A Pricing Information Service for Grid Computing

Alexandru Caracas*
IBM Research Labs Zürich
Säumerstrasse 4, 8803
Rüschlikon, Switzerland
xan@zurich.ibm.com

Jörn Altmann†
International University
Campus 3, 76646
Bruchsal, Germany
jorn.altmann@acm.org

ABSTRACT

This paper addresses two shortcomings that exist in the area of pricing Grid services in an economic Grid environment. The first shortcoming is that there are no standards for pricing schemes, caused by a large difference in the units that are traded (e.g. CPU cycles or virtual clusters) in Grid computing. The second shortcoming is the lack of a model for managing the pricing of informational elements (e.g. software applications) and computational elements (e.g. virtual machines, which comprise resources such as CPU, memory, disk space, network bandwidth). This paper presents a pricing service for Grid computing services, which resolves the shortcomings by introducing a general pricing scheme for informational and computational elements. We describe the functional requirements, architecture, and the interfaces of the pricing service. The pricing service allows expressing the proposed general pricing scheme as an XML document, which can be linked to service level agreements. Contrary to other proposals on pricing, the pricing service is separated from the functionality of metering, accounting, and payment. To validate the concept of a pricing information service, we portray two Utility Computing scenarios.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Services—*Web-based services*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Payment schemes*

General Terms

Management, Architecture

*This work is based on Alexandru Caracas' Master Thesis work at the International University in Germany.

†Jörn Altmann is also affiliated with TEMEP, Seoul National University, San56-1, Sillim-Dong, Gwanak-Gu, Seoul 151-742, South Korea.

MGC '07, November 26, 2007 Newport Beach - CA, USA

Keywords

Grid Utility Computing, Pricing Schemes, Pricing Information Service, Service Level Agreements

1. INTRODUCTION

Grid computing is an established field, which solves issues related to connecting and using distributed computing resources. The technology to manage, share, and use computing resources is called Grid middleware. The middleware offers all the necessary functions such as security [24], data transfer, communication, and scheduling. There are several Grid middleware available such as Globus [17], Gridbus [21], UNICORE [29], gLite [16]. The role of managing and scheduling jobs on computing resources (e.g. computer clusters and individual PCs) is taken by so-called resource management systems (RMSs).

Grid economics is a relatively young field that analyzes the economic principles needed to enable markets of Grid services [3, 10, 11]. One of the main concerns is to define the appropriate support needed from the Grid technology or middleware to perform economic tasks such as accounting, pricing, economically efficient brokering of various sorts, and capacity planning. The vision is that the Grid would evolve towards a new world-wide business platform, which offers services on-demand and creates new market opportunities[1, 2, 25]. Providers such as Amazon [5] and SUN [27], who currently have commercial offerings of Grid resources on-demand, took already the first step to bring the economic Grid vision closer.

One of the foremost concerns in Grid economics is the unit being traded and the respective pricing. Currently, Grid middleware provide no standard units of trade or pricing scheme and there is a lack of support for managing advanced pricing information. This paper looks at the next generation economic Grid and analyzes the units of trade in such an environment. Based on the analysis, this paper introduces a general pricing scheme for Grid services and an architecture for a pricing information service.

This paper is structured as follows. The next section describes the state of the art in Grid middleware, focusing on economic-related services, especially pricing services. The units of trade and their respective pricing schemes in an economic-enhanced Grid environment are introduced in Section 3. The design and architecture and prototype implementation of the proposed pricing service are discussed in Section 4. Section 5 illustrates the suitability of the pricing information service using two Utility Computing scenarios. The last section concludes with a summary of our work.

2. STATE-OF-THE-ART ECONOMIC GRIDS

This section describes Grid middleware and respective Grid projects that deal with pricing services. Grid middleware can be roughly classified according to the functionality provided in terms of economic-related services.

2.1 GRIA

GRIA is a lightweight economic middleware, which enables businesses to use the Grid in a secure, inter-operable, and flexible manner based on Web Services and SLAs [19].

The GRIA SLA management service accounts for the use of the following default resources: CPU, current activities, disk space, and jobs. There is also the possibility to define customized resource types. However, resources in GRIA have a single dimension and prices are statically defined in an SLA. Moreover, there is no dynamic pricing concept and no history of resource prices [28].

