# Dynamic Service Provisioning Using GRIA SLAs

Mike Boniface, Stephen C. Phillips, Alfonso Sanchez-Macian, and Mike Surridge

University of Southampton IT Innovation Centre,
2 Venture Road, Chilworth,
Southampton, SO16 7NP, UK
{mjb,scp,asm,ms}@it-innovation.soton.ac.uk
http://www.it-innovation.soton.ac.uk

**Abstract.** Service Level Agreements (SLA) include quality of service (QoS) constraints and bounds that have to be honoured by the service provider. To maximise the Service Provider revenue while satisfying the QoS requirements of the agreed SLAs it is important to be able to perform a dynamic distribution of the service provider resources between the services and SLAs. This distribution should be based on the current status and predicted evolution of the QoS characteristics. This paper describes the experiences managing SLA obligations from a service provider perspective in a scenario where dynamic deployment of services can be undertaken. The main issues faced to deal with the management of SLAs in this context are detailed. The adopted solution, based on GRIA (a Service Oriented Architecture framework) is discussed.

**Keywords:** SLA management, dynamic provisioning.

## 1 Introduction

Businesses designing and developing complex products are facing increasing market pressure to reduce product development cycles whilst improving capabilities and compliance with ever rigorous safety regulations. Traditional business structures and markets are fragmenting as companies continue to specialise through increased focus on core competencies. Outsourcing of previously core functions (product design, analysis codes, etc) is seen as a key strategy that will allow businesses to meet their goals and entire design supply chains are emerging. This has led to great interest in technologies such as service oriented architectures (SOA), Grid computing, and software as a service (SAAS) offerings. Businesses are anticipating that such technologies will provide flexible and adaptive infrastructures that will allow them to extend their enterprise and to inject innovation into their product design processes in a way that is secure, accountable, auditable and cost-effective.

In this paper, we present strategies that allow service providers to offer inter-enterprise service provision using SLAs. We explore how Web Service management using SLAs and dynamic service provision can maximise resource utilisation whilst satisfying quality of service (QoS) commitments to existing

E. Di Nitto and M. Ripeanu (Eds.): ICSOC 2007 Workshops, LNCS 4907, pp. 56–67, 2009.

customers. We describe the characteristics of such an SLA management system and how we have adapted the GRIA middleware to meet the needs of dynamic provisioning scenarios.

## 2 Web Service Management with SLAs

SLAs are at the cornerstone of business transactions and describe the QoS and other commitments by a service provider in exchange for financial commitments by a customer against an agreed schedule of prices and payments. SLAs are now widely considered a central component for managing extended enterprise business automation and many research initiatives within the EC 6th Framework Programme have investigated the role SLAs have in Web Service management. The NextGRID project [1] proposes the use of bipartite (two-party) SLAs to describe both the functional and non-functional aspects of a service to allow consumers to make informed choices. The NextGRID SLA is an agreement (as denoted by its name) and describes the obligations each each of its signatories must do in order to be compliant with the agreement. NextGRID SLAs have a strong linkage to business impact for both the customer and provider so that the customer can assign and manage costs transparently within their business and the service provider can retain the flexibility in operations to strive to reduce the cost of delivery and ultimately the price paid by the customer [2]. In the EU IST SIMDAT project [3], an SLA management service has been developed for GRIA to meet the demands of distributed product development teams needing to share IT assets [4]. GRIA SLAs have some of the characteristics of NextGRID SLAs, for example, bi-partite agreements and high-level business value exchange linked to low-level resourcing but GRIA focuses on capacity management rather than dynamic provisioning decisions.

