.

### A.1 List of Database Tables

```
+------------------------+
| Tables_in_gridbankAU_1 |
+------------------------+
| ChargeDetails          |
| RUR                    |
| accounts               |
| administrators         |
| authorised_banks       |
| deferred               |
| deferred_id            |
| histau_0001_00000001   |
| histau_0001_00000002   |
| inter_branch_trans     |
| nextaccount            |
| nextbranch             |
| nexttrans              |
| transfers              |
+------------------------+
```

## A.2 Accounts Table

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| AccountID    | varchar(16)  | YES  |     | NULL    |       |
| CertName     | varchar(150) | YES  |     | NULL    |       |
| Details      | varchar(50)  | YES  |     | NULL    |       |
| AvailBalance | float        | YES  |     | NULL    |       |
| LockBalance  | float        | YES  |     | NULL    |       |
| Currency     | varchar(10)  | YES  |     | NULL    |       |
| Credit       | float        | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

## A.3 ChargeDetails Table

```
+---------------------+---------------------+------+-----+---------+-------+
| Field               | Type                | Null | Key | Default | Extra |
+---------------------+---------------------+------+-----+---------+-------+
| TransID             | bigint(20) unsigned | YES  |     | NULL    |       |
| RateUnit            | varchar(20)         | YES  |     | NULL    |       |
| FlatCharge          | float               | YES  |     | NULL    |       |
| WallClockTimeRate   | float               | YES  |     | NULL    |       |
| WallClockTimeCharge | float               | YES  |     | NULL    |       |
| UserTimeRate        | float               | YES  |     | NULL    |       |
| UserTimeCharge      | float               | YES  |     | NULL    |       |
| SystemTimeRate      | float               | YES  |     | NULL    |       |
| SystemTimeCharge    | float               | YES  |     | NULL    |       |
+---------------------+---------------------+------+-----+---------+-------+
```

## A.4 Administrators Table

```
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| CertName  | varchar(150) | YES  |     | NULL    |       |
| CreatedBy | varchar(150) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
```

## A.5 Authorised Banks Table

```
+-------------------------+--------------+------+-----+---------+-------+
| Field                   | Type         | Null | Key | Default | Extra |
+-------------------------+--------------+------+-----+---------+-------+
| BankBranchNumber        | varchar(7)   | YES  |     | NULL    |       |
| BankCertName            | varchar(150) | YES  |     | NULL    |       |
| ServerURL               | varchar(50)  | YES  |     | NULL    |       |
| LocalCurrencyName       | char(3)      | YES  |     | NULL    |       |
| ExchangeRateLocalForeign| float        | YES  |     | NULL    |       |
| ForeignCurrencyName     | char(3)      | YES  |     | NULL    |       |
| ForeignCurrencyBalance  | float        | YES  |     | NULL    |       |
+-------------------------+--------------+------+-----+---------+-------+
```

## A.6 Resource Usage Records Table

```
+---------------------------+---------------------------------------------------------------------+------+-----+---------+-------+
| Field                     | Type                                                                | Null | Key | Default | Extra |
+---------------------------+---------------------------------------------------------------------+------+-----+---------+-------+
| TransID                   | bigint(20) unsigned                                                 | YES  |     | NULL    |       |
| Type                      | enum('computational','storage','specialized')                       | YES  |     | NULL    |       |
| UserCertificateName       | varchar(150)                                                        | YES  |     | NULL    |       |
| UserProjectName           | varchar(100)                                                        | YES  |     | NULL    |       |
| UserContact               | varchar(100)                                                        | YES  |     | NULL    |       |
| JobStatus                 | enum('done','failed','aborted')                                     | YES  |     | NULL    |       |
| ExecutableName            | varchar(255)                                                        | YES  |     | NULL    |       |
| JobSubmissionDateGMT      | datetime                                                            | YES  |     | NULL    |       |
| JobEndDateGMT             | datetime                                                            | YES  |     | NULL    |       |
| LocalJobID                | varchar(50)                                                         | YES  |     | NULL    |       |
| NumberOfProcesses         | tinyint(3) unsigned                                                 | YES  |     | NULL    |       |
| UserTimeSeconds           | mediumint(8) unsigned                                               | YES  |     | NULL    |       |
| UserTimeMilliseconds      | smallint(5) unsigned                                                | YES  |     | NULL    |       |
| SystemTimeSeconds         | mediumint(8) unsigned                                               | YES  |     | NULL    |       |
| SystemTimeMilliseconds    | smallint(5) unsigned                                                | YES  |     | NULL    |       |
| MemoryMeasurementUnit     | enum('B','KB','MB','GB','TB','PB','EB','ZB','YB')                   | YES  |     | NULL    |       |
| MainMemoryAllocated       | float                                                               | YES  |     | NULL    |       |
| MainMemoryActual          | float                                                               | YES  |     | NULL    |       |
| SecondaryStorageAllocated | float                                                               | YES  |     | NULL    |       |
| SecondaryStorageActual    | float                                                               | YES  |     | NULL    |       |
| NetworkIO                 | float                                                               | YES  |     | NULL    |       |
| ResourceCertificateName   | varchar(150)                                                        | YES  |     | NULL    |       |
| ResourceIPAddress         | varchar(15)                                                         | YES  |     | NULL    |       |
| ResourceManufacturer      | varchar(25)                                                         | YES  |     | NULL    |       |
| CPUType                   | varchar(10)                                                         | YES  |     | NULL    |       |
| CPUSpeedUnit              | enum('Hz','KHz','MHz','GHz','THz','PHz','EHz','ZHz','YHz')          | YES  |     | NULL    |       |
| CPUSpeed                  | float                                                               | YES  |     | NULL    |       |
| RAM                       | float                                                               | YES  |     | NULL    |       |
| HardDiskSpace             | float                                                               | YES  |     | NULL    |       |
| NetworkBandwidthUnit      | enum('bps','Kbps','Mbps','Gbps','Tbps','Pbps','Ebps','Zbps','Ybps') | YES  |     | NULL    |       |
| NetworkBandwidthIn        | float                                                               | YES  |     | NULL    |       |
| NetworkBandwidthOut       | float                                                               | YES  |     | NULL    |       |
| Charged                   | tinyint(1)                                                          | YES  |     | NULL    |       |
+---------------------------+---------------------------------------------------------------------+------+-----+---------+-------+
```