2.2 Globus

Globus Toolkit offers a mature basic Grid middleware based on open standards and components [17]. The current version of the Globus Toolkit (GT4) is used to manage, share, and use computational resources across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The GT4 can be used to build further services and high-level Grid frameworks such as Gridbus.

In addition to the basic functionality provided by GT4, there are 3rd party solutions, e.g. the Swedish Grid Accounting System (SGAS) that provides accounting functionality. However, even with the SGAS add-on, the GT4 lacks economic-aware components such as a proper billing stack with pricing, charging, and accounting for complex services. Different brokering components are missing as well as trust, risk, and SLA management.

2.3 Gridbus

The Gridbus project aims at producing a set of economic Grid middleware services to support e-science and e-business applications using the computational Grid service architecture, described in [12].

One of the components of the Gridbus middleware is the Grid Bank, which provides the infrastructure for Grid accounting and payment [7]. It uses SOAP over Globus Toolkit's sockets. The GridBank service lacks support for pricing schemes that support reservations or dynamic prices. Moreover, computational resources can have only a single dimension (e.g. CPU cycles).

2.4 EGEE

The EGEE project focuses on maintaining the gLite middleware and on operating a large computing infrastructure for the benefit of a vast and diverse research community. The gLite middleware is a solution that provides a basic framework for building Grid application that leverage distributed computing and storage resources across the Internet.

With respect to pricing, the EGEE project leverages the Distributed Grid Accounting System (DGAS) [13], which provides a Price Authority (PA) that assigns prices to the subset of Grid resources within its administrative domain [14]. The prices are kept in a historic database and are assigned either manually or using different pricing algorithms. The default dynamic pricing algorithm increases the price for jobs with lower waiting times. Moreover, the PA allows for custom implementations of dynamic pricing algorithms.

The main difference between our pricing service and the PA is that our approach provides a uniform way to express bundles of resources. Moreover, our approach allows defining pricing schemes without the need to write and compile custom source code. Another difference is that the design of our pricing service enables the computation of prices based on historic prices or resource utilization.

2.5 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) is a Web service that provides resizable compute capacity, designed to make Web-scale computing easier for developers, while the related Amazon Simple Storage Service (Amazon S3) provides storage on demand[4]. The core infrastructure uses the Xen Linux virtual machine [6].

Amazon EC2 offers on-demand computing resources in the form of a virtual machine that is accessible via the Internet. The user has full control of virtual machines equivalent to a 1.7GHz Xeon CPU, 1.75GB RAM, 160GB HDD, 250Mb/s network at a price of $0.10 per instance-hour (or part hour) [4]. The pricing scheme offered by Amazon is simplistic but inflexible, e.g. it is not possible to apply dynamic prices or to differentiate prices according to virtual machine performance. Moreover, reserving computational power is not possible as well.

2.6 Sun Grid Compute Utility

Sun offers on-demand Grid services. The Sun Grid Compute Utility enables customers to purchase computing and storage power as they need it [23]. The Sun Grid follows the utility computing model offering computing resources at the price of $1 per CPU hour using the infrastructure of Sun Grid Engine (SGE) [9].

As in the case of Amazon EC2, the pricing scheme offered is not flexible. There is no price differentiation and no possibility to make reservations. Moreover, the pricing is only uni-dimensional and considers only CPU cycles, with no memory or network traffic specifications.

2.7 Comparison of Grid middleware

Table 1 summarizes the pricing schemes and services available in the previously described Grid technologies. The table also illustrates the maturity level of the technologies and the degree to which they address economics aspects.

Grid Middleware	Pricing		Technology Maturity	Degree of Economics
	Service	Scheme		
GRIA	no	resource-usage	ripe	high
SUN Grid	no	CPU-usage	mature	low
EC2	no	VM-usage	novel	low
GT4	no	none	mature	none
Gridbus	no	resource-usage	mature	high
EGEE	yes	custom	ripe	high

Table 1: Grid middleware technologies comparison

The comparison shows that, although SUN Grid Utility and Amazon EC2 are commercial produces, GRIA and Gridbus are the most advanced Grid middleware with respect to economics functionality. In particular, from the analyzed Grid Middleware, only EGEE provides a pricing service.

