SLA Lifecycle Management in Services Grid - Requirements and Current Efforts Analysis[1]

André Ludwig, Bogdan Franczyk

Institut für Wirtschaftsinformatik
Universität Leipzig
Marschnerstr. 31
04109 Leipzig, Germany
ludwig@wifa.uni-leipzig.de
franczyk@wifa.uni-leipzig.de

Abstract: An important aspect of managing service-oriented grid environments is management of agreement relationships between service providers and their customers. The key concept in addressing these agreements is service level agreement. In this paper we motivate the need for SLA lifecycle management in services grid and present results of our analysis on functional and non-functional requirements. Afterwards the state of the art in managing service level agreements is revisited by a comprehensive evaluation of current efforts and their respective strength and weaknesses in service level agreement lifecycle management systems. The results of this evaluation have been applied to the development of the open management platform Adaptive Service Grid which is an EU founded research project.

1 Introduction

Loosely coupled distributed systems in service discovery, composition and execution have emerged as a new paradigm in realizing virtual organizations. Services Grid environments facilitate the usage of services across domain boundaries in a service-oriented fashion and aim at supporting adaptive discovery, composition, negotiation, and enactment of distributed services for scientific and commercial usage areas. With a possibly great number of services fulfilling required tasks, one of the goals of such environments is to provide requesters with services that are delivered according to a requested service level based on a selection of appropriate services.

[1] Acknowledgments: This work has been partly supported by the EU FP6 Integrated Project on Adaptive Services Grid (EU-IST-004617) funded by the Sixth Framework Programme of the European Commission.

In order to make these choices on utilization of services, service quality and according quality of service (QoS) attributes must be defined and taken as reference for service lifecycle management. For this purpose and in an on-demand environment in which services are contracted on a short notice, "best effort" service guarantees are not sufficient. A commercial service requires an agreement that specifies what the user receives from the offered resources and the relevant performance guarantees. The setup of agreements between the parties involved, including the definition of service terms and QoS properties are required. The key concept in addressing agreements between parties in service-oriented environments is service level agreement (SLA). A SLA document specifies what the user receives from the offered resources, its relevant performance guarantees; it captures the mutual responsibilities of the provider of a service and its client with respect to functional and non-functional parameters. For the management of resulting SLA configurations dynamic Service Level Agreement Lifecycle Management (SLA LCM) focusing on characteristics of services grid is necessary. The paper at hand provides a detailed discussion of functional and non-functional requirements on services grid SLA LCM. Furthermore a presentation and evaluation of current efforts in the area of dynamic SLA LCM is given.

The paper is organized as follows: after motivating SLA LCM, chapter 3 presents results of our analysis on functional and non-functional requirements for SLA LCM. Chapter 4 of this paper revisits the state of the art in managing service level agreements. We evaluate current efforts in service level agreement lifecycle management and show strength and weaknesses. The paper closes with a conclusions section.

2 Motivation

Service level agreement lifecycle management addresses the disciplined, proactive methodology and procedures used to ensure that adequate levels of service are delivered to all IT users in accordance with business priorities and at acceptable costs [2]. Moreover, it deals with the management of the relationships between service providers and their clients respectively any third parties and how associated service contracts can be formalized in service level agreements. While the area of service level management (SLM) has experienced a regressive growth over the past few years, the use and management of SLAs have not fully established themselves in a large scale yet. That is although service level agreements are the linchpin of SLM and central for its successful realization. The TeleManagement Forum's SLA Management Handbook defines a SLA as "formal negotiated agreement between two parties, sometimes called a service level guarantee. It is a contract (or part of one) that exists between the service provider and the customer, designed to create a common understanding about services, priorities, responsibilities, etc." [3]. In general there are a number of roles given to SLAs; SLAs define responsibilities, liabilities and accountability of parties that are involved in the service consumption and provision. Thus a SLA serves as an effective method of understanding the expectations of a service consumer, planning and implementing the required service infrastructure and calculating negative consequences such as penalties if the SLA was violated.