## A.7 Deferred Payments Table

```
+-----------------+---------------------+------+-----+---------+-------+
| Field           | Type                | Null | Key | Default | Extra |
+-----------------+---------------------+------+-----+---------+-------+
| ID              | bigint(20) unsigned | YES  |     | NULL    |       |
| DrawerAccID     | varchar(16)         | YES  |     | NULL    |       |
| DrawerAccCN     | varchar(150)        | YES  |     | NULL    |       |
| Amount          | float               | YES  |     | NULL    |       |
| NoOfPayments    | int(10) unsigned    | YES  |     | NULL    |       |
| RecipientAccID  | varchar(16)         | YES  |     | NULL    |       |
| RecipientAccCN  | varchar(150)        | YES  |     | NULL    |       |
| EntryDate       | timestamp(14)       | YES  |     | NULL    |       |
| ExpiryDate      | datetime            | YES  |     | NULL    |       |
+-----------------+---------------------+------+-----+---------+-------+
```

## A.8 Deferred Payment Next Transaction Identification Number Table

```
+-----------------+---------------------+------+-----+---------+-------+
| Field           | Type                | Null | Key | Default | Extra |
+-----------------+---------------------+------+-----+---------+-------+
| ID              | bigint(20) unsigned | YES  |     | NULL    |       |
| DrawerAccID     | varchar(16)         | YES  |     | NULL    |       |
| DrawerAccCN     | varchar(150)        | YES  |     | NULL    |       |
| Amount          | float               | YES  |     | NULL    |       |
| NoOfPayments    | int(10) unsigned    | YES  |     | NULL    |       |
| RecipientAccID  | varchar(16)         | YES  |     | NULL    |       |
| RecipientAccCN  | varchar(150)        | YES  |     | NULL    |       |
| EntryDate       | timestamp(14)       | YES  |     | NULL    |       |
| ExpiryDate      | datetime            | YES  |     | NULL    |       |
+-----------------+---------------------+------+-----+---------+-------+
```

## A.9 Account History Table

```
+---------+---------------------+------+-----+---------+-------+
| Field   | Type                | Null | Key | Default | Extra |
+---------+---------------------+------+-----+---------+-------+
| TransID | bigint(20) unsigned | YES  |     | NULL    |       |
| Type    | varchar(10)         | YES  |     | NULL    |       |
| Date    | timestamp(14)       | YES  |     | NULL    |       |
| Amount  | float               | YES  |     | NULL    |       |
| DoneBy  | varchar(150)        | YES  |     | NULL    |       |
+---------+---------------------+------+-----+---------+-------+
```

## A.10 Inter-Branch Transactions Table

```
+---------+---------------------+------+-----+---------+-------+
| Field   | Type                | Null | Key | Default | Extra |
+---------+---------------------+------+-----+---------+-------+
| TransID | bigint(20) unsigned | YES  |     | NULL    |       |
| Type    | varchar(10)         | YES  |     | NULL    |       |
| Date    | timestamp(14)       | YES  |     | NULL    |       |
| Amount  | float               | YES  |     | NULL    |       |
| DoneBy  | varchar(150)        | YES  |     | NULL    |       |
+---------+---------------------+------+-----+---------+-------+
```

## A.11 Next Account Number Table

```
+---------+---------------------+------+-----+---------+-------+
| Field   | Type                | Null | Key | Default | Extra |
+---------+---------------------+------+-----+---------+-------+
| Bank    | char(2)             | YES  |     | NULL    |       |
| Branch  | smallint(5) unsigned | YES  |     | NULL    |       |
| Account | int(10) unsigned    | YES  |     | NULL    |       |
+---------+---------------------+------+-----+---------+-------+
```

## A.12 Next Branch Number Table (Central Branch only)

```
+--------+----------------------+------+-----+---------+-------+
| Field  | Type                 | Null | Key | Default | Extra |
+--------+----------------------+------+-----+---------+-------+
| Branch | smallint(5) unsigned | YES  |     | NULL    |       |
+--------+----------------------+------+-----+---------+-------+
```

## A.13 Next Transaction Identification Number Table

```
+-------------+---------------------+------+-----+---------+-------+
| Field       | Type                | Null | Key | Default | Extra |
+-------------+---------------------+------+-----+---------+-------+
| Transaction | bigint(20) unsigned | YES  |     | NULL    |       |
+-------------+---------------------+------+-----+---------+-------+
```

## A.14 Transfers (local transactions) Table

```
+----------------+---------------------+------+-----+---------+-------+
| Field          | Type                | Null | Key | Default | Extra |
+----------------+---------------------+------+-----+---------+-------+
| TransID        | bigint(20) unsigned | YES  |     | NULL    |       |
| Date           | timestamp(14)       | YES  |     | NULL    |       |
| DrawerAccID    | varchar(16)         | YES  |     | NULL    |       |
| Amount         | float               | YES  |     | NULL    |       |
| RecipientAccID | varchar(16)         | YES  |     | NULL    |       |
| ForeignTransID | varchar(28)         | YES  |     | NULL    |       |
+----------------+---------------------+------+-----+---------+-------+
```

# Appendix B: GridBank API

The following C functions (in B.1) define GridBank accounting and payment protocols. These functions are translated into SOAP Web Services Definition Language (in B.2). They are invoked by GSCs and GSPs, which must first authenticate themselves to GridBank via the Grid Security Infrastructure (Globus I/O module of the Globus Toolkit). Once an entity is authenticated, Globus I/O provides the certificate subject name in to every function, and each protocol ensures that the account number (i.e. "fromAccount") is associated with that certificate name in order to authorise the transaction. "OperationResult" is the structure returned from each function invocation and is context-dependent. For example, request statement operation returns the statement in the string part and 0 to indicate successful outcome, whereas a failed operation returns the error code and a string explaining the error.

