European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

# Dynamic SLA-negotiation based on WS-Agreement

*Antoine Pichot*
Antoine.Pichot@alcatel-lucent.fr
*Alcatel-Lucent*
*Route De Villejust*
*91620 Nozay, France*

*Philipp Wieder*
ph.wieder@fz-juelich.de
*Research Centre Jülich, Central Institute for Applied Mathematics (ZAM)*
*D-52425 Jülich, Germany*

*Oliver Wäldrich, Wolfgang Ziegler*
{Oliver.Waeldrich, Wolfgang.Ziegler}@scai.fhg.de
*Fraunhofer Institute SCAI, Department of Bioinformatics*
*D-53754 Sankt Augustin*

---

CoreGRID Technical Report
Number TR-0082
June 24, 2007

Institute on Resource Management and Scheduling

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

# Dynamic SLA-negotiation based on WS-Agreement

Antoine Pichot
Antoine.Pichot@alcatel-lucent.fr
Alcatel-Lucent
Route De Villejust
91620 Nozay, France

Philipp Wieder
ph.wieder@fz-juelich.de
Research Centre Jülich, Central Institute for Applied Mathematics (ZAM)
D-52425 Jülich, Germany

Oliver Wäldrich, Wolfgang Ziegler
{Oliver.Waeldrich, Wolfgang.Ziegler}@scai.fhg.de
Fraunhofer Institute SCAI, Department of Bioinformatics
D-53754 Sankt Augustin

*CoreGRID TR-0082*
June 24, 2007

**Abstract**

A typical task of a grid level scheduling service is the orchestration and coordination of resources in the grid. Especially the co-allocation of resources makes high demands on this service. Co-allocation requires the grid level scheduler to coordinate resource management systems located in different domains. Provided that the site autonomy has to be respected negotiation is the only way to achieve the intended coordination. Today, it is common practice to do this by using web service technologies. Furthermore, service level agreements (SLAs) emerged as a new way to manage usage of resources in the grid and are already adopted by a number of management systems. Therefore, it is natural to look for ways to adopt SLAs for grid level scheduling. In order to do this, efficient and flexible protocols are needed, which support dynamic negotiation and creation of SLAs. In this paper we propose and discuss extensions to the WS-Agreement protocol addressing these issues.

## 1 Introduction

Service Level Agreements (SLAs) are contracts between a service provider and their customers that describe the service, terms, guarantees, responsibilities and level of the service to be provided. They have been widely used by network operators and their customers. They can be used by any service provider for computing, storage, data transport, etc. In this article we will consider SLA dynamically negotiated and created by software programs. A dynamic SLA is an SLA negotiated every time before the service is to be provided. Services to be negotiated are resource provisioning services, they are required to provide higher level services, such as “solve a complex problem”,

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

"run a given application", etc. Those services require the use of various resources like computing nodes, network connections, storage areas or any combination of these.

Resource consumption evolves in time and is sometimes dependent on the successful completion of previous tasks. An orchestrator communicates on behalf of customers (end-users in our use-cases) with several local resource managers to negotiate and create dynamic SLAs. In the rest of the document, for reasons of clarity, we will limit the problem scope to problems where computing and network resources are needed, and to a grid scheduler as orchestrator.

The grid scheduler is the component that has to negotiate, select and schedule resources in order to execute a user's job and fulfil its requirements. As we will see, co-ordinating the access to multiple resources at the same time requires specific protocol features that negotiation and agreement protocols do not necessarily have. The purpose of this introduction chapter is to clarify the functionality of a grid scheduler in our use-cases. Fig. 1 describes several steps performed by a grid scheduler.

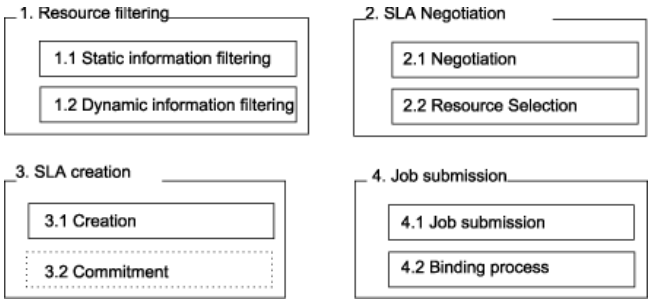


Figure 1: Resource selection & reservation

Section 2 describes SLA negotiation and creation problem, how distributed databases' commit protocol shed light to this problem. Section 3 describes a Meta-Scheduling service used in a real testbed. Section 4 describes how these problems can be solved using Web Service (WS) Agreement protocol being a proposed recommendation of the Open Grid Forum (OGF).