The basis for this method and at the same time one of the greatest advantages of SLAs is that expectations and requirements are formalized in terms of time frames, thresholds, volumes, locations, costs and so forth. This provides the opportunity to permanently verify that QoS is compliant to the agreement made among the service parties and it allows a pro-active reporting of the service behaviour. This enables QoS compliance visibility and encourages customer satisfaction and service delivery optimization. Moreover, SLAs play an important role in controlling the implementation and execution of services. The SLA is a reference document for management of the service execution and ensuring timely delivery. After all, SLA supports the communication between the service parties and assists in calculation the return on investment for the service provider and customer. Hence, SLA formalizes legal, operational and internal finance-critical aspects [4]. To be useful in defining roles and accountability and to assert itself as the central controlling instrument for service implementation and execution, a SLA document must express the characteristics of a service in a self-contained and unambiguous way. This needs to be done for every individual service level and every qualitatively specific service usage relationship. Although externally described homogeneously and encapsulated into services one can imagine that characterising varied services such as i.e. hotel booking services, high-bandwidth network access services, computational job processing services, and so forth is very diverse and describing service-specific qualitative properties is a complicated task that leads to versatile outcomes. But not only are the varying contents of SLAs important but also their mapping to non-abstract service performance indicators. Only with this direct mapping it is possible to monitor, control and verify the actual service behaviour against SLA values.

SLAs are considered to be an enabler technology for the commercialization of service-based environments [5]. Hence, it is crucial, that the SLA is defined unambiguously, precisely, and understandable for all parties. As a starting point for meeting this challenge the next section proposes a discussion of functional and non-functional requirements on automated SLA lifecycle management in services grid.

3 Requirements

In [6] we derived general SLA life cycle phases from business requirements. The individual characteristics and functional requirements on each of these phases – negotiation, formation, and fulfilment – were described there. On a high level point of view requirements analysis of SLA LCM systems includes the definition of a SLA language and appropriate SLA templates that utilize a standardized SLA terminology for several business domains. Secondly, for overall lifecycle SLA management mechanisms for creation, modification, operation, and termination must be developed. Thirdly, the development of a SLA repository that stores SLA documents during their entire lifetime and a SLA distribution concept that makes SLA documents available to all involved parties is needed.

3.1 Functional and non-functional Requirements on SLA Language/Templates

One of the most important issues that need to be provided to specify and interpret SLA documents is to formulate a language that defines the syntactic structure and semantics of an agreement document between a service provider and requestor with reference to a provided service. It includes support for parameterization of services, flexibility and unambiguousness for SLA definition, language support for the entire SLA lifecycle, and the assignment of unique SLA identifiers and the SLA lifetime. Furthermore, in order to limit the variety of composed SLAs to a set of well-know and pre-defined documents, SLA templates that build up on this SLA language will be used. This can be understood similar to XML documents referenced to a pre-know XML schema definition. Finally, since services in service grid environments may be composed of atomic services into end-to-end composite services for service requesters, compositionality and expression of relationships between SLAs must be supported.

Parameterization: A SLA document keeps all information for an unambiguous service behavior description at one place; this includes a set of parameters, the values of which quantitatively describe a service. Since services are specific, by means of its tier, actor, and domain, a SLA language must allow for a parameterization of service behavior description and provide a qualitative description of a service. Later on this aspect is referred to as quality of service description.

Flexibility and unambiguousness: It must be possible to formulate any service description term by a SLA language. As services are tier- and domain-specific, a SLA language must allow for a high degree of flexibility but also enable a precise formalization of what a service guarantee means. It must support a common understanding of the semantics of a SLA (what does SLA parameters stand for exactly), domain-specific service description terms such as service application specifications or service data specifications.