3. PRICING SCHEMES FOR GRID COMPUTING

Pricing schemes represent the business interface between the provider and the customer. They are used to achieve different objectives such as maximizing profit, maximizing social welfare, or defining certain schemes of fairness. A pricing scheme presents the unit of trade in a specified time period with a certain quality of service for a class of users. The following subsections describe an approach that can handle arbitrary pricing functions for Grid resources.

3.1 Units of Trade

Grid services can be information services or hardware services. Information services provide access to software and data, whereas hardware services provide access to hardware resources. One example of a hardware service is the Amazon EC2 described in Section 2. An example of data information services is Google Maps [18] and of software information services is Salesforce.com [26]. An information service provider acquires resources from a hardware service provider to execute its software offers. The hardware service provider also sells its services to end-users directly, who want to execute their private software.

Both services, information and hardware, operate with different units of trade according to Figure 1. On the one hand, the information service requires an information element (IE), which can range from software, via geographical coordinates, to maps. On the other hand, the hardware service requires a computational element (CE), which can be any hardware resource (e.g. CPU). Since a client of a hardware service requires not only CPU cycles but also main memory, network traffic, and persistent storage, it makes sense to trade a bundle of these resources. A bundle can consist of a set of CPUs, main memory, persistent storage, and network capacity. These bundles of resources will also be referred to as computational elements (CE). The advantage of using this bundles (i.e. high-level CEs) is that they can be understood easier by clients of hardware service providers, since CEs resemble a product familiar to the client (e.g. a virtual machine comprising a 1.3GHz P4 PC with 1GB main memory). Therefore, a CE represents a level of abstraction from the actual resources. The next natural abstraction level for a CE would comprise an entire cluster of servers, which is build using virtualization software [15]. This extra level of abstraction aims to improve the management and user friendliness of large systems.

3.2 General Pricing Scheme

A general pricing scheme is a quadruple (Q, T, C, U) of the *quantity* Q, the *time* T, the *quality class* C, and the *user profile* U. The quantity Q represents a tuple Q_L, Q_M, where Q_L specifies a limit for the quantity. Q_M is the quantity consumed above the limit Q_L. The quantity is the number of units of trade (i.e. either CE or IE), e.g. the number of transactions performed (e.g. per-transaction pricing), the number of resources consumed (e.g. per-byte pricing), or the duration of access to a unit of trade (e.g. per-time pricing). The time T is used for controlling reservations and is also expressed as a vector variable composed of: current time t_c, start time t_s, and duration t_d. The quality class C of services allows to specify different quality types. The last variable, the user profile U, represents history information of the user's consumption, the valuation of the user's importance to the business, or special promotions. To create a pricing scheme, one specifies the unit of trade and the parameters that determine the price variation. For example, a unit of trade for a hardware service could be defined as one *PC* (i.e. CE), which can be accessed *Now* at *Gold* quality for users belonging to the group *Consulting*.

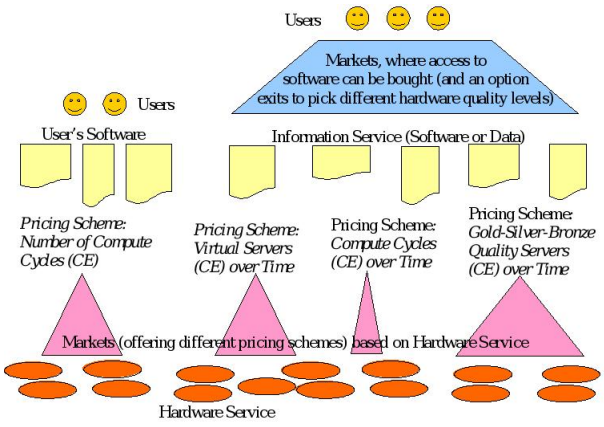


Figure 1: Markets for units of trade

The general pricing scheme allows service providers to define pricing variations as functions for any IEs and/or CEs, where an IE or CE can be a bundle of individual resources.

3.3 Example of Existing Pricing Schemes

In the following, we present a few examples of pricing schemes, which can be found in literature about Grid economics or in existing Grid market offers.

