Performance Measuring Framework for Grid Market Middleware

Felix Freitag[1], Pablo Chacin[1], Isaac Chao[1], Rene Brunner[1], Leandro Navarro[1], Oscar Ardaiz[2]

[1] Computer Architecture Department, Polytechnic University of Catalonia, Spain
{ felix, pchacin, ichao, rbrunner, leandro}@ac.upc.edu
[2] Department of Mathematics and Informatics, Public University of Navarra, Spain
oscar.ardaiz@unavarra.es

Abstract. Current implementations of Grid infrastructures provide frameworks which aim at achieve on-demand computing. In such a scenario, contribution and use of resources will be governed by business models. The challenge is to provide multi-level performance information which enables the participation of the different actors in such a system. In this paper we describe the performance measuring framework developed for Grid Market Middleware, a middleware which supports economic-model based selection of service-oriented Grid applications. This middleware is a distributed infrastructure, which we have implemented for providing a market of services and resources to be assigned to Grid applications. The objectives of the performance measuring framework is first to assess the behaviour of the middleware and the used economic models in a deployed system, and secondly allow the provision of metrics for the components of the middleware itself. We describe the design of the performance measuring framework, its implementation and show its capability and usefulness for our objectives by experiments.

Keywords: Multi-layer Performance Measurement, e-service Platforms, Grid Middleware Architecture

1 Introduction

In an on-demand scenario of service-oriented Grid applications, these applications will need service and resource providers to participate in a distributed software infrastructure. Clients will issue requests for Grid applications. The applications will be built on the fly in terms of complex services executing a sequence of basic services. Request for basic services are given to service providers and are executed on contracted resources.

There is a need for platforms that support such an on-demand infrastructures, where complex e-services can be assigned and executed. The resources need to be dynamically obtained to satisfy the requirement of changing application requirements. There is a need for technical openness and clear interfaces to the application level. At

the same time, there is a need for performance data that allows the participants making an economic evaluation of their participation.

In this paper we describe the performance measuring framework implemented in the Grid Market Middleware (GMM) [6] aiming at two objectives: 1) To obtain multi-level performance data from the deployed system, and 2) to show that the data obtained is useful for obtaining information about the system behaviour and its economic models used. The context in which we are interested to make our evaluation is that of the deployed middleware.

We have developed the Grid Market Middleware to support economic model based selection of service-oriented Grid applications. This middleware is a distributed infrastructure, which provides a market of services and resources to be assigned to Grid applications. The implemented market structure applies decentralized economic models based on bargaining [3]. The Grid Market Middleware offers an agent-based framework for dynamic location and management of Grid services based on economic criteria. It provides mechanisms to locate and manage the registered resources, services and applications, locate other trading agents, engage agents in negotiations, learn and adapt to changing conditions. Furthermore, the middleware offers a set of generic negotiation mechanism, on which specialized strategies and policies can be dynamically plugged in.

The presented performance measuring infrastructure for this middleware makes the following contribution: 1) Its functionality is shown by achieving the goal of detecting interesting behavior, such as load balancing. 2) It is a prototype that implements an approach to obtain multi-level performance data provided by the different layers of a distributed system, including application, middleware and base platform.

The remaining sections of this paper are organized as follows. Section 2 presents the Grid Market Middleware and the performance measuring goals. In section 3 we describe the design of the implemented performance measuring framework. Section 4 shows experimental results obtained with the implemented performance measuring infrastructure on the deployed middleware. Section 5 describes performance measurement frameworks of related domains. Finally, section 6 presents our conclusions.

2 Grid Market Middleware

2.1 Application Interaction with Grid Market Middleware

Grid Market Middleware as a software infrastructure, on which this performance measuring framework targets, contains an application level and the middleware. Figure 1 illustrates the interaction of the components on the application level with the middleware. When a client issues a request, an application component determines which Grid services are required to fulfill it. These Grid services represent both software services (e.g. a mathematical algorithm) and computational resources. The

application service translates these requirements into a WS-Agreement format [16], which is submitted to the Grid Market Middleware.

Once a request is received by the middleware, it searches among the available service providers, which have registered their particular service specifications. When a suitable service provider is found, agents within the middleware negotiate the application requirements. These agents act in behalf of the service providers as sellers and from the application perspective as buyers. Once an agreement is reached between the trading agents, a Grid service instance is created and a reference is returned to the application, which invokes it.