# 2 Negotiation, creation and commit protocols

In this section negotiation is briefly discussed followed by a presentation of commit protocols in distributed databases in Section 2.1 and commit protocols for distributed resource management systems in Section 2.2.

To run a job that requires several resources, like networking and computational resources, managed by different resource management systems (RMS), several steps must be performed by a grid scheduler. UponT receipt of the job request, the scheduler starts the first phase: resource filtering based on static information and dynamic information. Static information does not change over time: number of CPUs, operating system, location, etc. Dynamic information changes over time: availability, load, etc. The second phase is the negotiation process and results in the selection of resources that can satisfy the job request. The third phase is the SLA creation phase concluded by the commitment of all service providers (or local RMS) involved leading to a an reservation of the negotiated resources as described in the SLA. The last phase is the job submission followed by the execution.

Negotiation is a widely studied topic and there are numerous publications addressing different aspects, e.g. [14] is a general purpose negotiation journal, [3] is a survey about negotiation in distributed resource management systems, while [12] and [11] discuss aspects of service negotiation in the Grid. In our context and in the simplest case, a user's job has to be executed and the grid scheduler has to select between different target systems. If all systems are identical and only one parameter influences the selection, i.e. price, this case is similar to a typical business negotiation between one buyer and several sellers. An auctioning mechanism like the ones described in [4] can be used. Of course, we take the point of view of an end user, if we look at things from a resource provider's point of view, we have several jobs that compete for one resource, i.e. several buyers and one seller. If we look at the scheduler's point of view, we have many jobs that compete for several resources, i.e. many buyers and many sellers. Buyya [4] (page 36) also surveyed several distributed resource management systems based on price.

Automatic negotiation of SLAs is a complex and time consuming process [9, 15, 8], when even two users have to find an agreement on multiple criteria. Imagine how difficult the problem becomes when multiple entities have

to reach an agreement [5]. When at least two resources are needed at the same time to run a job, e.g. a network connection and a processing resource, several steps have to be performed before reaching an agreement between the resource providers and the consumer. Green [8] cites mainly two frameworks for automatic negotiation: ontologies and web services. According to him automated negotiation has three main considerations: The negotiation protocol, the negotiation objects and the decision-making models. He considers two options existing in order to achieve this type of negotiation. One option is for the originating agent to negotiate separately with each Autonomous System (AS) along each potential path to ensure that an end-to-end path is available. The dominant choice however, is to use a cascaded approach where each AS is responsible for the entire path downstream of itself. This approach enhances agent autonomy as it is only responsible for its immediate links. The autonomy of the cascaded approach struggles however with the issue of price. In a cascading scenario an intelligent agent would need to know the utility functions of all the downstream domains if the best price combination is to be determined, which is private information. In contrast, in this paper we limited the scope to protocols that permit the negotiation of agreements between two parties based on WS-Agreement [1] rather than tackling the full complexity of automated negotiation. These bi-lateral agreements might then be combined into one single agreement.

## 2.1 Commit protocols for distributed databases

Distributed transactional systems have been widely studied. One of their objectives is to propagate a consistent state across several systems, in a way that at any time all systems can show a consistent state to users. The consistent state or consistent view maintains and propagates between systems a logical coherent state. To provide crash recovery, several operations are logically grouped into transactions. Those transactions permit the change from one consistent view to another. For instance, you do not credit a bank account if you have not debited another bank account. However, these are two independent operations. A bank's distributed database system must group these two operations in one transaction. Thus it permits the change from one consistent state "before the transfer" to another "after the transfer". Database state changes are visible by other users once a transaction is committed to the system. In distributed systems, each transaction can impact several different systems not co-located. Thus distributed database experts have developed commit protocols [2, 10, 13]. As Skeen described in [16], "The processing of a single transaction is viewed as follows. At some time during its execution, a commit point is reached where the site decides to commit or to abort the transaction. A commit is an unconditional guarantee to execute the transaction to completion, even in the event of multiple failures. Similarly, an abort is an unconditional guarantee to "back out" the transaction so that none of its results persist. If a failure occurs before the commit point is reached, then immediately upon recovering the site will abort the transaction. Commit and abort are irreversible."