Usage-based pricing: Under usage-based pricing, the fee is based on the actual resource consumption of a CE or IE. For example, the fee for using a CPU (i.e. CE) is calculated by multiplying the price per compute cycle with the number of compute cycles that the CE has been used. Expressed in the general pricing scheme, it means that Q_L is zero and Q_M is the actual quantity consumed. If the considered quantity unit is time-based with a time period of a month or longer, the pricing is called *"flat-rate pricing"*.

Amazon EC2 (Elastic Compute Cloud), which offers a CE equivalent to a server with 1.7GHz Xeon CPU, 1.75GB RAM, 160GB HDD, 250Mb/s network, is priced at 0.10$ per instance-hour [5]. A similar approach is taken by the SUN Grid, which offers compute cycles at 1$ / per CPU-hour. In contrast to Amazon, SUN does not specify the equivalent of the hardware resources they provide.

The usage-based pricing plan is also applied to software applications. In this case, the time spent working with the application or the number of transactions performed with the application is considered.

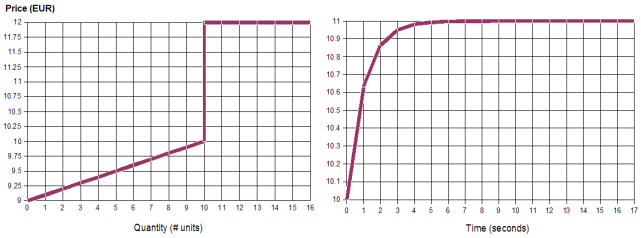
Progressive Co-design: Seller and buyer try to convene on a pricing plan. The seller announces a price pair (p_1, p_2), where $p_1 < p_2$. Subsequently, the buyer commits a consumption level Q_L for a fixed fee $p_1 Q_L$ and might buy additional units Q_M at p_2 if needed. The buyer chooses Q_L to maximize his surplus [8]. In this pricing scheme, the user specifies the tuple Q of the general pricing scheme.

Waiting-time-based pricing: The resource provider charges higher prices p per computer cycle for lower job queue waiting times . The goal of this pricing scheme is to

minimize queue waiting time through economic scheduling. This pricing is used within EGEE Price Authority [14]. The unit of trade is differentiated via quality parameter C within the Quadruple (Q, T, C, U) of the general pricing scheme.

4. DESIGN AND ARCHITECTURE OF A PRICING INFORMATION SERVICE

The design and architecture of a pricing service are based on the knowledge described within the previous two sections. The two main design goals are to manage informational and computational elements, and to manage the general pricing scheme described above, while keeping track of price history. The proposed pricing service is part of the general economic-enhanced Grid architecture developed within the GridEcon project [22].



(a) (b)

Figure 2: Price variance: quantity(a) and time(b).

We use two pricing functions to illustrate how the general pricing scheme is handled by the pricing service (see Figure 2). Figure 2(a) illustrates a pricing function where the price of the offered CE starts at 9 Euro and increases linearly to 10 Euro for the first 10 units consumed. Subsequently, the price per unit remains constant at 11 Euro for additional units consumed. Figure 2(b) shows a pricing function where the price of the CE increases asymptotically over time until it reaches 11 Euro.

Figure 3 shows the elaborate version of the general pricing scheme together with those two pricing functions. The pricing scheme is encoded using XML. The elaborated version includes constraints for hardware resource and price variation with respect to users and quality of service class. The example shows that the price for the user with an ID of 123 will get a price reduction of 2 Euro. The quality of service class *Gold* will cost 15 percent extra.

4.1 Functional Requirements

The pricing service provides the following main functions:

- Manage pricing schemes: create, edit, delete pricing schemes, and provide default schemes for commonly used bundles of resources (e.g. virtual machines).
- Set prices for CEs or IEs. It calculates the market price for a particular resource based on monitoring information (how many resources are utilized/free), past accounting information, and past prices. One example of price setting is a negotiation process.
- Keep price history of CEs and IEs. The history can be used to analyze trends. It may also serve as additional input to the price setting module.