Support for entire SLA lifecycle: The SLA language must provide syntactic and semantic support for the entire SLA life-cycle including negotiation, formation, fulfillment, and termination periods. As an example the SLA language must support state definitions during the fulfillment period to express the actual validity of a particular agreement term.

Assigning unique SLA identifiers and SLA lifetime: An agreement must have a unique identifier and may only be accessible by the individual agreement parties. Moreover, it must be possible to produce separate agreement documents for every service containing different values for the articles of an agreement. These characteristics in turn must be enabled by the SLA language. Analogously a SLA language must express the assigned lifetime of a SLA document. Each SLA is associated to a temporal validity of the contract. This temporal validity must be expressible using the lifetime management concepts of the SLA language.

SLA templates and structure pattern: In order to narrow the universe of discourse when defining SLAs a SLA language needs to support the formulation of SLA templates. SLA templates limit flexibility to a small set of variants of the same type of SLA, however, SLA templates lead to a few but well understood terms and limit the possibilities of choosing arbitrary parameters, e.g. for quality of service parameters. Summarized the general parameters of a SLA document a SLA language needs to be capable to express are:

- *Purpose and Scope* – describing the reasons behind the creation of the SLA and the services covered in the agreement
- *Involved parties* – describing the parties involved in the SLA and their respective roles and responsibilities.
- *Validity Period* – defines the period of time that the SLA will cover. This is delimited by start time and end time of the term.
- *Restrictions* – defines the necessary steps to be taken in order for the requested service levels to be provided.
- *SLA parameters* – are the levels of service that both the users and the service providers agree on. They include functional and non-functional service parameters (prices, penalties and so forth), metrics used to compute SLA parameters, parameters for monitoring etc. Each aspect of the service level, such as availability, will have a target level to achieve.
- *Negotiation states* – define whether a SLA parameter was mutually accepted/not accepted or whether it is negotiable/non-negotiable or optional/mandatory.
- *Penalties* – spells out what happens in case the service provider under-performs and is unable to meet the objectives in the SLA.
- *Exclusions* – specifies what is not covered in the SLA.

Compositionality and expression of relationships between SLAs: A service can be the result of cooperation between different atomic services that depend on each other. A SLA language must be capable to enable expression of such compositions and relationships. In addition, due to the ability to express relationships between SLAs, SLA violations and predicting their consequences on other SLAs should be supported.

Besides its functional requirements, described above the SLA language has to fulfill a number of non-functional requirements, which are independence from technical infrastructure, standardization, independence from negotiation model and application of security constructs for SLAs. These aspects are characterized as follows:

Independence from technical infrastructure: The SLA language and its terms must be independent from the provider's or customers' technical infrastructure. That means that the SLA language must be able to be applied and processed on different operational systems and hardware platforms. E.g. it makes no sense if a MS Windows™-based system is unable to access the SLA which is written in a language only used on UNIX™-based systems. For this reason a SLA language should be based on interoperable languages, such as extensible language models XML and XSD.

Standardization: The SLA language must be standardized or at least be based on an excepted standard. Only a standard guarantees that all contracting parties have a common understanding of the meaning of the SLA parameters. Standard and its associated predictability are also needed for a cheap and automated SLA deployment process where no manual interaction is needed. Recently the Global Grid Forum specified the Web Services Agreement Specification (WS-Agreement) [1] that describes an XML language for agreements between service parties. A SLA lifecycle management system should utilize such a standardized SLA language that can easily be processed by existing B2B middleware platforms.

Independence from negotiation model: One of the problems in on-demand service composition, enactment and provision platforms, i.e. based on Grid technologies, is to select services out of a variety of offered services for the same functionality. Automated negotiation as described by [7] is one possibility to deal with such selection problems. During a negotiation so called negotiation protocols define the set of rules that prescribe the circumstances under which the interaction between the parties takes place; the rules of encounter [8]. They cover the permissible types of participants, e.g. the negotiators and any relevant third parties, the negotiation states, e.g. accepting bids, negotiation closed, the events that cause negotiation states to change, e.g. no more bidders, bid accepted, and the valid actions of the participants in particular states, e.g. which messages can be sent by whom, to whom, at what stage. The SLA language must be independent from the negotiation model and support all possible SLA negotiation protocols.

