*Collaboration and the Knowledge Economy: Issues, Applications, Case Studies*
*Paul Cunningham and Miriam Cunningham (Eds)*
*IOS Press, 2008 Amsterdam*
*ISBN 978–1–58603–924-0*

# An SLA Framework for the GT4 Grid Middleware

Igor ROSENBERG[1], René HEEK[2], Ana JUAN[1]
[1]*Atos Origin, Avenida Diagonal, 200, Barcelona 08016, Spain*
*Email:igor.rosenberg@atosorigin.com, ana.juanf@atosorigin.com*
[2]*HLRS, Nobelstraße 19, Stuttgart, 70550, Germany*
*Tel: +49 711 68560442, Fax: + 49 711 65832, Email: heek@hlrs.de*

**Abstract:** Service Level Agreements are probably the most important documents in every business-aware framework –at least they should be. Every distributed piece of software faces the problem to assure a decent Quality of Service – no matter, if these agreements are legally binding contracts or not. Even though many Grid research or production projects accept the importance of SLAs and even though a specification for describing agreements and related services for SOA is emerging, most do not provide implementation supporting the negotiation and handling of SLAs. Unless the full integration of the complete lifecycle of SLAs is reached, fundamental business requirements cannot be fulfilled and critical barriers of Grid adoption cannot be overcome. Besides, it has to be considered that SLAs are of vital importance for new Grid business models, such as Utility Computing, SaaS or RaaS, due to the requirement for observing a certain QoS when providing a service. Within the BEinGRID project, confronted with this situation, we decided to produce an initial implementation of a comprehensive SLA Framework based on the Globus Toolkit.

**Keywords:** Service Level Agreement, Globus Toolkit, framework

## 1. Introduction - The BEinGRID Approach

BEinGRID – Business Experiments in Grid - is an ICT FP6 project of initially 75 partner organisations. The main objective of BEinGRID is to foster the adoption of Grid technologies for businesses and thereby crossing the chasm between the early market dominated by few visionary customers and the mainstream market dominated by a large number of pragmatic customers.

The BEinGRID project originally released 18 so called Grid Business experiments (BEs). Based on a clear business case, each BE developed a prototypic implementation for their specific requirements. The number of BEs was extended to 25 during Spring 2008. The extraordinarily high number of real business scenarios was the basis for a comprehensive field gap analysis, based on real needs expressed by potential users.

The BEs as a whole were analysed within technical cross-activities called clusters. The clusters focussed on a single specific aspect of Grid technologies. Members of the SLA cluster in cooperation with members of the architectures and interoperability cluster have developed the results presented in this article (see section 2).

In section 3, the architecture of BEinGRID SLA Framework for GT4 is presented and discussed. The BEinGRID SLA Management Framework is compared to the TrustCoM, Akogrimo, and AssessGrid SLA frameworks in section 4. Business benefits for the adoption of the BEinGRID SLA Framework are described in section 5. Conclusions, lessons learnt and future work can be found in section 6.

## 2. SLA Common Technical Requirements, Common Capabilities, Design Patterns and Components – The BEinGRID SLA Approach

Always based on the analysis of the BEs, the approach of BEinGrid was to first extract requirements, define corresponding functionalities, then generic architectures, and finally propose components.

### *2.1 SLA Common Technical Requirements*

The SLA cluster of BEinGRID, specialised in the SLA field, identified common technical requirements, based on the analysis of specific challenges presented by the BEs. These common requirements capture the essence of several challenges mentioned by one or more BEs. The final list of requirements (see [1]) for the SLA cluster in order of appearance in the SLA life cycle is:

a) SLA Template Specification: For a resource provider, a clear step-by-step procedure describing how to write an SLA template to provide with correct (and possible legal) service description
b) Publication and Discovery: Publish the provider offer, the customer QoS needs, and browse/ compare offers in a federated marketplace
c) Negotiation: Bargain-like transaction to agree SLA conditions between the customer and the provider.
d) Optimization of Resource Selection: Optimal resource management on the provider side (selection of the most suitable host) improving the current scheduler solutions.
e) Monitoring: Provide measures of the ongoing process, i.e. system values related to the SLA for internal and external usage
f) Evaluation**:** Comparing all the terms of the signed SLA with the metrics provided by the monitoring, in order to internally prevent upcoming violations and to externally discover potential violations
g) Re-negotiation: Changing the terms of an already accepted (enforced) SLA
h) Accounting: Charging the consumer for the use of services contracted by signing SLAs

The common technical SLA requirements, which correspond to needs during given periods of the SLA life-cycle, have been prioritised based on their business drivers and technical relevance (see more detail on the classification and the prioritisation of requirements in [1]). All requirements are equally important, but the classification is based on 18 real-world scenarios presented by the BEs.