When a user needs to make a change in a distributed database, a coordinator will propagate this change on all systems. As Skeen explains, upon receipt of a change request the coordinator forwards it to all distributed systems. Upon the change request receipt, all slaves go to the wait state. Then they can decide whether or not to accept this change, and send their response. The coordinator collects all responses to the change request, if one of them is negative, it goes in the abort state and sends an "abort" to all systems, if all responses are positive, then the coordinator goes in the commit state, and sends a "Commit" to all systems. Upon receiving a "Commit" (respectively "Abort") all systems must commit (respectively "Abort") the change request. Fig. 2 (left) represents a slave's two phase commit protocol finite state machine (FSM). This process is the two phase commit process, supported by a two phase commit protocol. The problem of this process is that in case of system failure. It's impossible to know whether the transaction was committed or aborted. The wait state leads to both commit and abort state. For instance, when the coordinator fails after having sent a "commit" to some slaves but not all, the remaining slaves can not know whether the transaction should be aborted or cancelled. The two phase commit protocol is an example of a blocking protocol.

To provide crash recovery, and avoid blocking problems, Skeen introduced a three phase commit protocol. He added an intermediary state before the commitment as shown in Fig. 2 (middle). This state corresponds to a prepare to commit. It's impossible to jump from this state to an abort state. He proved that if a state transition was possible between the prepare and the abort state, the protocol would be blocking. As a consequence, from any state on the slave's finite state machine it is possible to determine whether the transaction should be committed or aborted in case of failure. In case of failure a slave in the "Wait" state must abort, while a slave in the "Prepare" state must commit.

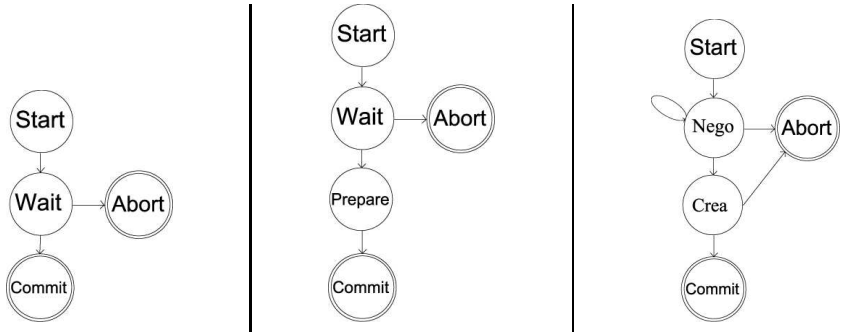


Figure 2: Two phase commit slave's FSM (left), three phase commit slave's FSM (middle), and SLA negotiation and creation resource provider's FSM (right)

## 2.2 Commit protocols for distributed resource management systems

In an environment with distributed RMS providing guarantees on resource usage, a grid scheduler may create SLAs with its users. In a co-allocation use case, this SLA takes into account several resources coming from several resource providers. With each independent resource provider a bilateral SLA has to be negotiated and created. A grid scheduler has to create these bilateral SLAs on behalf of its users. For instance, in VIOLA, users may request network and computational resources with a dedicated QoS. The grid scheduler has to orchestrate the individual reservation of network and computational resources. These two reservations are realised as two bilateral SLAs.

The essence of distributed databases' commit protocol is the transaction: a group of individual operations linked logically. In a distributed resource management system, co-allocation requires multiple bilateral SLAs. For a user or a particular service requiring multiple resources, either all of the individual bilateral SLA must be created, or none. The user SLA creation process is a transaction composed of multiple bilateral SLA creation.

Before reaching an agreement, two steps must be performed: negotiation and creation. The negotiation process can involve all resource providers. Its results are input to a resource provider selection process. When two resources are needed, e.g. network and computing, even if the negotiation involves many compute resource providers, only one computational resource will be selected. For many resources offered, the negotiation process does not lead to an SLA creation process. This is the main reason why negotiation must neither obligate the provider nor the consumer of the SLA. However, the SLA negotiation and creation process should minimize the number of discarded agreement creation requests when it has been previously negotiated. This should occur only when there is a race condition: when two or more users are competing simultaneously for the same resource at the same time. The separation of agreement negotiation and agreement creation process and minimising the number of discarded agreement creations after negotiation are conflicting objectives.