3.2 Functional and non-functional Requirements on SLA Lifecycle Management

This section describes the main functional and non-functional requirements on SLA lifecycle management. SLA lifecycle management comprises the operational part of the SLA lifecycle management, i.e. automated SLA creation, modification, and termination, controlled SLA access, distribution and inspection.

Automated SLA creation and modification: Once negotiation between service providers and customers has finished and service configurations that are acceptable by all involved parties fulfill a given overall quality of service, according SLA documents can be issued. It must be possible to create agreements for predefined services and resources modeling service state. A clever and foresighted contracting is needed; the goal of commercial service providers is to maximize profit which is reflected in how SLAs will behave. SLA LCM systems that create SLA documents for composite services have to recognize interdependencies between involved atomic services and derive QoS aggregation and penalty formulas. All these considerations must be made before anything happens at service execution runtime and rules must be set up that cover consequences on any possible fault scenario. Hence, in order to reflect this, relationships among services, e.g. due to service workflows, must be expressible by SLAs.

Automated SLA violation exception handling: In case monitoring activities observe SLA violation, it may be necessary to terminate a SLA and trigger re-planning. For such cases SLA LCM systems must provide an interface to terminate a SLA.

Similarly it is possible to modify SLA values such as extending the SLA lifetime or other modification. In each case dependencies and consequences to other related SLAs will be examined.

Controlled SLA access and inspection: During service provision and usage several components need to request the values of the SLA document, e.g. for monitoring, re-negotiation, re-planning, etc. SLA lifecycle management must deliver SLA values in a public interface. However, each component may only receive the part of the contract it needs to know to carry out its task. Following this "Who needs to know what?" principle allows for privacy and hides issues such as service sub-contracting etc. Beside SLA data access, a SLA lifecycle management component must facilitate SLA inspection. Inspection describes tasks like: provide all related SLAs of the composite service workflow. This information is essential for re-planning or re-negotiation activities.

Formal representation of all SLA aspects at one place: SLA lifecycle management must ensure that all relevant SLA parameters are available at one place. SLA aspects will be kept in documents that store data for automatic service provisioning and monitoring and information that enable an unambiguous service behaviour description.

4 Current Efforts in Services Grid SLA Management

This section examines and evaluates current efforts in SLA life-cycle management. In the area of SLA-based contracting and monitoring, there are several advanced approaches and frameworks such as those presented in [9] and [10]. Among the most promising approaches of SLA management in service-oriented environments are the WS-Agreement specification [1] developed in the Global Grid Forum's GRAAP working group, IBM's WSLA framework [16], and the SLAng language approach [19]. Each of these approaches is discussed below. Considering the generic requirements presented in chapter 3 we show strengths and weaknesses and identify space for improvement.

4.1 WS-Agreement

The WS-Agreement specification, which is being developed in the Global Grid Forum's GRAAP working group, defines a standard way of establishing SLAs between a service provider and a service customer. Following the notion of service orientation and virtualization of resources, the fundamental idea of WS-Agreement is the representation of a SLA as WS-Resource [15] in an agreement service. The objective of the WS-Agreement specification is to provide standard means to establish and monitor agreements on services independent of a particular application domain. The specification draft comprises three major elements: a description format for agreement templates and agreements, a basic protocol for establishing agreements, i.e. by negotiation, and an interface specification to monitor agreements at runtime. Using an XML-based language, agreements and agreement templates are structured into distinct parts. They consist of a context section, the agreement terms and the constraints section.