## B.1 C binding

### B.1.1 OperationResult

```
struct gb__OperationResult
{
  int OperationFailure;
  char* String;
};
```

### B.1.2 DirectPayment

```
int gb__DirectPayment(char* fromAccount,
                      char* toAccount,
                      char* amount,
                      char* resourceURL,
                      struct gb__OperationResult* r);
```

### B.1.3 Transfer Resource Usage Record

```
int gb__transferResourceUsageRecord(char* transactionReference,
                                    char* resourceUsageRecord,
                                     struct gb__OperationResult* r);
```

### B.1.4 Defer Resource Usage Record

```
int gb__deferResourceUsageRecord(char* foreignTransactionReference,
                                 char* lastHashToken,
                                 char* resourceUsageRecord,
                                  struct gb__OperationResult* r);
```

### B.1.5 Request Grid Cheque

```
int gb__requestCheque(char* fromAccount,
                      char* maxAmount,
                      char* toAccount,
                      char* toAccountCertificateName,
                      char* resourceURL,
                      struct gb__OperationResult* r);
```

### B.1.6 Redeem Grid Cheque

```
int gb__redeemCheque(char* deferredTransactionReference,
                     char* toAccount,
                     char* fromAccountCertificateName,
                     char* resourceUsageRecord,
                     char* actualAmount,
                     struct gb__OperationResult* r);
```

### B.1.7 Request Grid Hash Chain

```
int gb__requestGridHash(char* fromAccount,
                        char* hashValue,
                        unsigned int noOfHashes,
                        char* toAccount,
                        char* toAccountCN,
                        char* resourceURL,
                        struct gb__OperationResult* r);
```

### B.1.8 Redeem Grid Hash Chain

```
int gb__redeemGridHash(char* deferredTransactionReference,
                       char* toAccount,
                       char* fromAccountCertificateName,
                       char* resourceUsageRecord,
                       char* lastHash,
                       unsigned int hashSequenceNumber,
                       struct gb__OperationResult* r);
```

### B.1.9 Request GridBank Statement

```
int gb__requestStatement(char* accountNumber,
                         char* startDate,
                         char* endDate,
                         struct gb__OperationResult* r);
```

### B.1.10 Inter-Branch DirectPayment

```
int gb__interbankDirectPayment(char* fromAccount,
                               char* fromAccountCertificateName,
                               char* foreignTransactionReference,
                               char* toAccount,
                               char* amount,
                               char* currencyName,
                               char* resourceURL,
                               struct gb__OperationResult* r);
```

### B.1.11 Reconcile Inter-Branch Transactions

```
int gb__reconcileInterBranchTransactions(char* fromBankBranchNumber,
                                       char*xmlInterBranchTransactions,
                                       struct gb__OperationResult* r);
```

### B.1.12 Next Bank-Branch Number

int gb__getNextBankBranchNumber(struct gb__OperationResult* r);

### B.1.13 Authorise Branch

```
int gb__authoriseBranch(char* bankBranchNumber,
                        char* branchCertificateName,
                        char* serverURL,
                        char* foreignCurrencyName,
                        char* foreignCurrencyBalance,
                        char* exchangeRateLocalToForeign,
                        char* localCurrencyName,
                        struct gb__OperationResult* r);
```

### B.1.14 Create Administrator

```
int gb__createAdministrator(char* certificateName,
                            struct gb__OperationResult* r);
```

### B.1.15 Open Account

```
int gb__openAccount(char* accountCertificateName,
                    char* accountDetails,
                    char* currencyName,
                    struct gb__OperationResult* r);
```

### B.1.16 Deposit

```
int gb__deposit(char* accountNumber,
                char* accountCertificateName,
                char* amount,
                struct gb__OperationResult* r);
```

### B.1.17 Withdraw

```
int gb__withdraw(char* accountNumber,
                 char* accountCertificateName,
                 char* amount,
                 struct gb__OperationResult* r);
```

### B.1.18 Cancel Transaction

```
int gb__cancelTransfer(char* transactionReference,
                       struct gb__OperationResult* r);
```

### B.1.19 Change Credit Limit

```
int gb__changeCreditLimit(char* accountNumber,
                          char* newLimit,
                          struct gb__OperationResult* r);
```

### B.1.20 Close Account

```
int gb__closeAccount(char* accountNumber,
                     struct gb__OperationResult* r);
```

## B.2 SOAP Binding (GridBank Web Service Definition Language API)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="GridBank"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="gridbank.com.au:3000/GridBank.wsdl"
 xmlns:tns="gridbank.com.au:3000/GridBank.wsdl"
 xmlns:gb="gridbank.com.au:3000/gb.xsd">

<types>
 <schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  targetNamespace="gridbank.com.au:3000/gb.xsd"
  xmlns:gb="gridbank.com.au:3000/gb.xsd">

  <complexType name="OperationResult">
   <all>
    <element name="OperationFailure" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    <element name="String" type="xsd:string" minOccurs="1"
maxOccurs="1" nillable="true"/>
   </all>
  </complexType>

  <complexType name="getNextBankBranchNumber">
   <all>
   </all>
  </complexType>

 </schema>
</types>

<message name="DirectPaymentRequest">
 <part name="fromAccount" type="xsd:string"/>
 <part name="toAccount" type="xsd:string"/>
 <part name="amount" type="xsd:string"/>
 <part name="resourceURL" type="xsd:string"/>
</message>