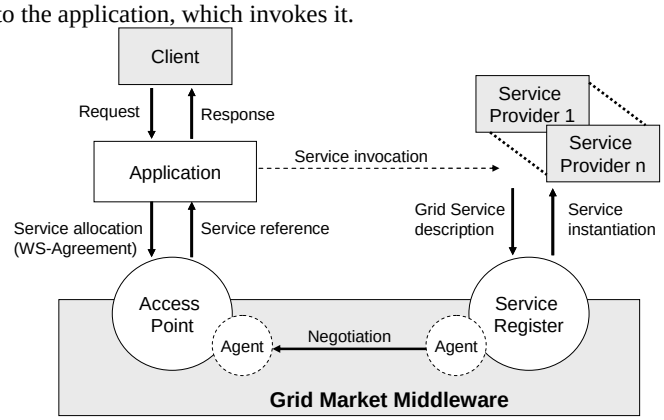

Fig. 1. Interaction between application and middleware.

The middleware is designed in a layered architecture with the five layers shown in figure 2. The *application layer* is for the domain specific end user application like collaboration tools, problem solving environments and others. The applications interact with the Grid Market Middleware in order to obtain the Grid services required to fullfil their functions. The Grid Market Middleware itself consists of three layers. Within the Market Middleware, the *economics algorithms* layer implements the high level economic behavior like negotiation and agent strategies used by the trading agents. The *economics framework* layer isolates these economic algorithms from the lower level technical details. The *P2P Agent* layer provides decentralized resource discovery and network topology maintenance. The *base platform* supports the application providing the hosting environment for the Grid services. A detailed description of the middleware architecture can be found in [1].

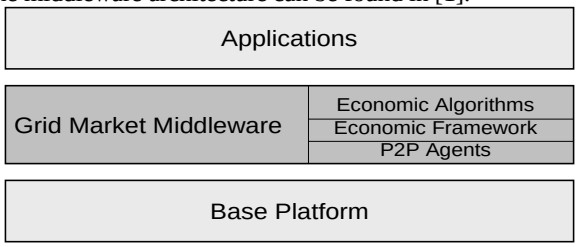

Fig. 2. Layered middleware architecture.

2.2 Middleware implementation

The implementation of the middleware builds on the use of different middleware toolkits, namely the DIET agent platform [2], JXTA [10] and the WSRF/OGSA implementation offered by the Globus Toolkit 4 [5]. DIET provides a modular, lightweight and scalable execution platform, JXTA offers a rich P2P networking environment and GT4 provides full support for resource management in different scenarios. The components of the middleware architecture have been implemented and an application, which uses this middleware, has been adapted [9].

3 Performance Measuring Framework

The development of the performance measuring framework follows the following approach. We define the goals and the metrics which should be obtained. We identify the measurement points given the developed middleware infrastructure. We proceed with the instrumentation and local data collection. Then, collecting and obtaining the data at a central point is the next step. The evaluation of the data is the final step of the process.

3.1 Goals of the performance measuring framework

The framework should be able to provide a large number of diverse metrics: The measurement should allow obtaining metrics both from the application, the middleware, and the physical level (base platform). In addition, there is a need for both technical and economic metrics. The economic metrics should be calculated from the technical metrics obtained by the framework and should assist the decision makers residing in the users (or applications). The framework should allow by means of mainly technical parameters evaluating the infrastructure itself.

The instrumentation of the middleware and application needs to be done at different levels. User agent related data is obtained at the application level. Technical parameters concerning the middleware performance will need to be instrumented at the corresponding levels of the middleware architecture. Finally, the monitoring of the physical resources is done by accessing the base platform.

The component of the metrics framework at the node level should locally gather the metrics obtained from the components working at the same node but at different layers of the infrastructure. The metrics collection should be done centrally, aiming to allow evaluating the prototype at this stage. Nevertheless, it is noted that the software infrastructure should run in a distributed manner on physically different devices and at a later stage with a potentially large number of nodes. For the prototype evaluation, however, although it is distributed, the number of nodes is small, such that a centralized approach for metrics collection at this stage appears acceptable.