### *2.2 SLA Common Capabilities, Design and Implementation Patterns*

The SLA requirements have been conflated into Common Capabilities (CCs), which represent a given SLA functionality. A CC is a description of a specific functionality. It represents a single or a group of technical requirements. We decided to merge the Negotiation and the Re-negotiation requirements, and the Monitoring and Evaluation.

Then CCs were refined into Design Patterns (DPs). The DPs are architecture-level documents, which describe a possible implementation of a CC. Only for the most relevant common capabilities have design patterns been produced. The SLA Template Specification is standalone in the sense that it does not require a component being developed. On the other hand, the Publication & Discovery common capability has not lead to the development of a component, even though a solution needs to be proposed - a discussion of a solution is drafted in 3.2.

BEinGrid produced software components by implementing the DPs. The components already developed or under way are:

a) SLA Negotiation for GT4
b) SLA Optimisation of Resource Selection for GRASP, and for GT4

c) SLA Monitoring and Evaluation for GRASP, and for GT4
d) SLA Accounting for GRIA and GT4

Within these, the four GT4 components form the basis for our SLA Framework.

## 3. Architecture of an SLA framework for GT4

### *3.1 Preface*

The decision to develop an SLA Framework based on the available components for the GT4 Middleware was driven by the high interest expressed by BEs in an integrated and interoperable solution. Nine out of the eighteen business experiments in the first wave are using the Globus Toolkit to build their Grid infrastructure. While writing up the different Implementation Patterns, it became evident that the GT4 middleware lacked of a general mechanism to handle SLAs, unlike other solutions like GRASP [19] or GRIA [16], which come ready with an SLA framework. Based on this strong assumption, we decided to propose a comprehensive framework, which would include the complete lifecycle of an SLA. This framework is meant to be available for direct use in conjunction with the GT4 middleware to provide sufficient SLA functionality for most cases.

### *3.2 Architectural Overview*

The main components and their interaction are shown in Figure 1. Solid boxes and arrows indicate components and interactions or core components of the framework. Dotted boxes and arrows indicate domain-specific components or interaction, which cannot be provided by such a framework in general.