<message name="OperationResult">
 <part name="OperationFailure" type="xsd:int"/>
 <part name="String" type="xsd:string"/>
</message>

<message name="transferResourceUsageRecordRequest">
 <part name="transactionReference" type="xsd:string"/>
 <part name="resourceUsageRecord" type="xsd:string"/>
</message>

<message name="deferResourceUsageRecordRequest">
 <part name="foreignTransactionReference" type="xsd:string"/>
 <part name="lastHashToken" type="xsd:string"/>
 <part name="resourceUsageRecord" type="xsd:string"/>
```

```
</message>

<message name="requestChequeRequest">
 <part name="fromAccount" type="xsd:string"/>
 <part name="maxAmount" type="xsd:string"/>
 <part name="toAccount" type="xsd:string"/>
 <part name="toAccountCertificateName" type="xsd:string"/>
 <part name="resourceURL" type="xsd:string"/>
</message>

<message name="redeemChequeRequest">
 <part name="deferredTransactionReference" type="xsd:string"/>
 <part name="toAccount" type="xsd:string"/>
 <part name="fromAccountCertificateName" type="xsd:string"/>
 <part name="resourceUsageRecord" type="xsd:string"/>
 <part name="actualAmount" type="xsd:string"/>
</message>

<message name="requestGridHashRequest">
 <part name="fromAccount" type="xsd:string"/>
 <part name="hashValue" type="xsd:string"/>
 <part name="noOfHashes" type="xsd:unsignedInt"/>
 <part name="toAccount" type="xsd:string"/>
 <part name="toAccountCN" type="xsd:string"/>
 <part name="resourceURL" type="xsd:string"/>
</message>

<message name="redeemGridHashRequest">
 <part name="deferredTransactionReference" type="xsd:string"/>
 <part name="toAccount" type="xsd:string"/>
 <part name="fromAccountCertificateName" type="xsd:string"/>
 <part name="resourceUsageRecord" type="xsd:string"/>
 <part name="lastHash" type="xsd:string"/>
 <part name="hashSequenceNumber" type="xsd:unsignedInt"/>
</message>

<message name="requestStatementRequest">
 <part name="accountNumber" type="xsd:string"/>
 <part name="startDate" type="xsd:string"/>
 <part name="endDate" type="xsd:string"/>
</message>

<message name="interbankDirectPaymentRequest">
 <part name="fromAccount" type="xsd:string"/>
 <part name="fromAccountCertificateName" type="xsd:string"/>
 <part name="foreignTransactionReference" type="xsd:string"/>
 <part name="toAccount" type="xsd:string"/>
 <part name="amount" type="xsd:string"/>
 <part name="currencyName" type="xsd:string"/>
 <part name="resourceURL" type="xsd:string"/>
</message>

<message name="reconcileInterBranchTransactionsRequest">
 <part name="fromBankBranchNumber" type="xsd:string"/>
 <part name="xmlInterBranchTransactions" type="xsd:string"/>
</message>

<message name="getNextBankBranchNumberRequest">
</message>

<message name="authoriseBranchRequest">
 <part name="bankBranchNumber" type="xsd:string"/>
 <part name="branchCertificateName" type="xsd:string"/>
 <part name="serverURL" type="xsd:string"/>
 <part name="foreignCurrencyName" type="xsd:string"/>
 <part name="foreignCurrencyBalance" type="xsd:string"/>
 <part name="exchangeRateLocalToForeign" type="xsd:string"/>
 <part name="localCurrencyName" type="xsd:string"/>
</message>

<message name="createAdministratorRequest">
 <part name="certificateName" type="xsd:string"/>
</message>

<message name="openAccountRequest">
 <part name="accountCertificateName" type="xsd:string"/>
 <part name="accountDetails" type="xsd:string"/>
 <part name="currencyName" type="xsd:string"/>
</message>
```

```
<message name="depositRequest">
 <part name="accountNumber" type="xsd:string"/>
 <part name="accountCertificateName" type="xsd:string"/>
 <part name="amount" type="xsd:string"/>
</message>

<message name="withdrawRequest">
 <part name="accountNumber" type="xsd:string"/>
 <part name="AccountCertificateName" type="xsd:string"/>
 <part name="amount" type="xsd:string"/>
</message>

<message name="cancelTransferRequest">
 <part name="transactionReference" type="xsd:string"/>
</message>

<message name="changeCreditLimitRequest">
 <part name="accountNumber" type="xsd:string"/>
 <part name="newLimit" type="xsd:string"/>
</message>

<message name="closeAccountRequest">
 <part name="accountNumber" type="xsd:string"/>
</message>