Pricing Scheme

Virtual machine bundle: http://myresource.com/vm
Currency: EUR
Basic price: 10

Resources

URI	Label	Constraints	Add
http://myresource.com/cpu	CPU	EQ 3GHz	Edit Delete
http://myresource.com/RAM	RAM Memory	EQ 4GB	Edit Delete
http://myresource.com/HDD	HDD Storage Space	LT 200GB	Edit Delete
http://myresource.com/NET	Network Access	EQ 1000Mb/s	Edit Delete

Price variation

Quantity

Function Type	Lower Bound	UpperBound	Lower Modifier	Upper Modifier	Add
LINEAR	0	10	-10 %	0 %	Edit Delete
LINEAR	10	INF	+2	+2	Edit Delete

Time (unit = sec)

Function Type	Lower Bound	UpperBound	Lower Modifier	Upper Modifier	Add
EXPONENTIAL	0	INF	0 %	+10 %	Edit Delete

User

User ID	Modifier	Add
123	-2	Edit Delete

Class

Quality	Resource URI	Modifier	Add
GOLD	http://myresources.com/vm-gold	+15 %	Edit Delete

Figure 3: Price scheme editor screenshoot

- Answer queries. E.g.: What CE contains resource A and B? What pricing schemes fulfill certain minimum individual resource constraints?
- Merge pricing schemes for several CEs or IEs into a common pricing scheme.

4.2 Architecture

Figure 4 shows the architecture of the pricing service, which has the following five main functional components: Pricing Scheme Manager, History Keeper, Price Setter, Query Processor and Merger. The pricing service interacts with the following general Grid services: Resource Provider (Resource Broker), Market, Billing and Charging, Monitoring, Accounting, and SLA Service. The Resource Provider could also be directly represented through an RMS.

The **Pricing Scheme Manager** acts as the gatekeeper for the interface with the other services and also as an event handler for the communication among the price service modules. The manager supports the creation, modification and deletion of of pricing schemes. Actual price values and functions in the scheme can be set manually, or dynamically through the Price Setter module.

The **Price Setter** takes input from various sources such as availability and current usage of CEs to compute and set the actual prices for CEs.

The **History Keeper** keeps track of prices for CEs. This component records resource utilization. The price and resource utilization history is feed back into the Price Setter module.

The **Query Processor** answers queries related to pricing schemes and resources.

The **Merger** is able to unify several pricing schemes for several CEs or IEs into a single pricing scheme with a new respective pricing scheme.

The interfaces to interact with the pricing service are the following. We assume that individual resources can be uniquely identified through an URI:

- *addPricingScheme(PricingScheme obj).* It is used by the service administrator to add a new pricing scheme for a resource. The action is performed by the Pricing Scheme Manager.
- *deletePricingScheme(PricingScheme obj).* It is used by the

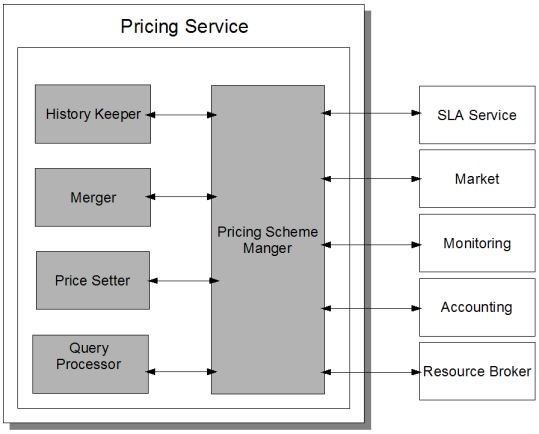


Figure 4: Pricing information service architecture

service administrator to delete an existing pricing scheme. The action is performed by the Pricing Scheme Manager.

- *getPricingScheme(String sqlLikeQuery).* It retrieves an existing pricing scheme based on given selection criteria. The action is performed by the Query Processor. This action can be used by any service (i.e. a client or the administrator of the service).
- *setCurrentPrice(String resourceUri, String, sourceUri, double priceValue).* It stores a price value for a current resource. The action is performed by the History Keeper. The function is called by a Market or Resource Broker service.
- *getCurrentPrice(String resourceUri).* This function can be called by the client, service administrator or any other service and is performed by the Query Processor with data from the History Keeper.
- *setCurrentUtilization(String resourceUri, Utilization util).* The function is called by the Monitoring service and is performed by the History Keeper.
- *getPriceHistory(String resourceUri, DateTime from, DateTime to).* The function can be called by the client, service administrator or any other service and is performed by the Query Processor with data from the History Keeper.
- *getUtilizationHistory(String resourceUri, DateTime from, DateTime to).* This function can be called by the client, service administrator or any other service and is performed by the Query Processor with data from the History Keeper.
- *mergePricingSchemes(PricingScheme[] objs).* It merges several pricing scheme into a unified pricing scheme. The function is called by the service administrator and is performed by the Merger.