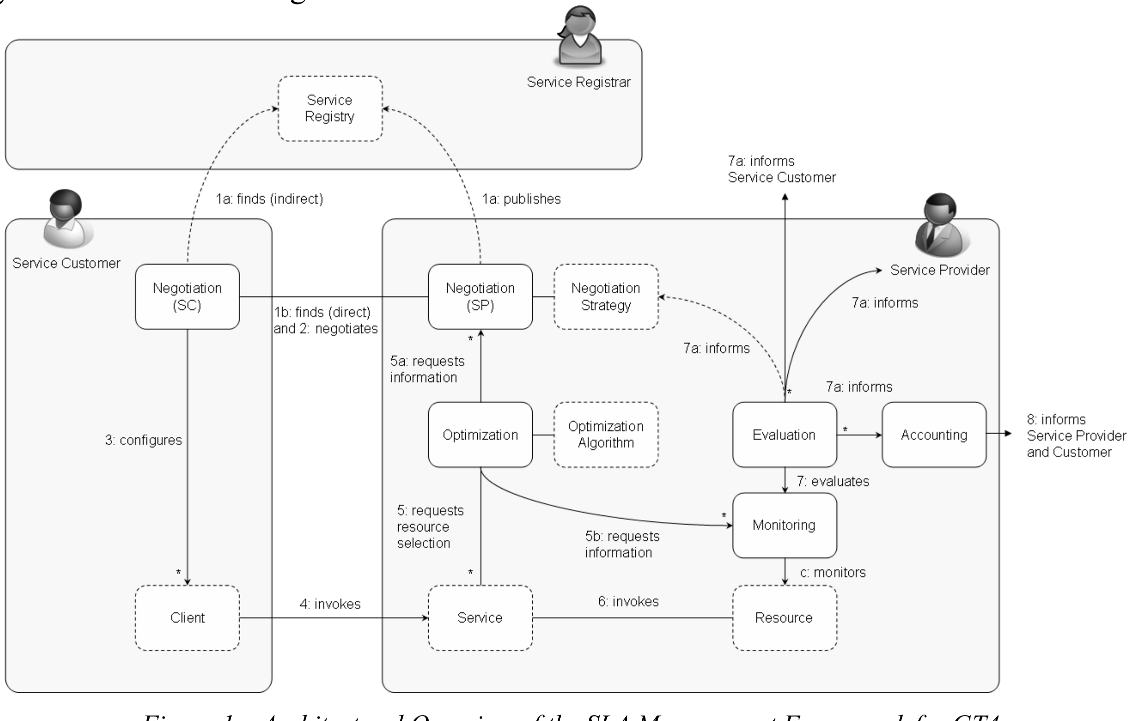

*Figure 1 – Architectural Overview of the SLA Management Framework for GT4*

The first need, which arises, is the capacity for customer (SC) and provider (SP) to discover each other. In a simple case, direct discovery, the user knows a set of providers, and queries the already known negotiation services for their templates. A more interesting approach is to consider the indirect discovery: a Discovery and Publication component should permit searching for required services, and advertising them. A marketplace or repository system can be envisaged, including advanced search, which should fuzzily match the service providers that offer corresponding services. A big technical issue comes from the fact that since the offer can be very broad-ranged (Grid being about any type of resources, not only computing power), categorizing the offers is problematic [2]. The client interface presents the same problem, as this interface should be generic enough to allow new service integration dynamically, while enabling intuitive search for the user. Related work in this area is covered in the AssessGrid [3], NextGRID [4] and Akogrimo [5] projects (see section 4). A good comparison on various types of service discovery concepts is discussed here [6].

The Negotiation component implements most of the interfaces suggested by the WS-Agreement [7] specification. Only some XML terms of the SLAs documents in WS-Agreement have been removed due to the incompatibility of these terms with Axis 1. The component supports the synchronous as well as the asynchronous negotiation of agreements and therefore provides a high-level of interoperability within the framework and for external clients. An easy plug-in mechanism for domain-specific negotiation strategies allows service providers to adapt the component to their requirements, e.g. respect the information provided by the evaluation component. The same plug-in mechanism for integrating domain-specific implementations of SLA template and SLA repositories is used. Implementations of file-based repositories are available. Related work was taken out by the AssessGrid project [8] and the implementations have been peer reviewed by members of each project.

The Resource Optimization component uses the agreed SLA and the Grid resources information to select the most appropriated resource where to run the job. The optimization algorithm can be customized to adapt to a particular application domain, or provider business rules.

The Monitoring and Evaluation component is in charge of controlling the execution of an SLA. Each resource is monitored, and its metrics are sent to a centralizing point. The information is then archived in a database (to keep records against litigations, but also for accounting). In parallel, the guarantees of the SLA are evaluated against these metrics. Violations and threats (when metrics break a warning threshold) are sent as notifications, for example to the service consumer and the service provider. Different notifications will correspond to different levels of implication in the provider's Grid infrastructure. The component has a set of evaluation rules, which define what functions to apply to discover violations and threats. The proposed implementation relies on the Ganglia [17][18] framework to monitor resources, and offers a bridge to store the metrics in a database. It also offers the notifications as WS-Notifications, with different topics corresponding to different confidence levels.

The SLA Accounting component retrieves from a database the metrics corresponding to the monitoring of the services used. This component then prepares a draft of a billing sheet, based on the price and penalties exposed in the SLA. The official financial department of the provider company must produce the real bill.

### *3.3 Component Interactions*

Customers can discover SLA templates either indirectly (step 1a of Figure 1) or directly (step 1b). With respect to the existing BEs, we are currently focussing on realising the direct discovery approach. Customers can find templates by querying already known negotiation services for templates directly. In principle, customers could discover templates indirectly by querying a registry either for endpoint references to negotiation services or for SLA templates, depending on the type of registry.

After discovering potentially several SLA templates, a customer will select one of them and start to negotiate (step 2) the final SLA with the service provider. WS-Agreement supports an asynchronous negotiation mode, in which the provider will inform customers if he accepts or rejects an SLA offer. In order to support domain-specific negotiation strategies, the negotiation component implements a simple plug-in mechanism for different strategies (e.g. interacting with the service provider). We foresee automatic negotiation strategies, which make use of monitoring and evaluation data (step 7a). In this case, the implementations of these strategies must register for notification by using the WS-Notification interface exposed by the negotiation and monitoring component.