The evaluation of these metrics, which are collected from the different nodes of the middleware, is done at a central point. The analysis and evaluation of the middleware is done off-line with external tools that provide the needed mathematical functions. Especially the higher-level economic metrics [15], see also appendix for the metrics pyramid, are computed off-line taking the raw data obtained from the performance measuring infrastructure. The metrics, which the agents need to take decisions, should be processed on-line by the agents themselves.

3.2 Metrics definition

We assign metrics to the layers of the software infrastructure. Beginning with the application layer (see also figure 1), there are a number of parameters to be measured in the client (which represents the end user) and the application. The client and the application perceive technical parameters, like the *service provision rate*, the *ratio between the number of requests and accepts*, and the *service provision time*, the *duration for obtaining an accept*. The client as end user will need to transform these technical parameters into economic ones, consider the benefits and the efforts, in order to determine the utility obtained by participating in this infrastructure. The application will calculate economic parameters from the technical data, if a business model for this component is defined.

Related to different levels of the middleware there are the following technical parameters: The *discovery time* refers to the time the middleware needs to find other agents to negotiate with. The *negotiation time* indicates the duration of the negotiation process. The negotiation process takes place in both the service and the resource market. Each negotiation consists of several messages according to the bargaining strategy. The *message size* is a parameter, which allows better describing the communication cost. The *number of messages* is another parameter concerning this cost. Load balancing is a metrics that should assess the efficiency of the resource assignment obtained with the Grid Market Middleware.

Concerning the base platform, the *resource usage* is measured. Initially, we focus on cpu usage.

In Table 1 some of the metrics classified into layers are summarized.

Table 1. Summary of metrics in different layers.

Layer	Metrics
Application layer	*service provision rate* *ratio between the number of requests and accepts* *duration for obtaining an accept*
Middleware layer	*discovery time* *negotiation time* *message size* *number of messages*
Base Platform layer	*resource usage*

3.3 Instrumentation and local data collector

In our approach, we took the design decision that data from one node should be locally collected. This way, we obtain at each node an event trace, which includes the metrics from the different middleware layers. Provision of these metrics is through agents (figure 3). The event trace contains the time stamps of the events, the metrics itself and a number of attributes like the agent number, transaction number, and others, in order to allow a detailed analysis of the behavior. The local data collector manages this data structure. In terms of implementation, a circular structure is used such that its size is controlled.

Access to this data structure is given in two ways. One hand, the data can be written to a file (log file), and on the other hand the local data collector can send it regularly to a global metrics collector located on a particular node.

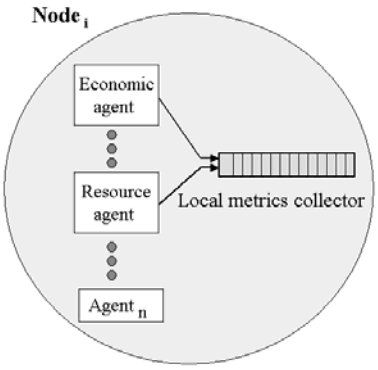

Fig. 3. Local metrics collector.

3.4 Global metrics collector

The data obtained at the different nodes is send to the global metrics collector, which resides on a particular node of the system. Data is send terms of a push mode: local data collectors initiate the sending to the global collector according to a configured behavior. The global metrics collector then processes and organizes the data into a format suitable for external packages. For our purpose we use Matlab for the evaluation of the data. We note that for larger scale usage beyond the current experimental environment, the automation of the clock synchronization between the nodes and the global metrics collector needs to be addressed.

4. Experimental results

Given the developed prototype and the implemented measurement infrastructure, we have carried out a number of experiments. These experiments aim to assure that the components work correctly, that multi-level performance data can effectively be obtained, and that this data is actually useful for the evaluation of the behaviour of the middleware.

4.1 Setup of the experiments

The particular goal of the experiment setting is to allow making an initial assessment on the load balancing behavior of the GMM by means of the implemented performance measuring framework.

The middleware uses as economic agents an implementation of the ZIP (Zero Intelligence Plus) agents [13]. Clients initiate negotiations with a price lower than the available budget. If they are not able to buy at that price, they increase their bids until either they win or reach the budget limit.