The context part contains the meta-data for the entire agreement, including the definition of the participants in the agreement, the end-point references of their planned agreement instances and their roles and related prior agreements. The agreement terms represent contractual obligations and include a description of the service as well as the specific guarantees given. It comprises service definition terms and guarantee terms. Service description terms provide information needed to identify a service to which this agreement pertains while guarantee terms specify the service levels that the parties are agreeing to. The guarantee terms are used to monitor the service and enforce the agreement. All terms can be composed using the compositors of the WS-Policy specification [14]. The WS-Agreement specification proposes two basic interface layers for interaction: the agreement layer: which provides a Web service-based interface that represents SLAs; the service layer, which represents the application-specific layer of the provided business service.

Strengths and Advantages: WS-Agreement provides a simple protocol to establish SLAs. The strength of WS-Agreement lies in a well-defined template for specifying agreements domain-independently and for all kinds of resources represented as services in SOA computing environments. WS-Agreement does not cover advertising a service nor does it comprise more advanced means of negotiation. This is evaluated clearly positive as it allows for separation of concerns and application of negotiation mechanisms outside the scope of WS-Agreement. The WS-Agreement content definition provides a general structure for SLAs. Other languages, such as domain specific languages, must be used to describe a service or to define guarantees. On the one hand, this is convenient since it allows parties to use existing languages such as WSDL to describe particular aspects of a service in an arbitrary, self-defined domain-specific language. On the other hand, parties must be able to deal with a variety of specifications whereas mostly no common ontology between SLA parties exists. Hence, agreement initiators will rely on agreement templates published by agreement providers to create agreements that will be understood by providers.

Weaknesses and open issues: The WS-Agreement specification defines only the top-level structure of agreements and agreement templates. This outer structure must be complemented by means of expression suitable for a particular domain. There are no language elements defined to specify the SLA's guarantee parts. Although this is positive for more flexibility, agreement parties have to choose a suitable condition language to express the logic expression defining this information, e.g. OMG's Object Constraint Language (OCL) [17]. However, choosing and agreeing on common SLA semantics and domain-specific languages is often just the main problem. WS-Agreement does not cover how to access a service according to an agreement. Transactional services entail a client application to send SOAP messages to URLs and it may be necessary to label messages as being subject to an agreement, for example by adding a contract ID to SOAP headers. WS-Agreement has significant weaknesses in interplaying with agent-based agreement negotiations. The main problem lies in its limitation of the number of messages (only offer and agree messages foreseen) and its lack of an interaction protocol between parties. Although it is seen positively that the WS-Agreement specification remains independent from particular negotiation protocols and mechanisms, it should at the same time be flexible enough to be applied to any kind of negotiation.

4.2 Cremona

WS-Agreement standardizes the interaction between the organizational domains and provides a standard means of agreements and agreement templates. In addition, providers require an infrastructure to manage agreement templates, implement the interfaces, check availability of service capacity and expose agreement states at runtime. At the same time, also agreement initiators need an infrastructure to read and fill in templates, create agreements, and monitor agreement state at runtime. For this purpose IBM proposes the Cremona (Creation and Monitoring of Agreements) [11], architecture, a middleware for implementing and processing WS-Agreement. The Cremona Java Library implements the WS-Agreement interfaces, provides management functionality of agreement templates and instances, and defines abstractions of service-providing systems.

Strengths and Advantages: Cremona provides a layered agreement management architecture and a library for integrating system or domain-specific functions. To the author's best knowledge, this is the first attempt to provide a processing middleware for the WS-Agreement specification draft and can be seen as an enabler for the specification in general. The Cremona library provides implementations of the domain-independent components in Java and defines interfaces that can be implemented in a domain-specific manner. Hence, Cremona remains platform-independent and can be used in multiple environments incorporating all kinds of resource services, i.e. Web and Grid services. The development of the Cremona Java libraries is work in progress and a prototypical sample implementation is freely available online [12].