4.3 Design

The *pricing service* is integrated as a prototype into the GRIA middleware. The service is implemented in Java and deployed as a Web Service in Tomcat using the framework provided by the GRIA middleware. The pricing service provides the price values for CEs and IEs to the SLA service in GRIA, which then charges the user account.

Figure 5 denotes the integration of the pricing service within the GRIA middleware together with the occurring interactions. The picture is an extension of the one provided in [20]. In *step 0*, the administrator of the service provider site designs and publishes pricing schemes and SLAs. To use a service, the client will have to: 1, obtain a trading account and 2, obtain an SLA. When the client wants to obtain an SLA for a given resource, the SLA service will query the pricing service to obtain a list of pricing schemes

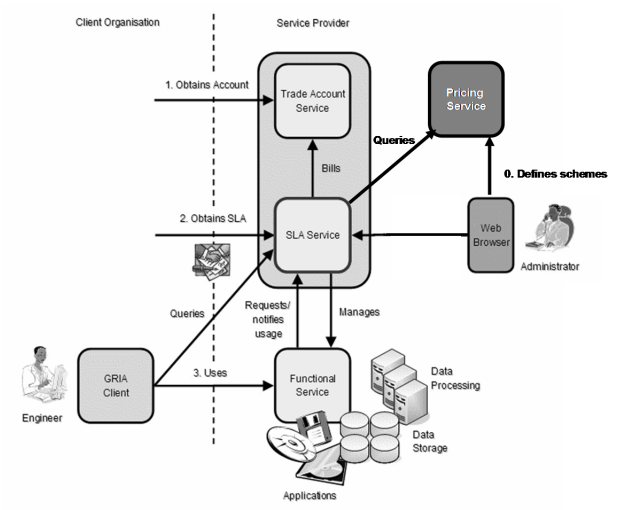


Figure 5: Pricing information service for GRIA

for that particular CE or IE. The client signs an SLA for the respective CE or IE, which contains a link to an appropriate pricing scheme. The price values in the pricing schemes are set by the pricing service on the provider side.

The pricing information service decouples the functionality of setting prices and defining pricing schemes from the process of defining and signing SLAs. This feature allows to dynamically set prices for CEs and IEs based on market inputs and other factors in a rapid changing business environment, while keeping the existing SLA. The service offers a comprehensive mean to define innovative pricing schemes for CEs and IEs.

5. VALIDATION

We validate the expediency of the pricing information service by presenting two Utility Computing scenarios which employ the service. The first scenario looks at the broad picture for Grid Utility Computing. The second scenario focuses on resource providers and shows that the flexibility of the general pricing scheme.

5.1 Scenario: Utility Computing

Utility Computing provides on-demand resources to consumers. However, different service providers offer different types of resources, priced and bundled in different ways, using heterogeneous Grid middleware technologies.

The proposed price information service allows a uniform way of expressing prices across heterogeneous Grid middleware by means of CEs or CIs bundles. This way of expressing prices allows for automatic processing of price information for example by a Grid broker service. The automatic processing of price information enables integration of the different Grid middleware into a global Utility Computing platform transparent to service consumers.

5.2 Scenario: Resource Utilization

Current SLAs only allow defining pricing functions linked to the resource usage of a user. The proposed pricing information service, however, allows prices to change dynamically based on various inputs: e.g., resource utilization information provided by the RMS, market indicators, or historic

data. Moreover, the proposed pricing scheme is general in the sense that it allows expressing arbitrary pricing functions. Being able to model generic pricing functions makes it possible to implement dynamic prices, e.g. prices which depend on the resource utilization by all consumers. The resource provider might want to increase prices if a certain utilization threshold is reached.

6. CONCLUSION

This paper analyzed economic-related issues, in particular pricing issues, present in today's Grid computing technologies. Based on those results, the paper introduced a general pricing scheme for informational and computational elements. The introduced general pricing scheme supports arbitrary pricing functions and allows for an uniform expression of prices across heterogeneous Grid middleware.