One way to observe atomicity of the SLA creation is to use a transaction and to rely on a two phase commit protocol. Once resources have been negotiated, the orchestrator starts the SLA creation process by sending an SLA creation request to the selected resource providers. Then each resource provider responds to the request with yes or a counter offer. If all providers agree, the orchestrator sends a commit reservation to all systems. Upon receipt of this message, the reservation is committed and the SLA created. Fig. 2 (right) shows this process.

When the resource provider receives an SLA negotiation offer, its state changes from "Start" to "Nego". It then answers the negotiation offer by either accepting it or making a counter offer. In case of a counter offer, it stays in the "Nego" state. It can also abort the negotiation and proceed to the "Abort" state. Once the orchestrator decides to start the SLA creation process, upon receipt of the SLA creation request, the resource provider's state changes to the "Prepare" state. It stays there if it accepts the reservation otherwise it goes to the "Abort" state. The final "Commit" state is reached when it receives a "Commit" message from the orchestrator and that resources are reserved and made unavailable to the rest of the world.

As mentioned above, this simple two phase commit scenario can lead to a race condition during the SLA creation process. While the resource provider is in the "Crea" state, other users see the previous consistent state where resources are still available. To prevent this, the "Crea" state could imply "locking" resources thus providing a pre-reservation for the transaction lifetime. This prevents other users from reserving the same resource at the same time. In case of a lock request, second users' transaction must wait for the lock to be released.

Although the FSMs shown in the middle and on the right-hand side of Fig. 2 look similar, we cannot say that the

SLA negotiation and creation process is a three phase commit. It is a blocking protocol as described by Skeen [16]. And it does not provide any guarantees against crashes. One could still imagine a non blocking SLA creation protocol relying on a three phase commit providing crash recovery. It will not be discussed in this article.

# 3 Resource negotiation with MSS

In the German project VIOLA, Vertically Integrated Optical Network for Large Application, a MetaScheduling Service (MSS) has been developed [17] to co-allocate network resources as well as computing resources.

The MSS is a grid scheduler capable of orchestrating and co-allocating resources. In order to co-allocate resources MSS mediates between the RMS involved. During this process, the MSS creates resource reservations, which are coordinated in time.

The communication between MSS and the involved RMSs can either be done directly or through a standard grid middleware system. At the moment the MSS supports UNICORE [6] as one grid middleware. UNICORE provides seamless and secure access to distributed resources and enables the MSS to access computational resources' management functions in a uniform way, even if the resources are located in different organizational domains. However, since the MSS should be extensible for the future, the integration of the MSS into UNICORE was done using the adapter patterns [7]. This allows the MSS to support different grid middleware systems.