<portType name="GridBankPortType">
 <operation name="DirectPayment">
  <documentation>Service definition of function
gb__DirectPayment</documentation>
  <input message="tns:DirectPaymentRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="transferResourceUsageRecord">
  <documentation>Service definition of function
gb__transferResourceUsageRecord</documentation>
  <input message="tns:transferResourceUsageRecordRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="deferResourceUsageRecord">
  <documentation>Service definition of function
gb__deferResourceUsageRecord</documentation>
  <input message="tns:deferResourceUsageRecordRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="requestCheque">
  <documentation>Service definition of function
gb__requestCheque</documentation>
  <input message="tns:requestChequeRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="redeemCheque">
  <documentation>Service definition of function
gb__redeemCheque</documentation>
  <input message="tns:redeemChequeRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="requestGridHash">
  <documentation>Service definition of function
gb__requestGridHash</documentation>
  <input message="tns:requestGridHashRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="redeemGridHash">
  <documentation>Service definition of function
gb__redeemGridHash</documentation>
  <input message="tns:redeemGridHashRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="requestStatement">
  <documentation>Service definition of function
gb__requestStatement</documentation>
  <input message="tns:requestStatementRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="interbankDirectPayment">
  <documentation>Service definition of function
gb__interbankDirectPayment</documentation>
  <input message="tns:interbankDirectPaymentRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="reconcileInterBranchTransactions">
  <documentation>Service definition of function
gb__reconcileInterBranchTransactions</documentation>
  <input message="tns:reconcileInterBranchTransactionsRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="getNextBankBranchNumber">
  <documentation>Service definition of function
gb__getNextBankBranchNumber</documentation>
  <input message="tns:getNextBankBranchNumberRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="authoriseBranch">
  <documentation>Service definition of function
gb__authoriseBranch</documentation>
  <input message="tns:authoriseBranchRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="createAdministrator">
  <documentation>Service definition of function
gb__createAdministrator</documentation>
  <input message="tns:createAdministratorRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="openAccount">
  <documentation>Service definition of function
gb__openAccount</documentation>
  <input message="tns:openAccountRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="deposit">
  <documentation>Service definition of function
gb__deposit</documentation>
  <input message="tns:depositRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="withdraw">
  <documentation>Service definition of function
gb__withdraw</documentation>
  <input message="tns:withdrawRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="cancelTransfer">
  <documentation>Service definition of function
gb__cancelTransfer</documentation>
  <input message="tns:cancelTransferRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="changeCreditLimit">
  <documentation>Service definition of function
gb__changeCreditLimit</documentation>
  <input message="tns:changeCreditLimitRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="closeAccount">
  <documentation>Service definition of function
gb__closeAccount</documentation>
  <input message="tns:closeAccountRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
</portType>

<binding name="GridBankBinding" type="tns:GridBankPortType">
 <SOAP:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="DirectPayment">
  <SOAP:operation soapAction="gridbank.com.au#DirectPayment"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="transferResourceUsageRecord">
```

```
  <SOAP:operation soapAction="gridbank.com.au#transferResourceUsageRecord"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="deferResourceUsageRecord">
  <SOAP:operation soapAction="gridbank.com.au#deferResourceUsageRecord"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="requestCheque">
  <SOAP:operation soapAction="gridbank.com.au#requestCheque"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="redeemCheque">
  <SOAP:operation soapAction="gridbank.com.au#redeemCheque"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="requestGridHash">
  <SOAP:operation soapAction="gridbank.com.au#requestGridHash"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="redeemGridHash">
  <SOAP:operation soapAction="gridbank.com.au#redeemGridHash"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="requestStatement">
  <SOAP:operation soapAction="gridbank.com.au#requestStatement"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="interbankDirectPayment">
  <SOAP:operation soapAction="gridbank.com.au#interbankDirectPayment"/>
  <input>
```

```
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="reconcileInterBranchTransactions">
  <SOAP:operation
soapAction="gridbank.com.au#reconcileInterBranchTransactions"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="getNextBankBranchNumber">
  <SOAP:operation soapAction="gridbank.com.au#getNextBankBranchNumber"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="authoriseBranch">
  <SOAP:operation soapAction="gridbank.com.au#authoriseBranch"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="createAdministrator">
  <SOAP:operation soapAction="gridbank.com.au#createAdministrator"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="openAccount">
  <SOAP:operation soapAction="gridbank.com.au#openAccount"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="deposit">
  <SOAP:operation soapAction="gridbank.com.au#deposit"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="withdraw">
  <SOAP:operation soapAction="gridbank.com.au#withdraw"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
```

```
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="cancelTransfer">
  <SOAP:operation soapAction="gridbank.com.au#cancelTransfer"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="changeCreditLimit">
  <SOAP:operation soapAction="gridbank.com.au#changeCreditLimit"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="closeAccount">
  <SOAP:operation soapAction="gridbank.com.au#closeAccount"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
</binding>

<service name="GridBank">
 <documentation>gSOAP 2.1.6b generated service
definition</documentation>
 <port name="GridBankPort" binding="tns:GridBankBinding">
  <SOAP:address location="gridbank.com.au:3000/GridBank.cgi"/>
 </port>
</service>

</definitions>
```

# Appendix C: GridBank Service Provider Server API

This appendix is similar to appendix B, but lists functions implemented by a GSP. These are invoked by the GridBank server to confirm the amount of the payment or to cancel a transaction. A GSC only has access to one function: “payGridHash”. It is used to forward the next hash token every payment period.

## C.1 C Binding

### C.1.1 Operation Result

```
struct gb__OperationResult
{
  int OperationFailure;
  char* String;
};
```

### C.1.2 Submit DirectPayment

```
int gb__confirmDirectPayment(char* toAccount,
                             char* amount,
                             char* fromCertName,
                             char* transactionRef,
                             struct gb__OperationResult* r);
```

### C.1.3 Submit Grid Hash Chain

```
int gb__confirmGridHash(char* deferredTransRef,
                        char* toAccount,
                        char* toAccountCN,
                        char* hashValue,
                        char* zeroHash,
                        unsigned int rateSeconds,
                        char* expiryDate,
                        char* fromCertName,
                        struct gb__OperationResult* r);
```

### C.1.4 Pay Grid Hash Token

```
int gb__payGridHash(char* fromCertName,
                    char* hash,
                    unsigned int hashSeqNo,
                    struct gb__OperationResult* r);
```

### C.1.5 Submit Grid Cheque

```
int gb__confirmGridCheque(char* deferredTransRef,
                          char* toAccount,
                          char* toAccountCN,
                          char* maxAmount,
                          char* expiryDate,
                          char* fromAccountCN,
                          struct gb__OperationResult* r);
```

### C.1.6 Cancel Account and Authorisation