Services start selling the resources at a price, which is influenced by the node's utilization. Then, the pricing model is combined with the demand. If a service agent sells its resources, it will increase the price to test to what extend the market is willing to pay. When it no longer sells, it will lower the price until it becomes competitive again or it reaches a minimum price defined by the current utilization of the resource.

We have deployed the Grid Market Middleware in a Linux server farm. Each server has 2 Xeon processors and 2GB of memory. The machines in the server farm are connected by an internal Ethernet network at 100Mbps.

Three basic services (BS) are deployed on three servers (BS-74 on node 74, BS-75 on node 75, BS-79 on node 79, respectively), and two complex services (CS) are launched on two other servers (CS-72 on node 72, CS-73 on node 73, respectively). On each machine with a BS we also deploy a web service representing the application, which performs a CPU intensive calculation. These web services are exposed in a Tomcat server. Access to execute these web services is what is negotiated between complex services (buyer) and basic services (seller).

We run an artificial background load on two of the nodes (node 79, node 75) configured for 50% and 100% CPU usage to simulate background activity. This is chosen since in such a setting the behaviour of the agents should lead to load balancing of the web service executions.

The experiments consist in launching 2 clients (represented by complex services CS-72 and CS-73) concurrently as clients. Each client performs 50 requests in intervals of 15 seconds. Whenever a client wins a bid with a service, it invokes the web service in the selected node. The data obtained from the experiment with the performance measuring infrastructure has been the following:

1. *allocation*: an entry by each successful negotiation with a basic service, reported by the complex service
2. *price*: a periodic report of the price of the basic services
3. *utilization*: a periodic report of the CPU utilization given by the resource agents
4. *negotiation.time*: time needed to negotiate with a basic service, reported

by the complex service (transaction-based)
5. *execution.time*: time needed to actually execute the service, reported by the somplex service (transaction-based)

We mention that the data used is mixed in nature in the sense that some metrics are collected periodically (*price* and *utilization*), others are recorded after each successful transaction (*execution.time*, *negotiation.time* and *allocation*).

4.2 Experimental results

Figure 4 shows the load (% cpu usage) on the three nodes (74, 75, 76). A background load of 50% and 100% in nodes 79 and 75, respectively, can be observed. The up-going spikes which can be seen in the load of node 79 and node 74 correspond to the execution of the negotiated web services on these nodes.

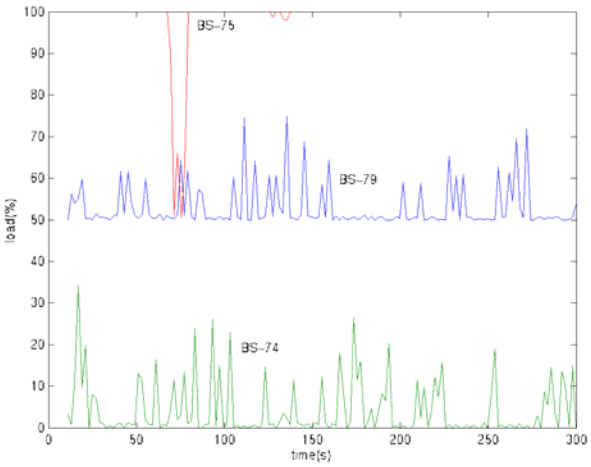

Figure 4. Load on nodes 74, 75, and 79. Node 79 and node 75 are with 50% and 100% background load, respectively.

Figure 5 shows a zoom on the price of the basic services. It can be observed that the price calculation of the agents takes into account the success of past negotiations, where the price rise is made after a successful sale. The configured buyer price is 100 money units.