Weaknesses and open issues: WS-Agreement is suitable for a wide range of services and different aspects of a service such as QoS, interfaces, etc. However, its set of obligation types is not very rich and the set of obligations is static, no new obligations can be added by, e.g., exercising an option. This is likely to change in the future development of WS-Agreement and will have to be reflected in Cremona. The effort needed on implementing domain- and service-specific SLA solutions remains high in Cremona. Additional support for a dynamic and automated creation of SLA documents having the same unambiguous expressiveness is required.

4.3 Web Service Level Agreement

The Web Service Level Agreement (WSLA) [16] language is a XML based Web services SLA language developed by IBM for the purpose of service level management in Web-service-based environments. It is part of the IBM Emerging Technologies Toolkit [13] and aims at unambiguous and clear specification of SLAs that can be monitored by service providers, ease of SLA creation via template based authoring (i.e. XML schema to represent WSLAs), and a distributed monitoring framework that translates SLAs into configuration information to perform measurement and supervision activities. WSLA provides both an XML schema definition for describing SLAs and a runtime environment for the actual SLA management. The structural elements of the WSLA schema definition include a parties section, service description section and an obligation section and are explained in detail in [20].

The WSLA runtime environment includes a deployment service (for installing and configuring the technical environment), a measurement service (for measuring the QoS), and a condition evaluation service (for identifying contract infringements).

The WSLA approach uses a three-tier architecture including service customer using a service, a service provider offering the service and an independent third-party service hub supplying/brokering the service. The service customer and provider do not have a direct connection, as the service hub is responsible for processing the service business transaction. It is based upon XML and aims to be applicable to any domain. One of the main ways it achieves this is by allowing domain-specific language extensions. The mechanism for doing this is type derivation by XML schema. A set of standard extensions also exists to cover common usage situations.

The first section of an agreement in WSLA specifies the parties involved. As well as signatory parties (the service provider and the client), WSLA allows supporting parties to be specified. One particularly useful result of this is that a trusted third party for measurement and monitoring may be agreed. The service definition section follows. For each QoS parameter, a metric is defined and for each metric, a Measurement Directive is specified. This directive contains the information needed to retrieve the metric. Composite metrics (e.g. average, minimum, maximum, etc.) may be specified as a function in the SLA or exposed by the service provider as a well-defined interface for further processing. SLA Parameters map to a metric and may be associated with a permitted range for a specific customer. The final section of the SLA is the parties' obligations. This may contain two types of obligation: the Service Level Objective and the Action Guarantee. The first type guarantees a particular state of SLA parameter in a given time period. The second guarantees that a specified action will be performed in given circumstances. The details of negotiation are not specified in WSLA, but ebCPP [21] is hinted at as a possible means of doing so.

Strengths and Advantages: The most important benefit of WSLA is that it is thoroughly documented, a full language specification is publicly available, and it exists as integrated part of a wider framework. WSLA is extensible, XML-based, allows logical expressions such as trade-offs between parameters to be expressed and allows failure behaviour to be specified. WSLA documents include both, parameters describing assurances on the QoS, and business terms and conditions including pricing and penalties. Therefore, WSLA is structured in such a way that monitoring clauses can be separated from contractual terms for distribution to a third party. This principle of syntactic separation is clearly useful. WSLA provides templates associated with an offering that define the QoS properties of the service. Templates contain constraint-described fields to be filled in during the SLA creation process. The WSLA runtime environment defines management actions, including notifications in the event of SLA violations.

Weaknesses and Open Issues: WSLA is a specification that uses the concept of formalized SLA contracts in the context of electronic services. However, this approach is specific to a particular aspect of a service, namely the QoS agreements of Web services. It has been used to drive a system provisioning [22] but is restricted to Web services and does not propose solutions for service consumers. WSLA allows the definition of management information, including financial terms associated with SLAs. Although needed, these are not presented with a defined semantic, but are clearly a desirable feature of SLA languages. WSLA assumes that all QoS measurements are provided by a web service encapsulating monitor. No constraints are placed on the implementation of such monitors, so a common understanding of their role remains external to the definition of the language. WSLA provides the ability to create new metrics defined as functions over existing metrics. This is useful to formalise requirements expressed in terms of multiple QoS characteristics, without impacting on notions of compatibility of SLAs. The semantic for expressions over metrics is not formally defined yet.