Both SIMDAT and NextGRID views contrasts with the use of SLAs in the more established academic Grid community where the SLAs described in [5], [6] and [7] focus on resources (e.g. computers, network bandwidth, storage devices) rather than services. This is of course appropriate for a community of experienced users running often experimental codes. However, in order to address the needs of business users (as opposed to technical users) the value of the service must be articulated at the appropriate business level rather than the resource level. The customer should not be concerned with the resources required to provide the service just that the service exists and provides a clear business benefit. It is also important to consider SLA information when composing services to provide an integrated business solution. NextGRID uses a workflow enactor to deal with this issue, but other alternatives have been studied. For example, in [18] an agent-driven architecture is used to manage individual QoS commitments and map them to the end-to-end QoS constraints. The business perspective of QoS is also investigated in [8] where client and service providers can define their QoS requirements and negotiation strategies. Unfortunately, the defined architecture is based on the use of a negotiation broker, which conflicts with traditional supply chain approaches.

Previous work has been done in dynamic deployment of web services and SLA implications of these deployments. In [9], some predictive and reactive approaches are suggested based on mathematical formulae and the use of heuristics. Finally, an architecture based on the Web Service Level Agreement (WSLA) language is proposed in [10], where dynamic provisioning is also commented. That work has some interesting ideas, but many of the issues commented on in section 3 below are not addressed in that paper, such as the transparency to the user or the different ways of sharing resources between different SLAs (an inter-SLA vision of the problem).

The following sections describe an approach for service provider management that allows a service provider to offer high-level SLA metrics whist incorporating strategies to manage SLA violations, mapping high-level SLA metrics to measured metrics and aggregation of QoS measurements across multiple customers.

## 3 Issues in Managing Dynamic Provisioning and SLAs

Using dynamic provisioning of services helps the service provider to manage their resources in a way that he is capable of honouring the SLA requirements for the contracts he has signed. The possibility of deploying service instances dynamically allows a service provider to define policies to optimise the distribution of resources against SLA commitments and therefore increase the number of agreements with their customers. Additionally, it is also possible to outsource tasks when SLA commitments cannot be met by adaptively procuring additional resources from other service providers However, some issues need to be solved in this situation which are presented in the following subsections.

### 3.1 Management of Multiple Services and Multiple SLAs

The use of dynamic provisioning by a service provider must be transparent to the service consumer. This means that from a users perspective, the service has to behave in a similar way and the functionality has to be provided as if a standalone service was used. Thus, the capability of deploying services dynamically and the actual number of services deployed have to be transparent to the software used by the consumer and the terms included in the SLA should not be changed.

When trying to honour an agreed SLA, the dynamic deployment of new service instances can help to prevent QoS constraint breaches. During the provisioning of a service, the QoS is monitored and checked against the constraints defined in the SLA. If a service provider detects that a future obligation is unlikely to be met (e.g. the maximum job queue time for a job execution), the service provider can decide to deploy an additional instance of the service in a new machine to manage the situation. Of course, the challenge is more complex as service providers need to manage multiple SLAs that may have conflicting resource requirements and potential QoS breaches resulting in different business impact. Service providers (or the policies defined in their software) have to decide how to distribute or share resources to minimise costs in this situation.

Usually, there is a many-to-many relationship between SLAs and services. A service (e.g. a Job Service) can provide functionality to users under different SLAs and one SLA can include agreements to use several different services (e.g. a Job Service and a Data Service) from a service provider. In a dynamic provisioning scenario, a service is provided by several service instances in a transparent way. Although the service can still be used under different SLAs, a decision must be made about how service instances are managed. They may be used to provide the functionality for a specific SLA in an exclusive way or they can be shared amongst different SLAs. Additionally, it is necessary to consider the capabilities of each machine to identify how many service instances can coexist at the same time. The decisions affect the policies that a service provider can use to honour the SLAs.

The first strategy is to allocate service instances to many SLAs. The main issue to consider with this strategy is that it is necessary to distinguish between invocations related to different SLAs. The service provider needs to monitor QoS parameter usage against each specific SLA to manage violations and for load balancing. There are three policies that can be applied using this strategy a) deploy new service instance b) promote invocations or resources among SLAs, giving priority to the invocations related to the compromised SLA, and c) redirect requests or migration of resources between service instances. In a), the new service instance should be deployed on a different machine where no service instance of this type is available, as it is likely that the problem is related to the performance of the hardware. b) can be implemented using queues, but c) is far more complex. In c) migrating resources between service instances requires state and context to be stored and moved to a new location. All these policies must be triggered by a component that is able to receive the QoS information and match it against the SLA constraints.