```
int gb__cancelAccount(char* CertName,
                      struct gb__OperationResult* r);
```

## C.2 SOAP Binding (GridBank Service Provider WSDL API)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="GridBank"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="gridbank.com.au:3000/GridBank.wsdl"
 xmlns:tns="gridbank.com.au:3000/GridBank.wsdl"
 xmlns:gb="gridbank.com.au:3000/gb.xsd">

<types>
 <schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  targetNamespace="gridbank.com.au:3000/gb.xsd"
  xmlns:gb="gridbank.com.au:3000/gb.xsd">

  <complexType name="OperationResult">
   <all>
    <element name="OperationFailure" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
    <element name="String" type="xsd:string" minOccurs="1"
maxOccurs="1" nillable="true"/>
   </all>
  </complexType>

 </schema>
</types>

<message name="confirmDirectPaymentRequest">
 <part name="toAccount" type="xsd:string"/>
 <part name="amount" type="xsd:string"/>
 <part name="fromCertName" type="xsd:string"/>
 <part name="transactionRef" type="xsd:string"/>
</message>

<message name="OperationResult">
 <part name="OperationFailure" type="xsd:int"/>
 <part name="String" type="xsd:string"/>
</message>

<message name="confirmGridHashRequest">
 <part name="deferredTransRef" type="xsd:string"/>
 <part name="toAccount" type="xsd:string"/>
 <part name="toAccountCN" type="xsd:string"/>
 <part name="hashValue" type="xsd:string"/>
 <part name="zeroHash" type="xsd:string"/>
 <part name="rateSeconds" type="xsd:unsignedInt"/>
 <part name="expiryDate" type="xsd:string"/>
 <part name="fromCertName" type="xsd:string"/>
</message>

<message name="payGridHashRequest">
 <part name="fromCertName" type="xsd:string"/>
 <part name="hash" type="xsd:string"/>
 <part name="hashSeqNo" type="xsd:unsignedInt"/>
</message>

<message name="confirmGridChequeRequest">
 <part name="deferredTransRef" type="xsd:string"/>
 <part name="toAccount" type="xsd:string"/>
 <part name="toAccountCN" type="xsd:string"/>
 <part name="maxAmount" type="xsd:string"/>
 <part name="expiryDate" type="xsd:string"/>
 <part name="fromAccountCN" type="xsd:string"/>
</message>

<message name="cancelAccountRequest">
 <part name="CertName" type="xsd:string"/>
</message>

<message name="updateServiceRatesRequest">
 <part name="serviceRates" type="xsd:string"/>
</message>

<portType name="GridBankPortType">
 <operation name="confirmDirectPayment">
  <documentation>Service definition of function
gb__confirmDirectPayment</documentation>
  <input message="tns:confirmDirectPaymentRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="confirmGridHash">
  <documentation>Service definition of function
gb__confirmGridHash</documentation>
  <input message="tns:confirmGridHashRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="payGridHash">
  <documentation>Service definition of function
gb__payGridHash</documentation>
  <input message="tns:payGridHashRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="confirmGridCheque">
  <documentation>Service definition of function
gb__confirmGridCheque</documentation>
  <input message="tns:confirmGridChequeRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="cancelAccount">
  <documentation>Service definition of function
gb__cancelAccount</documentation>
  <input message="tns:cancelAccountRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
 <operation name="updateServiceRates">
  <documentation>Service definition of function
gb__updateServiceRates</documentation>
  <input message="tns:updateServiceRatesRequest"/>
  <output message="tns:OperationResult"/>
 </operation>
</portType>

<binding name="GridBankBinding" type="tns:GridBankPortType">
 <SOAP:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="confirmDirectPayment">
  <SOAP:operation soapAction="gridbank.com.au#confirmDirectPayment"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
```

```
  </output>
 </operation>
 <operation name="confirmGridHash">
  <SOAP:operation soapAction="gridbank.com.au#confirmGridHash"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="payGridHash">
  <SOAP:operation soapAction="gridbank.com.au#payGridHash"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="confirmGridCheque">
  <SOAP:operation soapAction="gridbank.com.au#confirmGridCheque"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="cancelAccount">
  <SOAP:operation soapAction="gridbank.com.au#cancelAccount"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="updateServiceRates">
  <SOAP:operation soapAction="gridbank.com.au#updateServiceRates"/>
  <input>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="gridbank.com.au"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
</binding>

<service name="GridBank">
 <documentation>gSOAP 2.1.6b generated service
definition</documentation>
 <port name="GridBankPort" binding="tns:GridBankBinding">
  <SOAP:address location="gridbank.com.au:3000/GridBank.cgi"/>
 </port>
</service>

</definitions>
```

# Appendix D: XML-based XDRs

This appendix presents examples of various external data representations that are returned to the caller of a successful operation. The Resource Usage Record is forwarded by a GSP to the GridBank server at the end of a computation in order to redeem a deferred payment. The GridBank statement is returned in response to a request statement operation. The Service Rates Update is sent from a GridBank Service Consumer Client by a resource administrator to the GridBank Service Provider Server when updating service rates. The inter-branch reconciliation records are exchanged by branches when GridBank administrators invoke the reconciliation operation.