After a successful negotiation of an SLA, two actions happen simultaneously. The negotiation component of the service provider configures the SLA Monitoring and Evaluation component. On the customer's side, the negotiation component must configure the client stub appropriately, allowing the service provider to identify the SLAs for subsequent service invocation (step 4). This can by realised by using an endpoint references containing an SLA identifier.

During a clients service invocation (e.g. a job submission), the optimisation component will make an optimal selection of resources for the service provider based on the negotiated SLA (step 5). When choosing a resource, this component will request information about the SLA for this specific service invocation by using the SLA identifier to query the negotiation component (step 5a). In addition, the optimal resource selection will be based on the current resource states. The optimization component can query the monitoring and evaluation component for further information about available resources (step 5b). The optimization component will send back a list of available and optimal resources to the service, which selects one of them and connects to this resource (step 6).

The resources, continuously monitored (step c) by the monitoring component, produce metrics stored in a database. The evaluation component evaluates this data with respect to an SLA and informs interested entities about relevant events (e.g. an SLA violation). The database also serves the accounting component, which upon receiving notification from the evaluation, starts operating on these historical data sets, before informing the financial department with a pre-bill.

## 4. Comparisons to other SLAM Frameworks

Comparing our proposal of an SLA framework for GT4 to other results, it becomes evident that it is a very lightweight implementation. This reflects the inversed history of origins of this framework. A comparison of the frameworks studied is presented in Table 1.

The decision for implementing an SLA framework in other projects was made at the very beginning in most cases, sometimes even before the project starts (e.g. during the formulation of the proposal). The BEinGRID approach made it possible to refer to the actual needs of BE and develop firstly components to support them technically in achieving their business requirements by developing common components.

The proposed SLA Framework for GT4 shows similarities with the TrustCoM Contract Management Framework [9]. In terms of components, one could find similar components in the SLA Management of TrustCoM (e.g. the negotiator) in most cases. However, they main difference between the framework described here and the TrustCoM framework is the scope of application. A main objective of TrustCoM was to establish trust between partners in a virtual organisation. Therefore, some SLA components of TrustCoM can be rated as redundant or less important for simpler use cases where a virtual organisation is not established explicitly (e.g. the Notary component and related components, like the SLA Repository). On the other hand, some components, which are represented in the BEinGRID framework, are of higher if not highest importance for business-oriented pilot projects. One example for such a component is the SLA accounting module. Considering the scope of BEinGRID, this seems not to be very surprising: the main objective of each experiment is (potentially) making money. Another point worth mentioning in comparison to TrustCoM is that a majority of BEs have chosen GT4 as middleware, so there is a need for a framework, easily integrated with GT4. The TrustCoM implementations for the SLA management are based on .NET and the Windows Communication Foundation (WCF).

Another compared framework has been the Akogrimo SLA subsystem [5]. It is split into layers, SLA High level services, in charge of the SLA Negotiation phase, and the SLA Enforcement layer, in charge of controlling the execution of a service. In the Akogrimo SLA subsystem, the negotiation interaction protocol follows the WS-Agreement specification adding a final interaction with the Execution Manager to select the most suitable service, in terms of QoS parameters. This latter interaction offers the same functionality as the Optimization component in the BEinGRID framework. Akogrimo uses business high-level terms related to a specific application domain to establish the SLA between service provider and service consumer. The Akogrimo Translator Component makes the translation between the business terms and QoS parameters (CPU usage, memory usage, bandwidth …). At runtime, the Akogrimo SLA Enforcement layer collects QoS parameters about the execution of the services. These measures are sent to the SLA Decisor module that evaluates them with the thresholds defined in the contract and the system policies, to take the appropriate corrective actions. The interactions that occur at run-time are similar in the BEinGRID framework and the Akogrimo SLA subsystem. The BEinGRID negotiation component allows an easy plug-in of domain-specific negotiation strategies respecting the information provided by the Evaluation component. The Akogrimo SLA subsystem does not provide an Accounting component.