The second strategy is to make the service instances restricted to a particular SLA. QoS can be monitored on a per-instance basis. The deployment of service instances, in this case, can be done in a machine where other instances of that type are already deployed if they are not assigned to the same SLA and the machine is able to cope with the new instance. With the per-SLA distribution of instances, the alternative available policies are different. Migration would be still possible between two instances assigned to the particular SLA, but promoting invocations in a service instance is not a solution as all the requests are related to the same SLA. In this situation, it is possible to prioritise between the service instances deployed in the same machine and create mechanisms for pausing or undeploying some service instances, giving priority to those related to a compromised SLA.

For both strategies the machine capabilities must be monitored and the overall capacity pre-defined. Some heuristics can be used to decide how many service instances of a specific type can reside in the same machine. As different service types have different requirements, it would be necessary to define a way of sharing the machine between them such as defining an equivalence between number of service instances of two different types.

### 3.2 Strategies to Manage SLA Violations

From a service providers perspective, there are two possible policy enforcement strategies for handling SLA violations: prevention before violation or reaction after violation. The prevention strategy is based on predictions of a probable SLA violation. The prediction can come from some prevention thresholds for the QoS characteristics being monitored or more complex algorithms derived from previous experiences. For dynamic provisioning, when the prevention threshold is exceeded, a new service instance is deployed so the load balancing algorithm can redirect the requests to the new address. The thresholds policy have to be defined on a per-SLA basis. One approach is to treat thresholds as private constraints using the same representation (standard web service language or an ontology-driven pattern) as agreed constraints within the SLA. The reaction strategy is only acceptable if violations result in service provider penalties rather than SLA cancellation. The service provider can decide that it is better to accept violations in some SLAs to give priority to others based on business impact. Different level of penalties can be defined in a SLA depending on the specific violation or the affected QoS metrics. If this possibility is considered, then the SLA framework must be flexible enough to define these terms. Both of violation strategies are not exclusive as a service provider could decide to adopt prevention for some SLAs and even specific QoS metrics while applying reaction for others. Even if the prevention strategy is applied, not all the situations can be overcome and a combination of both could be needed or preferred depending on the penalties applied.

### 3.3 Mapping SLA Metrics to Measured Metrics

The metrics defined in an SLA can be different from those measured in the services so that a provider offer service at an appropriate business level for their customers. In this case the measured metrics must be translated in a way that can be checked against the SLA. Metric translation applies to both single services and multiple service instances, however, an additional translation is required for multiple instances to aggregate measurements of the same metric.

It is also important to have a method for mapping different metrics. There is previous work in the Semantic Web community that can be reused. Oldham [11] focused on the negotiation of SLAs combining Semantic Web technologies with the WS-Agreement standard [12]. In [13] some mechanisms using OpenMath and Semantic Web ontologies and rules were defined to map QoE to QoS metrics. Dobson [14] defines a QoS ontology for service centric systems that can represent the required concepts. Those approaches would be applicable once a suitable implementation is available. The definition of the mapping between metrics can be stored in a global repository at the service provider or described in the SLA.

The operations needed to aggregate usage from different service instances to get the final usage for the whole service depends on the way the metric is calculated. If every instance performs the calculation of these metrics and send the information to a central factory which aggregates them, some problems can

arise when trying to do this aggregation. For example, when using the "average job time in queue" metric, every service instance could send the individual value for that metric, but to get a real aggregated value you need to know also the number of jobs per instance to be able to weight each individual value. On the other hand, metrics like "number of jobs queued" can be easily computed by simply adding the individual values coming from the instances. A possible solution for this aggregation is to defer the calculation of the metrics for all the instances and doing them in a centralised point (e.g. an SLA management service).