## D.1 Example of a Resource Usage Record

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<ResourceUsageRecord version="1.0">
  <Type>computational</Type>
  <ClientDetails>
    <ClientCertificateName>
      /O=Grid/O=Globus/OU=cs.uwa.edu.au/CN=Alexander Barmouta
    </ClientCertificateName>
    <ClientAddress>https://koel.csse.uwa.edu.au:33233/</ClientAddress>
  </ClientDetails>
  <JobDetails>
    <JobStatus>done</JobStatus>
    <ExecutableName>/home/local/barmouta/misc/test</ExecutableName>
    <JobSubmissionDateGMT>20040224134914</JobSubmissionDateGMT>
    <JobEndDateGMT>20040224134921</JobEndDateGMT>
    <GlobusJobmanagerJobID>2780.1077630548</GlobusJobmanagerJobID>
    <NumberOfProcesses>1</NumberOfProcesses>
    <UserTimeSeconds>6</UserTimeSeconds>
    <UserTimeMilliseconds>890</UserTimeMilliseconds>
    <SystemTimeSeconds>0</SystemTimeSeconds>
    <SystemTimeMilliseconds>100</SystemTimeMilliseconds>
    <MemoryMeasurementUnit>MB</MemoryMeasurementUnit>
    <PageReclaims>27</PageReclaims>
    <PageFaults>211</PageFaults>
  </JobDetails>
  <ResourceDetails>
    <ResourceCertificateName>
      /O=Grid/O=Globus/CN=host/koel.csse.uwa.edu.au
    </ResourceCertificateName>
    <ResourceAddress>koel.csse.uwa.edu.au</ResourceAddress>
    <ResourceManufacturer>pc</ResourceManufacturer>
    <CPUType>i686</CPUType>
    <OperatingSystem version="2.4.18">Linux</OperatingSystem>
    <CPUSpeedUnit>GHz</CPUSpeedUnit>
    <CPUSpeed>1.02</CPUSpeed>
    <MemoryMeasurementUnit>MB</MemoryMeasurementUnit>
    <RAM>128.0</RAM>
    <HardDiskSpace>31404.2</HardDiskSpace>
    <NetworkBandwidthUnit>Mbps</NetworkBandwidthUnit>
    <NetworkBandwidthIn>100.0</NetworkBandwidthIn>
    <NetworkBandwidthOut>100.0</NetworkBandwidthOut>
  </ResourceDetails>
  <ChargeDetails>
    <TimeRateUnit>G$second</TimeRateUnit>
    <Charge>
      <ChargeName>UserCPUTime</ChargeName>
      <Rate>0.1</Rate>
      <ItemCharge>0.7</ItemCharge>
    </Charge>
    <Charge>
      <ChargeName>SystemCPUTime</ChargeName>
      <Rate>0.2</Rate>
      <ItemCharge>1.378</ItemCharge>
    </Charge>
    <Charge>
      <ChargeName>WallclockTime</ChargeName>
      <Rate>0.3</Rate>
      <ItemCharge>0.03</ItemCharge>
    </Charge>
    <TotalCharge>2.108</TotalCharge>
  </ChargeDetails>
</ResourceUsageRecord>
```

## D.2 Example of a GridBank Statement

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<GridBankStatement version="1.0">
  <AccountDeails>
    <AccountID>au-0001-00000001</AccountID>
    <CertName>
      /O=Grid/O=Globus/OU=cs.uwa.edu.au/CN=Alexander Barmouta
    </CertName>
    <OrgName>UWA</OrgName>
    <AvailBalance>484.32</AvailBalance>
    <LockBalance>0</LockBalance>
    <Currency>G$</Currency>
    <Credit>0</Credit>
  </AccountDetals>
  <Deposits>
    <Deposit>
      <TransID>1</TransID>
      <Type>deposit</Type>
      <Date>20040123161027</Date>
      <Amount>50.12</Amount>
      <DoneBy>
        /O=Grid/O=Globus/OU=cs.uwa.edu.au/CN=Alexander Barmouta
      </DoneBy>
    </Deposit>
  </Deposits>
  <Withdrawals/>
  <Transfers>
    <Transfer>
      <TransID>2</TransID>
      <Date>20040213163625</Date>
      <DrawerAccID>au-0001-00000001</DrawerAccID>
      <Amount>0.1</Amount>
      <RecipientAccID>au-0001-00000001</RecipientAccID>
      <ForeignTransID>NULL</ForeignTransID>
      <ResourceUsageRecord version="1.0">
        <Type>computational</Type>
        <ClientDetails>
          <ClientCertificateName>
            /O=Grid/O=Globus/OU=cs.uwa.edu.au/CN=Alexander Barmouta
          </ClientCertificateName>
          <ClientAddress>https://koel.csse.uwa.edu.au:33233/</ClientAddress>
        </ClientDetails>
        <JobDetails>
          <JobStatus>done</JobStatus>
          <ExecutableName>/home/local/barmouta/misc/test</ExecutableName>
          <JobSubmissionDateGMT>20040224134914</JobSubmissionDateGMT>
          <JobEndDateGMT>20040224134921</JobEndDateGMT>
          <GlobusJobmanagerJobID>2780.1077630548</GlobusJobmanagerJobID>
          <NumberOfProcesses>1</NumberOfProcesses>
          <UserTimeSeconds>6</UserTimeSeconds>
          <UserTimeMilliseconds>890</UserTimeMilliseconds>
          <SystemTimeSeconds>0</SystemTimeSeconds>
          <SystemTimeMilliseconds>100</SystemTimeMilliseconds>
          <MemoryMeasurementUnit>MB</MemoryMeasurementUnit>
          <PageReclaims>27</PageReclaims>
          <PageFaults>211</PageFaults>
        </JobDetails>
        <ResourceDetails>
          <ResourceCertificateName>
            /O=Grid/O=Globus/CN=host/koel.csse.uwa.edu.au
          </ResourceCertificateName>
          <ResourceAddress>koel.csse.uwa.edu.au</ResourceAddress>
          <ResourceManufacturer>pc</ResourceManufacturer>
          <CPUType>i686</CPUType>
          <OperatingSystem version="2.4.18">Linux</OperatingSystem>
          <CPUSpeedUnit>GHz</CPUSpeedUnit>
          <CPUSpeed>1.02</CPUSpeed>
          <MemoryMeasurementUnit>MB</MemoryMeasurementUnit>
          <RAM>128.0</RAM>
         <HardDiskSpace>31404.2</HardDiskSpace>
          <NetworkBandwidthUnit>Mbps</NetworkBandwidthUnit>
          <NetworkBandwidthIn>100.0</NetworkBandwidthIn>
          <NetworkBandwidthOut>100.0</NetworkBandwidthOut>
        </ResourceDetails>
        <ChargeDetails>
          <TimeRateUnit>G$second</TimeRateUnit>
          <Charge>
            <ChargeName>UserCPUTime</ChargeName>
            <Rate>0.1</Rate>
            <ItemCharge>0.7</ItemCharge>
          </Charge>
          <Charge>
            <ChargeName>SystemCPUTime</ChargeName>
            <Rate>0.2</Rate>
            <ItemCharge>1.378</ItemCharge>
          </Charge>
          <Charge>
            <ChargeName>WallclockTime</ChargeName>
            <Rate>0.3</Rate>
            <ItemCharge>0.03</ItemCharge>
          </Charge>
          <TotalCharge>2.108</TotalCharge>
        </ChargeDetails>
      </ResourceUsageRecord>
    </Transfer>
  </Transfers>
</GridBankStatement>
```