4.4 SLAng

SLAng [19] is an extensible SLA language approach that uses XML for defining QoS attributes and specifies the semantics of these attributes using UML models. SLAng distinguishes itself from other approaches in the following respects: Firstly, in contrast with other languages that focus on web services exclusively, SLAng defines SLA vocabulary for a spectrum of IT services, including Application Service Provision, Internet Service Provision, Storage Service Provision and component hosting, motivated by the observation that federated distributed systems must manage the quality of all aspects of their deployment. Secondly, the meaning of SLAng is formally defined in terms of the behaviour of the services and clients involved in service usage. Benefits of the formal semantics include the reduction of ambiguity in the meaning of the language and the means to check the semantics to ensure the absence of inconsistencies and loopholes. The style of semantic definition used aims to be user-friendly, allowing it to serve as a reference for human negotiators. It also provides a formal basis for comparisons between SLAs, and an abstract reference model of systems employing SLAs that can guide implementation and analysis efforts. These latter facilities address two types of compositionality for SLAs: inter-service composition in which required QoS levels are compared to offered QoS levels, and intra-service composition in which the QoS levels offered by a service are related to the levels provided by its components.

The structure of an SLA in SLAng consists at the top level of an indication of whether it is horizontal or vertical. Below this comes the SLA type, which is one of seven set types including, for instance, application (a vertical SLA type) and networking (horizontal). The SLA then contains responsibilities in three sections: client, server and mutual. In each of these sections, parameters specific to the SLA type (and therefore limited to those defined in the language itself) are defined.

SLAng uses the Unified Modelling Language (UML) [18] to model the language and producing an abstract syntax. The UML language model is related to an abstract model of services, service clients and their behaviour. As one of the means of SLAs is constraining the behaviour of associated services and service clients, SLAng uses the Object Constraint Language (OCL) [17] with accompanying natural language descriptions to define these constraints formally and the semantics of the language.

Strengths and Advantages: The advantages of the SLAng approach are to reduce the ambiguity of a SLA language by determining a correspondence between elements in an abstract service model and services and events in the real world. It documents the semantics of the language using the OCL standard. As the aim of SLAng is to provide the basis for a common understanding of SLA terms between parties to an SLA, it enables the comparison between SLAs, supports SLA composition, and provides a reference for negotiation and arbitration of agreements. The mechanism that underlies this common semantic is a QoS catalogue, which is modelled as a UML extension.

Weaknesses and Open Issues: SLAng is work in progress, an approach and evolving language that is neither standardized nor largely adopted by industries and academia. Although realizable by extending SLAng, it lacks mechanisms to relate service violations to payment of penalties, features enabling the reuse of agreements, SLA templates, and the specification of management actions, performed in response to QoS state changes or SLA violations. However, the most important unsolved problem is the absence of the ability to define new QoS parameter types, i.e. in terms of existing types. Dynamic creation of agreement documents specific to the versatility of the underlying service, i.e. all kind of grid services, is therefore also limited.

4.5 Overall Evaluation of Current SLA Lifecycle Management Contributions

In the context of service-oriented environments, a number of approaches targeting at service level agreements and management arose within the last years. Efforts like IBM's Web Service Level Agreement (WSLA) language, GGF's WS-Agreement specification together with the Cremona middleware architecture, as well as SLAng have laid a recognizable foundation for describing and managing SLAs by XML-based representations in service-oriented environments. However, the evaluation above has also shown that a number of issues remain open while they are crucial for a successful commercialization of service-oriented platforms like ASG [23]. One of the most important aspects that is often not considered, for example by WSLA, is the extension of the service term definition onto all kinds of electronic services that can be considered in an adaptive services grid environment. This includes not only Web services but also resources such as computational power, network access etc. which can be represented homogeneously by WSRF services. Consequentially distinguishing characteristics of these resources (like different meanings of QoS) have to be analyzed, structured and finally integrated into SLA documents.