## 4 The GRIA SLA Management Service

### 4.1 Introduction

The GRIA SLA Management Service, developed in the EU IST SIMDAT Project, performs functions related to the process of agreeing, monitoring and enforcing an SLA between the customer and the service provider:

1. It provides access to service level agreement "templates" that can be filled in and submitted by a potential consumer as service level "proposals";
2. It decides whether to enter into a new service level agreement (with a given QoS) when proposed by a consumer, and responds accordingly;
3. It decides whether a requested application service "activity" is covered by an existing service level agreement, and detects when such an activity (even if covered) would exceed the capacity of the provider;
4. It decides which service level agreement(s) should be breached when capacity is about to be exceeded, and initiates relevant management actions to resolve the situation;
5. It tracks the quality of service actually delivered, and initiates charges when appropriate;
6. It detects when a consumer is exceeding the limits of a service level agreement, and initiates load reductions in the corresponding application service(s) to prevent this.

The interactions with the SLA manager are shown in 1. The service manager (or agent) is responsible for deploying application services and configuring the SLA service to manage those services. This would be done by composing and publishing SLA templates appropriate to the particular application services. A Customer can obtain any of the templates made available to them, fill it in, and send it to the SLA manager as an SLA proposal. At this point the SLA manager may accept the request, and start using the proposed SLA to manage services, or refuse the request. If accepted, the SLA manager will respond with a WS-Addressing Endpoint Reference (EPR) including a context reference for the SLA. The Customer must include this reference in the context header for subsequent requests to the SLA manager related to this SLA.

The application service shown in Figure 3 can be any Web Service that requires management within the context of a procurement process. The interface

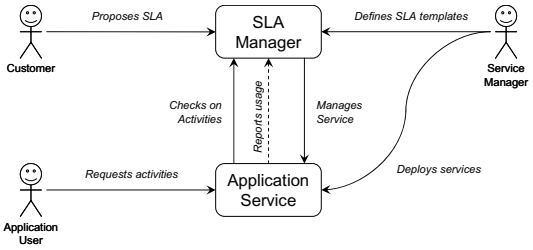


**Fig. 1.** Interactions between the GRIA SLA Manager, an application service and other actors

between an application service and the SLA management service uses the WS-Trust and WS-Notification standards as well as a small number of custom operations. A developer kit including helper APIs is available to make application service development easy.

The Application User in Figure 3 interacts with application services provided under the terms of an SLA. The nature of these interactions obviously depends on the application service, but in general, the application service will consume resources in order to respond to the Application User. It should do so only to the level specified in the SLA, or the service provider resources may become overloaded and be unable to response to other Application Users to the required level. The Application User must specify the EPR of the SLA when initiating an interaction. If the user has no SLA, or specifies an SLA that does not cover the application service interaction, or to which this user has no access rights, the application service should reject the interaction.

The application services will ask the SLA manager when they need to check if the SLA requires them to respond to a user request. The SLA manager will ask the application service to terminate an interaction if it becomes necessary to stop the service from consuming resources under a given SLA. In addition, the application service should report service usage to the SLA manager.

### 4.2 Metrics

The GRIA SLA Management Service is designed to be very flexible and to support a wide range of application services. It retrieves usage information from application services (e.g. job and data services), records the usage and optionally constrains and/or bills for the usage. Different application services will want to report usage of different measurable quantities (metrics). For example, a job service may report usage of CPU but a data service may report usage of disc space. The SLA service does not understand the meaning of each metric, it just records usage and acts according to service provider policies when usage exceeds a constraint (see below).

The use of metrics is recorded in terms of “instantaneous” measurements and “cumulative” usage. The cumulative usage is the integration of the instantaneous measurements over time. For some metrics, data-transfer for example, the

instantaneous measurement is best thought of as a rate (bytes per second) and the cumulative usage has no time dimension (bytes). For other metrics, such as CPU, the instantaneous measurement is just the quantity in use at the time (e.g. 3 CPUs) and it is the cumulative usage that has the time dimension, e.g. 180 CPU.seconds. The SLA service can convert between the "instantaneous" measurements and "cumulative" usage e.g.