Another contribution of this paper is the introduction of an extensible pricing information service, which separates the functionality of setting, managing, and keeping track prices and pricing schemes from the definition and signing of service level agreements. The service has an extensible design and provides interfaces, which allow automatic processing of price information. The service is separated from the functionality of other Grid services such as metering, accounting, and payment.

A pricing service with features such as a generalized pricing scheme, uniform expression of units of trade, and separation from other billing functionality can be applied in many Grid middleware and in different Utility Computing scenarios. The proposed pricing service is the first step to enable trading of complex Grid computing services in an economic-aware Grid environment.

7. REFERENCES

[1] J. Altmann, C. Courcoubetis, J. Darlington, and J. Cohen. GridEcon - The Economic-Enhanced Next-Generation Internet. In *GECON 2007, Workshop on Grid Economics and Business Models*, Rennes, France, August 2007. Springer LNCS.

[2] J. Altmann, M. Ion, and A. B. Bany Mohammed. Taxonomy of grid business models. In *GECON 2007, Workshop on Grid Economics and Business Models*, Rennes, France, August 2007. Springer LNCS.

[3] J. Altmann and D.J. Veit. *Grid Economics and Business Models*. LNCS 4685, ISBN 0302-9743. Springer-Verlag, Berlin, Germany, August 2007.

[4] Amazon. Amazon EC2 Developer Guide, 2006.

[5] Amazon EC2. `http://aws.amazon.com/ec2`.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[7] A. Barmouta and R. Buyya. GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 245.1, Washington, DC, USA, 2003. IEEE Computer Society.

[8] H.K. Bhargava and A. Bagh. Tariff Structures for Pricing Grid Computing Resources. In *Gecon 2006*, Singapore, 2006.

[9] P. T. Bulhoes, C. Byun, R. Castrapel, and O. Hassaine. N1 grid engine 6 features and capabilities. In *SUPerG, the Sun Users Performance Group*, Phoenix, Arizona, May 2004.

[10] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service oriented grid computing. In *IPDPS '01: Proceedings of the 10th Heterogeneous Computing Workshop (HCW)*, page 20083.1, Washington, DC, USA, 2001. IEEE Computer Society.

[11] R. Buyya, D. Abramson, H. Stockinger, and J. Giddy. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 14:1507–1542, November-December 2002.

[12] R. Buyya and S. Venugopal. The Gridbus toolkit for service oriented grid and utility computing: An overview and status report. In *1st IEEE Int. Workshop Grid Economics and Business Models (GECON)*, 2004.

[13] DGAS. `http://www.to.infn.it/grid/accounting/`.

[14] EGEE DGAS: Price Authority. `https://edms.cern.ch/file/571271/1/EGEE-DGAS-PA-Guide.pdf`.

[15] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang. Virtual clusters for grid communities. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 513–520, Washington, DC, USA, 2006. IEEE Computer Society.

[16] gLite. `http://glite.web.cern.ch/glite/`.

[17] Gloubs toolkit. `http://www.globus.org/toolkit`.

[18] Google Maps. `http://maps.google.com`.

[19] GRIA. `http://www.gria.org`.

[20] GRIA Developers Kit Documentation.

[21] Gridbus. `http://www.gridbus.org`.

[22] GridEcon Consortium. D3.1 Grid Component Specification Report, 2007.

[23] Sun Micosystems Inc. Sun Grid Compute Utility: Developer's Guide, 2006.

[24] N.V. Kanaskar, U. Topaloglu, and C. Bayrak. Globus security model for grid environment. *SIGSOFT Softw. Eng. Notes*, 30(6):1–9, 2005.

[25] D. M. Quan and J. Altmann. The impact of business models on mapping policies for SLA-based workflows with light communication. In *HPCC 2007, High Performance Computation Conference*, Houston, USA, September 2007.

[26] Salesforce.com. `http://www.salesforce.com`.

[27] Sun Grid Compute Utility. `http://sun.com/sungrid`.

[28] M. Surridge, S. Taylor, D. De Roure, and E. Zaluska. Experiences with GRIA: Industrial applications on a web services grid. In *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*, pages 98–105, Washington, DC, USA, 2005. IEEE Computer Society.

[29] UNICORE. `http://www.unicore.eu`.

* All Web References have been accessed on August 2007.