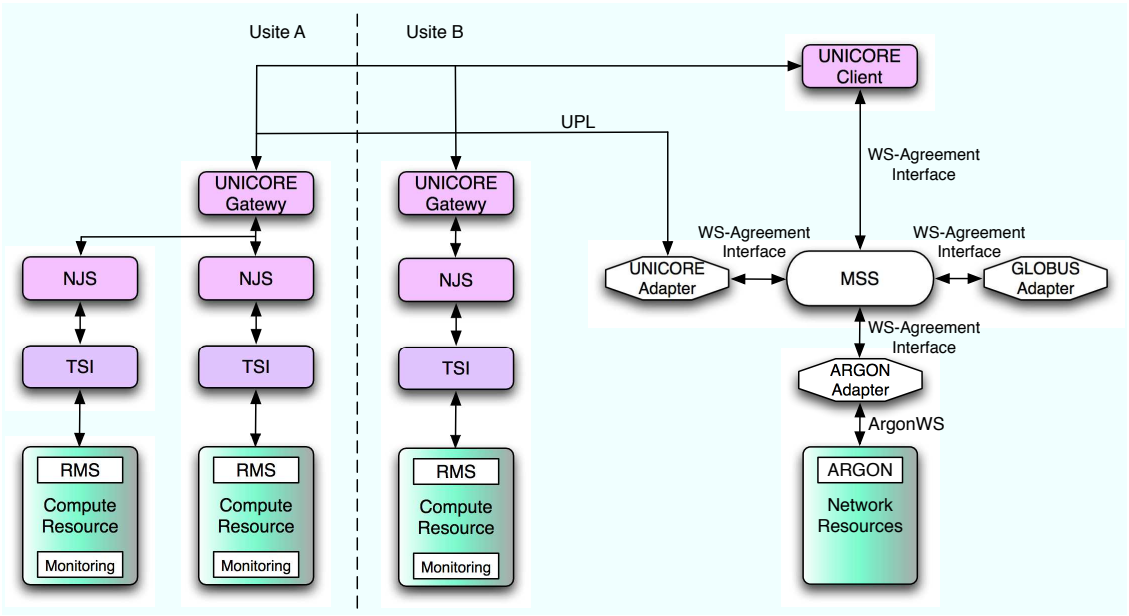


Figure 3: MSS architecture

We use the WS-Agreement protocol for the communication between MSS and the adapters. This is a very natural approach, since the original purpose of MSS was the orchestration and co-allocation of resources using advance reservation. A reservation of resources is an instance of an SLA, where the availability of the required resources at a specific time is guaranteed. These atomic SLAs are orchestrated by MSS in order to create more complex SLAs. Such a complex SLA can for instance contain guarantees that compute resources of different (remote) sites are available at the same time, and a certain level of network quality of service (QoS) is provided between these sites. The WS-Agreement protocol provides a standard way to create and monitor atomic as well as complex SLAs.

The following section describe different ways for negotiating and creating SLAs with WS-Agreement.

# 4 SLA negotiation and creation with WS-Agreement

In order to co-allocate different types of resources and/or resources from different domains, MSS has to negotiate SLAs for the required resources. The easiest way of SLA negotiation is a one step process, where the context, the

subject, and the constraints of the negotiation problem are defined. The WS-Agreement protocol natively supports this kind of negotiation by the GetTemplate method. This method returns a set of agreement templates representing acceptable agreement offers for an agreement provider. These agreement templates only provide hints on agreement offers which might be accepted by an agreement provider. They do not guarantee the agreement will be accepted. An agreement template defines one ore more services that are specified by their *service description terms* (SDT), their *service property terms* (SPT), and their *guarantee terms* (GT). Additionally an agreement provider can constrain the possible values within the SDTs, SPTs, and GTs by defining appropriate creation constraints within the templates.

The creation constraints in agreement template can be static or dynamic. Typical examples of a static creation constraints are the minimum and maximum numbers of CPU, nodes, or memory. As these are properties of computing systems that are not likely to change frequently agreement templates that only contain static information usually are not restricted in their lifetime.

Agreement templates can also contain more dynamic information. Such dynamic information can be used to e.g. restrict the guaranteed execution time of a given service based on the current resource availability. Since the availability of resources is likely to change frequently, templates that contain such dynamic components have a short lifetime. A grid scheduler can use these dynamic templates to efficiently find suitable time slots in order to e.g. co-allocate resources.

However, it is not always desired to expose availability information, or sometimes it is even not possible to do this in a convenient way. A typical example for this is the creation of an SLA in the network domain. Here, it is simply not possible to include the availability information for all possible network paths in a domain within one single SLA template. This would make the templates far to complex and therefore practically unusable. Therefore, the efficient agreement on time constraints in SLAs in only one phase is simply not feasible in this case. More advanced multi-step negotiations are needed to solve this problem.

## 4.1 Negotiation of Agreement Templates

Negotiation requires an iterative process between the parties involved. To rely on WS-Agreement and minimize the extensions to the proposed standard, we suggest not to negotiate SLAs but to negotiate and refine the templates that can be used to create an SLA. Here, our focus is on the bilateral negotiation of agreement templates.

In the following scenario we describe how an agreement initiator (e.g. the grid scheduler) negotiates agreement templates with an agreement provider (e.g. a service provider). We propose a simple offer/counter offer model. In order to use this model in the WS-Agreement protocol, we propose a new function negotiateTemplate. This function takes one template as input (offer), and returns zero or more templates (counter offer). The negotiation itself is an iterative process. In the following scenario we describe a simple negotiation process. During the negotiation process we use the term 'negotiation initiator' for the agreement initiator (e.g. the grid scheduler). Accordingly we refer to the agreement provider (e.g. the resource provider) as 'negotiation responder'.