- If a job runs on 1 CPU for 5 minutes then the SLA service will be notified that the instantaneous measurement of CPU usage went to 1 at the start and then to 0 five minutes later. The SLA service can infer that 300 CPU.seconds of CPU time have been used ($1 * 5 * 60 = 300$ CPU.seconds).
- If a service reported that it had used 120 units of a resource in a 1 minute period, the SLA service would infer that the average instantaneous measurement (rate of usage) had been 2 units/s.

All metrics have both instantaneous measurements and cumulative usage which may be recorded or inferred. For some metrics one or other concept will not be useful, but the SLA manager has no idea of what it is counting, restricting or billing for in each metric, and so can cope with either type of measurement and can always infer one from the other.

### 4.3 Constraints

Constraints play a major role in GRIAs SLA definitions. An SLA can contain any number of constraints, each one defining a usage limit that defines the commitment made by the service provider which the consumer is not supposed to exceed. Each constraint places a limit on a *metric* (see above), for instance:

- Amount of CPU time $\leq 50$ CPU.hours/day
- Amount of disc space $\leq 100$ GB
- Number of databases $\leq 3$
- Number of simulation timesteps $< 100000$/week
- Size of mesh $< 1000000$ elements
- Number of video frames rendered $\leq 1000$/day

The first three constraints in this list are self-explanatory and they have a trivial mapping on to typical service provider resources required to fulfill an SLA. However, in some situations a client will want to deal with a service provider in "business terms" (what the client wants to get done) rather than "technical terms" (what resources are required to do the job). For example, a client who requires frames of a video to be rendered may prefer to have an SLA that specifies how many frames of high-definition video they were permitted to render per day, rather than an SLA that described how many CPUs and how much memory etc is on offer.

To support this scenario, the concept of "private constraints" has been introduced. A private constraint is treated in the same way as a public (normal) constraint in every way except that it is not revealed to the customer. They can

be used to hide the technical detail about how the service provider will use its resources (CPU etc) and expose the business offer of interest to the customer, e.g. an SLA template could have these constraints:

- A public constraint saying "Number of video frames rendered $\leq 1000$/day".
- A private constraint saying "Amount of CPU time $\leq 50$ CPU hours/day".
- A private constraint saying "Amount of disc space $\leq 100$ GB".

In composing the SLA template, the service provider has had to make a judgement about what basic resources (CPU, disc) are required to fulfill the offer of 1000 frames of video per day. All three constraints then go into the SLA template and the template is published. When the client views the SLA template though, the private constraints are removed so all the client sees is the information they are interested in, namely how many frames they will be able to render. The SLA manager will pay attention to all the constraints though and will ensure that the service provider has sufficient capacity to meet the CPU and disc space constraints.

### 4.4 Pricing Terms

As well as constraining the usage of resources, an SLA can also (optionally) charge for resource usage. The pricing system in the GRIA SLAs is also flexible. Firstly, a billing period can be defined and the SLA manager will aggregate usage over the billing period and place a single aggregated charge onto the users account at the end of each period. Secondly, a "signing fee" and a "subscription fee" may be defined. The signing fee is paid up front on agreeing the SLA and the subscription fee is levied at the end of each billing period. Finally, metric-specific charges may be defined, either on the cumulative amount of a metric used in the period (e.g. a charge per CPU second) or on how many times a metric has been used (e.g. a charge per job executed). These metric-specific charges can also be augmented with usage thresholds, so for instance pricing terms could be:

- 0.01 \$/CPU.second in the range 0-3600 CPU.seconds
- 0.05 \$/CPU.second in the range 3600-$\infty$ CPU.seconds.

## 5 Adopted Solution

The GRIA SLA service is being adapted to meet dynamic provisioning requirements in two phases. The first phase is completed and some initial solutions have been incorporated and are being tested. In the second phase the solution will be refined, learning from the outputs of the tests being carried out. The main experiments have been achieved using a service factory and resource factory (DynaSOAr [15] and Debut [16]) to provision GRIA job and data application services which provide stateful resources accessed under the terms of an SLA.