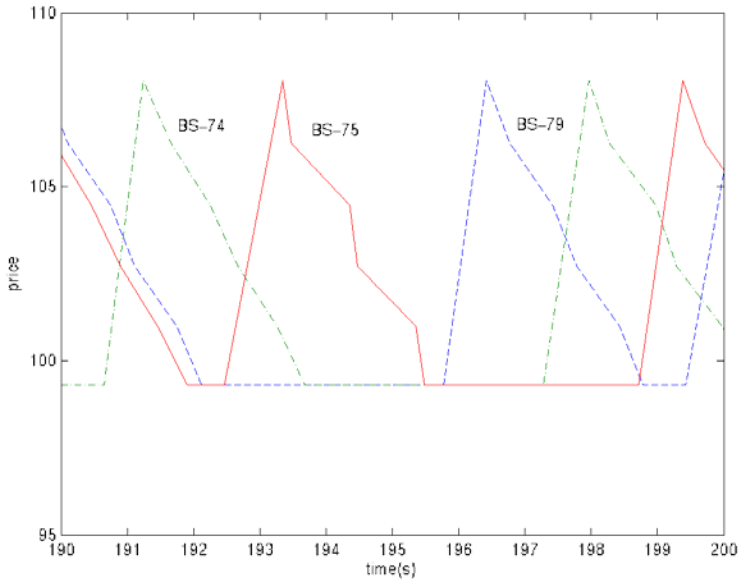

Figure 5. Zoom on the price evolution of the basic services in nodes 74, 75, and 79.

The next figure 6 is to assess the expected load balancing behavior, which we should obtain with this setting. It can be seen that effectively the BS-74, which runs on the least loaded node, makes most of the sales. And the BS-75, which runs on the node with the highest background load, makes less sells than the other two basic services. We can see that the performance measuring framework achieves one of our goals which was revealing such behavior.

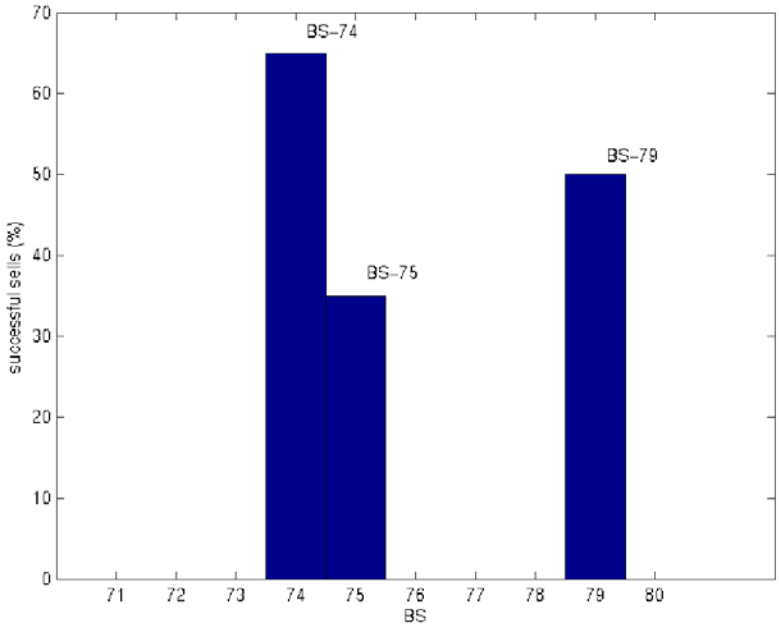

Figure 6. Percentage of sales of the three basic services. BS-74 which resides on the least loaded node, makes most of the sells.

Finally, in figure 7 we observe two metrics together: the successful sales by BS-74 and the execution of the sold service when invocated by the clients (the web service is executed on the same node 74). Successful sales by the BS-74 are indicted with a star symbol and are normalized here to the value 30 for easier visualization. It can be seen that on each successful sale an execution of the web service follows. It can be seen that the duration of the execution is approximately 4 seconds.

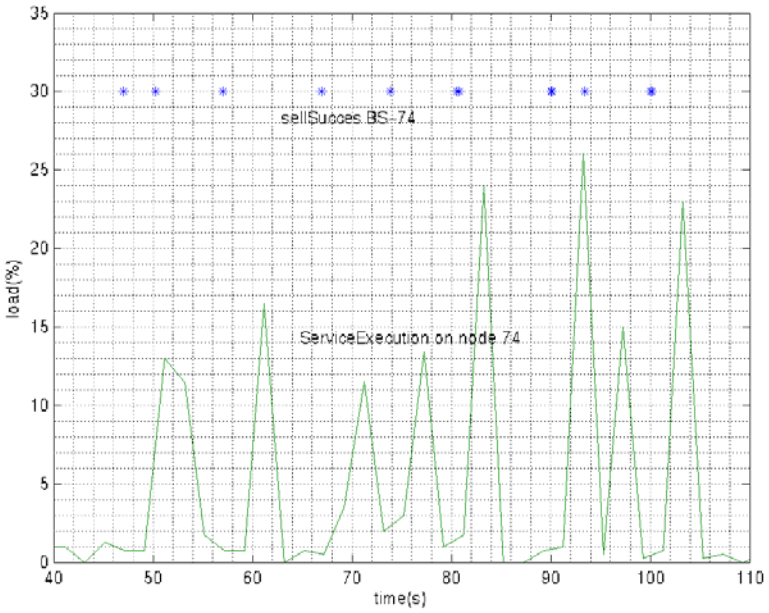

Figure 7. Successful sells of BS-74 and web service execution on node 74.