| **Comparison of SLA capabilities** | | **BEinGRID GTv4 Framework** | **TrustCoM Contract Management Framework** | **Akogrimo SLA Subsystem** | **Assesgrid SLA Architecture** |
|---|---|---|---|---|---|
| BEinGRID identified SLA capabilities | SLA Template specification | ✗ | ✗ | ✗ | ✗ |
| | Publication and Discovery | ✗ | ✓ | ✓ | ✗ |
| | Negotiation | ✓ | ✓ | ✓ | ✓ |
| | Optimization of Resource Selection | ✓ | ✗ | ✓ | ✓ |
| | Monitoring | ✓ | ✓ | ✓ | ✓ |
| | Evaluation | ✓ | ✓ | ✓ | ✓ |
| | Re-negotiation | ✗ | ✓ | ✗ | ✗ |
| | Accounting | ✓ | ✓ | ✗ | ✗ |
| Implementation Technology | | GTv4 | .Net, WCF | .NET, GTv4 | GTv4 |

*Table 1: Comparison of capabilities of different SLA frameworks*

AssessGrid [3] includes risk management mechanisms to the SLA management in order to move beyond the best-effort approach in the service provision. It considers that negotiating SLAs is a business risk for both the Service provider and the Service Consumer. The AssessGrid architecture considers three roles; the service provider, the broker and the customer. It considers that an SLA must exists between all parties. The life-cycle of the SLA exposed in AssessGrid is the classical vision also followed in the BEinGRID framework. The SLA negotiation follows the WS-Agreement protocol with the restriction that the initiating party is always the customer. The negotiation includes an interaction with the Consultant service to estimate the risk of SLA failure (PoF). The SLA template is based on the standard WS-Agreement structure describing the service terms using the JSDL HPC profile [11]. As in the BEinGRID SLA framework, the AssessGrid architecture does not provide a mechanism to Publish and Discover SLAs. The functionality of selecting the most appropriated resource in which to run a job, provided by the Optimization component in BEinGRID framework, in AssessGrid is performed by Scheduler component that also takes into account potential risk. AssessGrid uses Nagios daemons in the Monitoring services to monitor the Grid resources while BEinGRID uses Ganglia with the same intention.

## 5. Business Benefits

While research and academic institutions are interested in Grids that provide access to a higher computing performance to satisfy peak demands and support to face collaborative projects, enterprises understand grid computing as a way to address the changing service needs in an organization. Terms such as On-demand and Utility Computing are related to the Enterprise view of Grid [12]. Utility-computing is defined as the capacity to provide computing power as it happens at the moment with other facilities such as water or power, billing it in a pay-per-use model. On-demand computing relates to the ability of acquiring additional resources to meet changing requirements. In both terms, there is a clear separation between the Service provider and consumer and the ability to negotiate a desired quality of service for a determined period. Grid technology, by empathising the requirement of solid SLA management system, is an enabler of this IT provisioning methods.

Grid infrastructures aim to attract and enable new businesses and radically change the relationships between a customer and its supply chain, offering a flexible platform for a global collaboration. Grids can improve industry competitivity by better utilizing heterogeneous resources virtualized as services. Even more, innovative grid-enabled applications can be offered in a SaaS (Software as a Service) manner to new customers creating new business models. However, in all these new business models the Grid is capable to offer are sustained by the ability to establish, negotiate, monitor and evaluate a service level agreement taking part between the service consumer and the service provider.

Recently the convergence and interaction of Grid, SOA and Virtualization as well as Cloud Computing [13] techniques have been largely discussed. All these business models, SaaS, On-demand, Utility and Cloud computing, share the same requirement to be wide adopted in enterprise environments, the need of the establishment of SLAs [14][15] to assure that the business applications meet the required performance benchmarks.

## 6. Conclusions, lessons learnt and future work

In this paper, we started by presenting the BEinGrid requirement analysis performed on real scenarios (BEs). This provided generic technical requirements (needs expressed in a given scenario), common capabilities (a functionality which answers this technical need), design patterns (the middleware-agnostic architecture needed to provide this functionality), and components (middleware-specific implementations). Based on the gap-analysis, we could justify our proposition of an architecture of a framework for SLAs on the GT4 middleware. This framework is an integrated answer to the needs stressed by the BEs. It is based on components: negotiation, optimization, evaluation & monitoring, and accounting, of which functionalities and interactions were shown. We also highlighted interoperability issues that came up with the component-based approach. This architecture covers the whole lifecycle of an SLA, minus the initial discovery & publication phase. We also compared to other FP7 projects frameworks, namely TrustCoM, Akogrimo, and AssessGrid, highlighting the weaknesses and strong points of each.