1. *Initialization of the negotiation process*

   First, the negotiation initiator initialises the process by querying a set of SLA templates from an agreement provider. From this templates, the initiator chooses the most suitable one as a starting point for the negotiation process. This template defines the context of the subsequent iterations. All subsequent offers must refer to this agreement template. This is required in order to enable an agreement provider to validate the creation constraints of the original template during the negotiation process, and therefore the validity of an offer.

2. *Negotiation of the template*

   After the negotiation initiator has chosen an agreement template, it will create a new agreement template based on the chosen one. The new created template must contain a reference to the originating template within its context. Furthermore, the agreement initiator may adjust the content of the new created template, namely the content of the service description terms, the service property terms, and the guarantee terms. These changes must be done according to the creation constraints defined in the original template. Additionally, the negotiation initiator may also include creation constraints within the new created template. These constraints provide hints for the negotiation responder, within which limits the negotiation initiator is willing to create an agreement. After the initiator created the new agreement template according to its requirements, the template is send to the responder via a negotiateTemplate message.

When the responder has received a negotiateTemplate nessage, it must first check the validity of the input document (refined template). This step includes (i) retrieving the original agreement template that was used to create the input document, (ii) validating the structure of the input document with respect to the originating template, and (iii) validating the changes of the content in the input document with respect to the creation constraints defined in the originating template.

Once this is done, the agreement provider now checks whether the service defined in the request could be provided or not. In the first case it just returns the agreement template to the client, indicating that an offer based on that template will potentially be accepted. In the latter case the provider employs some strategy to create reasonable counter offers. During this process the agreement provider should take into account the constraints of the negotiation initiator. Counter offers are basically a set of new agreement templates that base on the template received from the negotiation initiator. The relationship between dynamic created templates and original ones must be reflected by updating the context of the new templates accordingly. After creating the counter offers the provider sends them back to the negotiation initiator (negotiateTemplate response).

3. *Post-processing of the templates*

   After the negotiation initiator received the counter offers from the negotiation responder, it checks whether one or more meets its requirements. If there is no such template, the initiator can either stop the negotiation process, or start again from step 1. If there is an applicable template, the initiator validates whether there is need for an additional negotiation step or not. If yes, the initiator uses the selected template and proceeds with step 2, otherwise the selected template is used to create a new SLA.

## 4.2 SLA creation

After the negotiation of an agreement template acceptable for both parties, the initiator needs to create the agreement. At this point, a problem similar to the transaction problem of distributed database systems arises. The goal of a grid scheduler is to create a set of SLAs with different resource providers in order to provide co-allocation. Therefore, the scheduler first negotiates a set of templates with the providers, which identify the possible provisioning times of the required resources. However, we must not forget that templates only provide hints of what SLAs an agreement provider might accept. There is no guarantee associated with a template. This means that we are in need of a strategy to create all SLAs or none. In principle there are two major strategies to achieve this:

1. *to use transactions to create the SLAs, or*
2. *to create each SLA within one step, applying policies to the SLA.*

The usage of transaction mechanisms to create distributed SLAs, namely the usage of the two-phase-commit protocol, was already discussed in this paper. Since there is no support for two phase commit in WS-Agreement today, we need to extend the proposed standard to address this problem. This process has been started recently in the OGF working group that created WS-Agreement.

The creation of an SLA within only one step is already possible using today's WS-Agreement functionality. However, it is not very obvious how we can solve our co-allocation problem with this approach. In order to achieve this, we have to investigate the content of an SLA. On one hand, an SLA describes the service and its properties. On the other hand, it specifies the guarantees for a specific service. In a co-allocation scenario, where a grid scheduler uses SLAs to coordinate e.g. network and computational resources, it employs execution guarantees in order to assure that the different services are provided at the same time. These guarantees may also include costs that are associated with the service if it is provided successfully, as well as penalties that arise when a guarantee is violated. However, an SLA might be prematurely terminated by the agreement initiator, before the service is actually provided. In fact, this is a cancellation of an SLA. When a service provider guarantees a certain execution time for a service, this normally comprises resource reservations. Therefore, the resource provider wants to prevent the termination of an existing SLA. This can be achieved by including a basic payment within the SLA. The basic payment is potentially a very small amount of money that is even charged if the SLA is terminated by the agreement initiator before the service was actually provided. It is therefore a termination penalty and represents the costs for the overhead produced by the resource reservation. In order to enable the grid scheduler to efficiently negotiate and create SLAs, there could be a certain time period in which the SLA can be terminated without penalty. The duration of this period can dynamically

be specified during the negotiation process. The Agreement provider could use a certain trust index in order to determine the maximum length of this period. For example, such a trust index could be computed by the ratio of successful created agreements and prematurely terminated agreements. This offers a feasible solution for the orchestration of multiple resources using the current one-step SLA creation of WS-Agreement.