The SLA service has been adapted to manage the dynamic deployment scenario while keeping the public SLA terms. In the scenario, users working under

different SLAs want to create and run jobs. The SLAs being used keep the constraints defined in section 4.3, such as "amount of disc space" and the pricing terms defined in section 4.4. The user only access a single entry point (see section 3.1) to request for job creation, but the actual jobs are allocated in different service instances. This means that the SLA service has now to be able to group usage monitoring reports coming from the different service instances, e.g. it has to aggregate the amount of disc space or the amount of CPU time consumed by the jobs in all the instances. Implementing this strategy was simple due to GRIA's WS-Notification [17] based monitoring architecture and the storage of that information in the SLA usage database located in the SLA service. The calculation of aggregated values is done using the information retrieved from SLA usage database.

The SLAs used in the experiment define constraints that must be honoured by the service provider. This includes constraints to keep the Average Queue Time and the Average Queue Length of jobs below a threshold during a specified time period. To manage SLA violations, prevention thresholds (see section 3.2) have been defined. For the constraint "Average Queue Length $< 4$", a private constraint has also been specified "Average Queue Length $< 2$". A service provider can then use the SLA constraint to check when an SLA has been violated and the private constraint to trigger additional deployments to prevent the violation.

To deal with the dynamic deployment and multiple SLAs, we have opted to associate each instance of the application service with a single SLA: the second strategy discussed in section 3.1. This means that the requests from different users in the experiment are directed to different service instances. When the prevention threshold is exceeded, a decision is taken depending on the management actions defined in the SLA service. These actions can trigger a new service deployment or, if no additional hosts are available, the problem can be escalated allowing the service provider to prioritise existing commitments based on predicted penalties. The service provider can suspend or force the undeployment of a service instance associated to a different SLA that is competing for the CPU or memory resources. The policies to prioritise SLAs to maximise service provider revenues are planned to be addressed in the second development phase.

Some metrics reported by service instances need to be mapped to the metrics defined within the SLA, for example, "average queue length per service". Each job service instance reports information about the time when a job starts queuing and the time when it finishes queuing. The SLA service aggregates these reports and converting them into the "number of jobs queued per SLA". Additionally, there is a second metric coming from the single entry point and providing information about the "number of job services deployed". To calculate the SLA metric these two metrics are combined using the following simple rule:

$$\text{average queue length per service} = \frac{\text{number of jobs queued per SLA}}{\text{number of job services deployed}}$$

This type of conversions has been implemented in the SLA service using a pluggable metric architecture (Figure 2). When the SLA manager needs to check a constraint it queries registered metric handlers to determine how to calculate

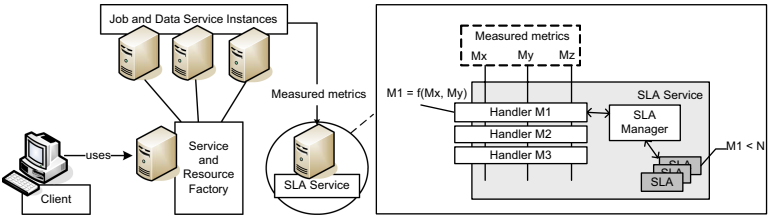


**Fig. 2.** Mapping measured to SLA metrics

current usage. If no specific metric handler is registered, it assumes that the metric maps to a measured one and calculates directly the aggregated value as explained in section 4.2. The definition of the metric translation using an adapted XML language or ontology is being evaluated for the second phase of development.

## 6 Conclusion

In this paper we have described strategies that allow service providers to use dynamic service provisioning with Service Level Agreements (SLAs) to maximise resource utilisation. Various issues have been discussed that impact service provider provisioning decisions. These issues have driven enhancements to the GRIA SLA management service allowing a wide range of application services to be dynamically deployed. We have shown that the adopted solution has been able to address most of the challenges presented in section 3 in a simple way. However, additional work needs to be done to define the business policies to prioritise SLAs and to work with competing QoS objectives. The final enhanced GRIA infrastructure will be used as a component within the UK DTI CRISP project to explore how inter-enterprise infrastructure can support engineering application service provision. The current implementation is being evaluated by industrial users to determine the impact such technology can have on improving their ability to design and develop complex products and services.