The experimental results demonstrate the two contributions we want to emphasize: First, the measurement framework achieves obtaining multi-level performance data of the distributed application. The data shown in the experiment is taken from the three main levels of the architecture: The time of the web service execution is measured at the application level. The evolution of prices is taken at the economics algorithm layer of the middleware. Finally, the load at each node is taken from the base platform layer. Secondly, the obtained data is useful for the analysis of the middleware and application (the setting was deliberately chosen to force load balancing behavior). We have seen that with the obtained data the expected behavior can effectively be observed.

4.3 Discussion

In this performance measuring framework one of the challenges is to obtain metrics from all layers of the system. As such, this framework needs to go beyond existing monitoring toolkits which mainly focus on the physical resources. For our purpose we also need to include application and middleware data. The results obtained from the implemented measuring framework experimental show that this has been achieved.

Another challenge relates to the usage of the metrics by the middleware (agents) itself, such that there will be different destinations for some of these metrics. In this

sense, there is the central metrics collection point, to which (currently for evaluation purpose) most of the data is sent and where the data is a posteriori analyzed and evaluated. On the other hand, there are the participants (the applications) as destination of metrics, since they need application layer metrics in order to take decisions and evaluate their performance. This second challenge, to route data to particular groups, has not been implemented yet in our framework. Our view on this is to apply publish/subscribe mechanisms in order to assign metrics to groups.

Anther issue, which has already become important when we measured particular agent strategies, is the scalability of the performance evaluation framework towards the quantity of the measured data, since large amounts of data are already obtained. This scalability problem affects the number of parameters which can be monitored, and may require in the future additional solutions such as shown in [4] to tackle the size of the traces obtained.

5. Related work

Related work can be found in multi-agent systems (MAS), high-performance computing and generic monitoring toolkits. Here we describe system from these three domains which also target on the performance evaluation of distributed systems.

In [7] the performance measuring system built into the Cougaar agent architecture is described. Cougaar belongs to distributed multi-agent systems (MAS). Like the GMM, it has been implemented 100% in Java. For performance measurement of Cougaar the authors decided to develop a custom-made performance measuring system which has been built into the system architecture. Cougaar also considers multiple data categories which are accessed via channels. The need for flexibility with respect to where to carry out the data processing, the access by semantics to categories, and dynamic plug-ins for changing needs has been taken into account in Cougaar. Similar to this approach, we have also decided to develop our own performance measuring framework directly fitting to our measurement needs, instead of adapting an exiting toolkit.

There are a number of measurement toolkits mainly with origin in high-performance computing, such as DiPerF [14] and NetLogger [7]. DiPerF aims to provide automatic performance measurement of networked services. It particularly targets on deployed services (such as a particular Grid service) and has been used in real testbeds such as PlanetLab [12]. DiPerF offers besides a set of preconfigured ones the possibility to include user specific metrics. NetLogger is oriented to anomaly detection and is presented as a service which can be activated for Grid applications. Its main purpose is monitoring and addressing the instrumentation level in running Grid processes, as part of a Grid monitoring system. Compared to our purpose, these toolkits focus on the executed application. In our case, however, we are particularly interested in measuring what happens before execution, i.e. within the middleware and the economic models used for allocating services and resources, before finally the application (service) becomes executed.