# 5 Outlook

In this paper we discussed basic functionality for resource orchestration in grids, namely mechanisms to dynamically negotiate and create service level agreements using WS-Agreement. SLAs are a basic building blocks for grid resource orchestration and distributed resource management. We have shown how a bilateral WS-Agreement based negotiation process is used to dynamically negotiate SLA templates. For this we propose a simple extension of the WS-Agreement protocol in order to support a simple offer/counter-offer model. We did not address extensions to support auctions based negotiation in WS-Agreement so far. This is still future work.

The second relevant part of the resource orchestration process is the creation of distributed SLAs. We have discussed two different strategies to co-allocate SLAs in the grid. In future we need to further investigate the advantages, disadvantages and limitations of one- and two-phase-commit protocols in the distributed resource management domain.

# 6 Acknowledgements

Some of the work reported in this paper is funded by the German Federal Ministry of Education and Research through the VIOLA project under grant #123456. This paper also includes work carried out jointly within the CoreGRID Network of Excellence funded by the European Commission's IST programme under grant #004265.

# References

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). GWD-R (Proposed Recommendation), Open Grid Forum, 2007.

[2] B. Bhargava. *Concurrency and Reliability in Distributed Database Systems*. Van Nostrand Reinhold, 1987.

[3] C. Briquet and P.-A. de Marneffe. Grid resource negotiation: survey with a machine learning perspective. In *PDCN'06: Proceedings of the 24th IASTED international conference on Parallel and distributed computing and networks*, pages 17–22, Anaheim, CA, USA, 2006. ACTA Press.

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

[5] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. Snap : A protocol for negotiation of service level agreements and coordinated resource management in distributed systems. In *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh, Scotland*, July 2002.

[6] D. Erwin. UNICORE Plus final report. Uniform interface to computing resources. Technical report, 2003.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series)*. Addison-Wesley Professional, 1995.

[8] L. Green. Service level negotiation in a heterogeneous telecommunication environment. In *Proceeding International Conference on Computing, Communications and Control Technologies (CCCT04), Austin, TX, USA*, August 2004.

[9] N. Jennings, P. Faratin, A. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *Group Decision and Negotiation*, 10(2), March 2001.

[10] W. Kohler. A survey of techniques for synchronization and recovery in decentralized computer systems. *ACM Computing Surveys*, 13(2), June 1981.

[11] D. Kuo, M. Parkin, and J. Brooke. A framework & negotiation protocol for service contracts. In *Proceedings of the 2006 IEEE International Conference on Services Computing (SCC 2006)*, pages 253–256, 2006.

[12] D. Kuo, M. Parkin, and J. Brooke. Negotiating contracts on the grid. In *Exploiting the Knowledge Economy - Issues, Applications, Case Studies, Volume 3, Proceedings of the eChallenges 2006 (e-2006) Conference*. IOS Press, 2006.

[13] M. Oszu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.

[14] M. Shakun, editor. *Group Decision and Negotiation*. Springer Netherlands, 2002.

[15] W. Shen, H. H. Ghenniwa, and C. Wang. Adaptive negotiation for agent-based grid computing. In *Proceedings of AAMAS2002 workshop on agentcities: Challenges in Open Agent Environments*, pages 32–36, Bologna, Italy, 2002.

[16] D. Skeen. Nonblocking commit protocols. In *Proceedings of ACM SIGMOD Int'l Conf. Management of Data*, June 1981.

[17] O. Wäldrich, P. Wieder, and W. Ziegler. A meta-scheduling service for co-allocating arbitrary types of resources. In R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, editors, *Proceedings of the Second Grid Resource Management Workshop (GRMWS 05) in conjunction with Parallel Processing and Applied Mathematics: 6th International Conference (PPAM 2005)*, volume 3911 of *Lecture Notes in Computer Science*, pages 782–791. Springer, 2005. ISBN: 3-540-34141-2.