**Acknowledgments.** SIMDAT has received research funding from the European Commission under the Information Society Technologies Programme(IST), contract number IST-2004-511438.

CRISP has received research funding from the UK DTI under the technology programme, DTI project number TP/2/IC/6/S/10044.

## References

1. Snelling, D., Fisher, M., Basermann, A., Wray, F., Wieder, P., Surridge, M.: NextGRID Vision and Architecture White Paper V5, `http://www.nextgrid.org/download/publications/NextGRID_Architecture_White_Paper.pdf`

2. McKee, P., Taylor, S., Surridge, M., Lowe, R., Ragusa, C.: Strategies for the service market place. In: Veit, D.J., Altmann, J. (eds.) GECON 2007. LNCS, vol. 4685, pp. 58–70. Springer, Heidelberg (2007)
3. EU IST SIMDAT Project, http://www.simdat.org
4. Boniface, M., Phillips, S.C., Surridge, M.: Grid-Based Business Partnerships using Service Level Agreements. In: Proc. of Cracow Grid Workshop 2006, Cracow, Poland (2006)
5. Buyya, R., Abramson, D., Venugopal, S.: The grid economy. Proceedings of the IEEE 93(3), 698–714 (2005)
6. Czajkowski, K., Foster, I., Kesselman, C.: Agreement-Based Resource Management. Proceedings of the IEEE 93(3), 631–643 (2005)
7. Yeo, C.S., Buyya, R.: Service Level Agreement based Allocation of Cluster Resources: Handling Penalty to Enhance Utility. In: Proc. of the 7th IEEE International Conference on Cluster Computing (Cluster 2005). IEEE Computer Society, Los Alamitos (2005)
8. Comuzzi, M., Pernici, B.: An architecture for Flexible Web Service QoS Negotiation. In: Proc. of the 9th IEEE International EDOC Enterprise Computing Conference (EDOC 2005), pp. 70–82. IEEE Computer Society, Washington (2005)
9. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P.: Dynamic Provisioning Of Multi-Tier Internet Applications. In: Proc. of the 2nd IEEE International Conference on Autonomic Computing, pp. 217–228. IEEE Computer Society, Washington (2005)
10. Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., Youssef, A.: Web services on demand: WSLA-driven automated management. IBM Systems Journal 43(1), 136–158 (2004)
11. Oldham, N., Verma, K., Sheth, A., Hakimpour, F.: Semantic WS-agreement partner selection. In: Proc. of the 15th International Conference on World Wide Web, WWW 2006, pp. 697–706. ACM Press, New York (2006)
12. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Kakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification (WS-Agreement) GFD 107 Recommendation. Open Grid Forum (2007)
13. Sánchez-Macián, A., López, D., López de Vergara, J.E., Pastor, E.: Framework for the Automatic Calculation of Quality of Experience in Telematic Services. In: Proc. of the 13th HP-OVUA Workshop, Côte d'Azur, France (2006)
14. Dobson, G., Russell, L., Sommerville, I.: QoSOnt: a QoS Ontology for Service-Centric Systems. In: Proc. of the 31st Euromicro conference on Software Engineering and Advanced Applications, pp. 80–87. IEEE Computer Society, Washington (2005)
15. Watson, P., Fowler, C.P., Kubicek, C., et al.: Dynamically Deploying Web Services on a Grid using Dynasoar. In: Proc. of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, Gyeongju, Korea, pp. 151–158. IEEE Computer Society Press, Washington (2006)
16. Belfast e-Science Centre, www.besc.ac.uk
17. WS-Notification, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.htm
18. Chhetri, M.B., Lin, J., Goh, S., Zhang, J.Y., Kowalczyk, R., Yan, J.: A Coordinated Architecture for the Agent-based Service Level Agreement Negotiation of Web Service Composition. In: Proc. of the 2006 Australian Software Engineering Conference (ASWEC 2006), pp. 90–99. IEEE Computer Society, Washington (2006)