Ganglia [11] is a fairly generic measurement toolkit. It is devised to be a scalable distributed monitoring system for high performance computing systems such as clusters and Grids. It is deployed in PlanetLab [12] and has also been used with

Globus [5]. The Ganglia implementation consists of the gmond deamon, which runs on every node of the system. This deamon interacts with a client in a listen/announce protocol, such that it responds to a client request by returning an XML representation of the monitored data. The metrics which gmond handles are of two types: built-in metrics and user-defined. The built-in metrics relate to the physical resources of the node. User-defined metrics could include application specific data. Due to this second possibility, Ganglia has been a candidate considered for being used with the GMM. Ganglia, however, relates machines with physical IP addresses, while some of the components we measure in the GMM cannot be identified this way, they are addressed, for instance, by identifiers from an overlay network.

6. Conclusions

Platforms, which support the execution of complex services, are needed in order to make real an on-demand scenario. In order to attract a large user and industrial community, these platforms need to have the capability to give performance feedback to users and providers.

Our performance measuring framework goes beyond the monitoring of physical resources and includes middleware and application layer metrics. These multi-level metrics are necessary, since the technical metrics from all those layers can finally be interpreted in economic terms. Such an economic evaluation of the participation in the system is needed for operating with business models.

We have shown that the implementation of such a multi-level performance measuring framework is feasible. We have applied the measuring framework to obtain performance data form several layers of the deployed middleware. We have also shown that this data is useful for observing the behavior of the system. As example for this we have shown how the balancing capacity of the system can be confirmed.

Acknowledgments. This work was supported in part by the European Union under Contract CATNETS EU IST-FP6-003769 and in part by the Ministry of Education and Science of Spain under Contract TIN2006-5614-C03-01.

References

1. Ardaiz, P. Chacin, I. Chao, F. Freitag, L. Navarro . “An Architecture for Incorporating Decentralized Economic Models in Application Layer Networks”. International Journal on Multiagent and Grid Systems. Special Issue on Smart Grid Technologies. Volume 1, Number 4 / 2005, pp. 287 – 295.
2. Diet Agents Platform, February 2007. http://diet-agents.sourceforge.net/
3. T. Eymann, M. Reinicke, F. Freitag, L. Navarro, O. Ardaiz, P. Artigas. “A hayekian self-organizing approach to service allocation in computing systems”. Advanced Engineering Informatics, 19(3): 223-233 (2005). Editorial: Elsevier.
4. F. Freitag, J. Caubet, J. Labarta, “On the Scalablitiy of Tracing Mechanisms”. Euro-Par, Paderborn, Germany, August 2002.

5. Globus Toolkit, February 2007, http://www.globus.org/
6. Grid Market Middleware (GMM) http://recerca.ac.upc.edu/gmm/
7. Dan Gunter, Brian Tierney: NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging. Integrated Network Management 2003: 97-100
8. Aaron Helsinger, Richard Lazarus, William Wright, John Zinky. "Tools and techniques for performance measurement of large distributed multiagent systems". Second International Joint conference on Autonomous agents and multiagent systems (AAMAS), pp. 843 – 850, Melbourne, Australia, 2003.
9. L. Joita, O. F. Rana, Pa. Chacin, I. Chao, F. Freitag, Leandro Navarro, O. Ardaiz. "Application Deployment on Catallactic Grid Middleware". IEEE Distributed Systems Online, vol. 7, no. 12, 2006, art. no. 0612-oz001.
10. Project JXTA , February 2007, http://www.jxta.org/
11. Matthew L. Massie, Brent N. Chun, and David E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience". Parallel Comput-ing, Vol. 30, Issue 7, July 2004.
12. PlanetLab. http://www.planet-lab.org/
13. C. Preist, M. van Tol, "Adaptive agents in a persistent shout double auction. In Proceedings of the First international Conference on Information and Computation Economies (IEC), Charleston, South Carolina, United States, 1998.
14. Ioan Raicu, Catalin Dumitrescu, Matei Ripeanu, and Ian Foster. "The Design, Performance, and Use of DiPerF: An automated DIstributed PERformance evaluation Framework". Journal of Grid Computing. Volume 4, Number 3 / September, 2006.
15. M. Reinicke, W. Streitberger, T. Eymann, M. Catalano, G. Giulioni. "Economic Evaluation Framework of Resource Allocation Methods in Service-Oriented Architectures". Proceedings of the 8th Conference on E-Commerce Technology (CEC06), San Francisco, 2006.
16. Web Services Agreement Specification (WS-Agreement), 2005/09. www.gridforum.org/Public_Comment_Docs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf

Appendix: Metrics pyramid [15[