The work presented is a software architecture, which was missing in the GT4 middleware. It was discovered that a comprehensive framework, including details for all steps of the life-cycle of an SLA, is not currently required. The basic building blocks are sufficient, as the market is not mature enough to accept open competition and assessment. So a simple implementation based on the core functionalities is proposed to help grid adoption; it will benefit all parties by including SLAs in their business cycle.

Our next step will be to complete our first simple implementation of the proposed architecture, and validate it in real business situations. We are also planning to stress the interoperability problems raised by different middleware stacks, and their solutions (namely, offering interfaces with GRASP and GRIA). We will also re-use existing BEinGrid business analysis to specify in detail the market opportunities of our solution. All this will be done in collaboration with the second wave of BEs, which have the purpose of validating the implementation results of BEinGrid.

## References

[1] Design patterns for SOA and Grid. T. Dimitrakos et al., BEinGRID AC1 Meta-Deliverable, 2007

[2] Interoperability and Reuse with WS-Agreement. A. Andrieux, K. Czajkowski, 2004, http://www-unix.mcs.anl.gov/~keahey/Meetings/GRAAP/karl.pdf

[3] Introducing Risk Management into the Grid, Djemame, K.; Gourlay, I.; Padgett, J.; Birkenheuer, G.; Hovestadt, M.; Odej Kao; Voss, K. e-Science and Grid Computing, 2006. e-Science apos;06. Second IEEE International Conference on Volume , Issue , Dec. 2006 Page(s):28 - 28

[4] Towards Autonomous Brokered SLA Negotiation. P. Hasselmeyer, C. Qu, L. Schubert, B. Koller, P. Wieder, Exploiting the Knowledge Economy - Issues, Applications, Case Studies. Volume 3, October 2006

[5] An Enhanced Strategy for SLA Management in the Business Context of New Mobile Dynamic VO, D'Andria, F., Martrat, J., Laria, G., Ritrovato, P., Wesner, S., In Exploiting the Knowledge Economy: Issues, Applications, Case Studies, Paul Cunningham and Miriam Cunningham (Eds), IOS Press, Amsterdam, 2006

[6] On Service Discovery Process Types. P. Hasselmeyer, 3rd International Conference On Service Oriented Computing (ICSOC '05), Amsterdam, The Netherlands, December 2005. Springer-Verlag, LNCS 3826, ISBN 3-540-30817-2, pp. 144-157, December 2005

[7] Web Services Agreement Specification (WS-Agreement). A. Andrieux et al., Specification from the Open Grid Forum (OGF), 2007

[8] Implementing WS-Agreement in a Globus Toolkit 4.0 Environment. D. Battré, O. Kao, K. Voss Usage of Service Level Agreements in Grids Workshop in conjunction with The 8th IEEE International Conference on Grid Computing (Grid 2007), September 2007

[9] TrustCoM Framework V4, M. Wilson, A. Arenas, L. Schubert, Deliverable of the TrustCoM EU FP6 project, 2006, http://213.27.211.106/trustcom/wp-content/uploads/2007/08/d9+trustco.pdf

[10] A Comparison of SLA Use in Six of the European Commissions FP6 Projects, CoreGRID Technical Report Number TR-0129, M. Parkin, R. M. Badia, J. Martrat, http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0129.pdf

[11] JSDL HPC Profile Application Extension, Version 1.0. M. Humphrey, C. Smith, M. Theimer, and G. Wasson., V1.0.

[12] The Different Faces of IT as a Service, I. Foster, http://www.ogf.org/documents/Diff_Faces_foster.pdf

[13] Cloud Computing, Greg Boss, http://www.ibm.com/developerworks/websphere/zones/hipods/

[14] SLAs A Key Commercial Tool. B. Mitchell, P. Mckee, Innovation and the Knowledge Economy: Issues, Applications, Case Studies, 2005

[15] The increasing role of SLAs in B2B. P. Masche, B. Mitchell, P. Mckee, Proceedings of the 2nd international conference on web information systems and technologies, Setubal, Portugal, April 2006

[16] GRIA Website, http://www.gria.org/

[17] Ganglia Website, http://ganglia.info

[18] The ganglia distributed monitoring system: design, implementation, and experience. M. L. Massie, B. N. Chun, D. E. Culler, Parallel Computing 30 (2004) 817–840

[19] The GRASP project http://www.eu-grasp.net