## D.3 Example of a Service Rates Update

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<ServiceRatesUpdate>
   <CurrencyName>GridDollar</CurrencyName>
   <UserCPUTime>
    <Unit>second</Unit>
    <Rate>1.2</Rate>
   </UserCPUTime>
   <SystemCPUTime>
    <Unit>second</Unit>
    <Rate>1.5</Rate>
   </SystemCPUTime>
   <WallclockTime>
    <Unit>second</Unit>
    <Rate>0.4</Rate>
   </WallclockTime>
   <MainMemory>
    <Unit>MB*second</Unit>
    <Rate>0.1</Rate>
```

```
  </MainMemory>
  <SecondaryStorage>
    <Unit>GB*second</Unit>
    <Rate>0.06</Rate>
  </SecondaryStorage>
  <NetworkTraffic>
    <Unit>MB</Unit>
    <Rate>12.48</Rate>
  </NetworkTraffic>
</ServiceRatesUpdate>
```

## D.4 Example of a Inter-Branch Transactions Reconciliation Record

### D.4.1 Inter-Branch Transactions Reconciliation Request

```
<?xml version="1.0" encoding="UTF-16" standalone="no" ?>
<InterBranchReconciliation>
  <Transaction>
    <LocalTransactionReference>13</LocalTransactionReference>
    <EntryDate>20040213164853</EntryDate>
    <ExpiryDate>20050213164853</ExpiryDate>
    <DrawerAccount>AU-0001-00000001</DrawerAccount>
    <DrawerCertificateName>
      /O=Grid/O=Globus/OU=cs.uwa.edu.au/CN=Alexander Barmouta
    </DrawerCertificateName>
    <RecipientAccount>US-0001-00000002</RecipientAccount>
    <RecipientCertificateName>
      /O=Grid/O=Globus/CN=host/koel.csse.uwa.edu.au
    </RecipientCertificateName>
    <ForeignCurrency>USD</ForeignCurrency>
    <ForeignAmount>6.18</ForeignAmount>
    <NumberOfPaymentTokens>1</NumberOfPaymentTokens>
    <Reconciled>no</Reconciled>
  </Transaction>
    <LocalTransactionReference>14</LocalTransactionReference>
    <EntryDate>20040213165404</EntryDate>
    <ExpiryDate>20050213165404</ExpiryDate>
    <DrawerAccount>AU-0001-00000001</DrawerAccount>
    <DrawerCertificateName>
      /O=Grid/O=Globus/OU=cs.uwa.edu.au/CN=Alexander Barmouta
    </DrawerCertificateName>
    <RecipientAccount>US-0001-00000002</RecipientAccount>
    <RecipientCertificateName>
      /O=Grid/O=Globus/CN=host/koel.csse.uwa.edu.au
    </RecipientCertificateName>
    <ForeignCurrency>USD</ForeignCurrency>
    <ForeignAmount>3.17</ForeignAmount>
    <NumberOfPaymentTokens>1</NumberOfPaymentTokens>
    <Reconciled>no</Reconciled>
  <Transaction>
  </Transaction>
</InterBranchReconciliation>
```

### D.4.2 Inter-Branch Transactions Reconciliation Reply

```
<?xml version="1.0" encoding="UTF-16" standalone="no" ?>
<InterBranchReconciliation>
  <Transaction>
    <LocalTransactionReference>1</LocalTransactionReference>
    <EntryDate>20040213165725</EntryDate>
    <ExpiryDate>20050213165725</ExpiryDate>
    <DrawerAccount>AU-0001-00000001</DrawerAccount>
    <DrawerCertificateName>
      /O=Grid/O=Globus/OU=cs.uwa.edu.au/CN=Alexander Barmouta
    </DrawerCertificateName>
    <RecipientAccount>US-0001-00000002</RecipientAccount>
    <RecipientCertificateName>
      /O=Grid/O=Globus/CN=host/koel.csse.uwa.edu.au
    </RecipientCertificateName>
    <LocalCurrency>USD</LocalCurrency>
    <LocalAmount>6.01</LocalAmount>
    <ForeignTransactionReference>13</ForeignTransactionReference>
    <NumberOfPaymentTokens>1</NumberOfPaymentTokens>
    <Reconciled>yes</Reconciled>
  </Transaction>
  <Transaction>
    <LocalTransactionReference>2</LocalTransactionReference>
    <EntryDate>20040213165934</EntryDate>
    <ExpiryDate>20050213165934</ExpiryDate>
    <DrawerAccount>AU-0001-00000001</DrawerAccount>
    <DrawerCertificateName>
      /O=Grid/O=Globus/OU=cs.uwa.edu.au/CN=Alexander Barmouta
    </DrawerCertificateName>
    <RecipientAccount>US-0001-00000002</RecipientAccount>
    <RecipientCertificateName>
      /O=Grid/O=Globus/CN=host/koel.csse.uwa.edu.au
    </RecipientCertificateName>
    <LocalCurrency>USD</LocalCurrency>
    <LocalAmount>2.85</LocalAmount>
    <ForeignTransactionReference>14</ForeignTransactionReference>
    <NumberOfPaymentTokens>1</NumberOfPaymentTokens>
    <Reconciled>yes</Reconciled>
  </Transaction>
</InterBranchReconciliation>
```