SLAng provides a first approach to model these differences in UML diagrams. Nevertheless, no mechanisms to define new QoS parameters, i.e. by composing existing operators or methods to create SLA instances from these models are proposed yet. One of the advantages of the WS-Agreement specification draft is the flexibility to describe different characteristics of services syntactically. Controlling the implementation and execution of services and serving as a reference document for configuring services and ensuring timely delivery is one of the key roles that are given to SLAs. For instance, WSLA and Cremona consider these possibilities and are either embedded in a wider framework or model a surrounding framework for service execution management. This aspect should be considered for other SLA lifecycle management systems as well.

A major problem in SLA management that is still unsolved yet is the lack of a common ontology between the service customer and service provider that can be reflected in the SLA. While for instance, WSLA allows the definition of management information including financial terms associated with SLAs, no defined semantic is presented. In addition, the semantic for expressions over metrics is not standardized and remains a problem of all presented approaches. Hence, one of the main targets of future platforms should be the development or application of an ontology language. This enables the formal semantic description of services and the interaction with service customers and providers. Once these semantics are defined and formalized they will be included in the SLA document as well, which thereby becomes commonly understood by all parties and processable by management applications.

Rudimentary covered by WS-Agreement but not supported by frameworks like Cremona or WSLA is the representation of relationships between agreements. Services are often to be included in composite services defined using, for example Business Process Execution Language for Web Services (BPEL4WS) [24]. In this case of a process composed of services, it is necessary to understand how the individual QoS properties of one element of a composite service contribute to the overall QoS of the process. The underlying information source for these QoS decisions is provided by SLA documents.

Although most of the work discussed above recognises SLA negotiation as a key aspect of SLA management, they usually provide little guidance of how negotiation (especially automated negotiation) can be realised. In a more general context, automated negotiation has been an important part of agent research. The objective of SLA lifecycle management systems should be the development of an interaction pattern between the Negotiation Manager and the SLA Life-cycle Manager to incorporate these closely connected research areas.

While the formalization of contracts by extensible languages is broadly applied in all approaches, the definition of semantics and a common ontology for service and service level agreements respectively remains a challenge. However, the basis for creating service semantics may be to structure and model services by modelling languages such as UML. The service term definition, comprising all kinds of electronic services, must be the basis for abstraction, finding common characteristics and emphasizing differences. The SLAng approach commences this idea. Once modelling of service resources is supported successfully, mechanisms to create SLA documents dynamically on demand and in an adaptive fashion are needed.

5 Conclusions

Dynamic management of service level agreements is an area of research in services grid that has recently been attracting more and more attention. SLA lifecycle management is difficult to automate as it needs a precise and unambiguous definition of the SLA as well as a customizable engine that understands the specification, customizes instrumentation, collects the necessary data, models it in a logical manner and evaluates the SLA at the required times. In this paper we identified functional and non-functional requirements on dynamic SLA LCM and provided a comparison of existing SLA languages. As a result we showed that existing SLA languages do not provide sufficient support for these requirements and call for further improvement. The most important questions stimulating research in this area include proper description of services and service qualities and embedding them into interoperable platform-independent specification languages such as XML. Further management platforms for entire SLA lifecycle management are needed for creation, fulfilment and evaluation of SLA documents.

References

[1] Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Pruyne, J. Rofrano, J., Tuecke, S., Xu, M.: Web Service Agreement Specification (WS-Agreement) 1.1 (2004)

[2] Sturm, R., Morris, W., Jander, M.: Foundations of Service Level Management, Sams, 2000.

[3] Telemanagement Forum: SLA Management Handbook, Concepts and Principles (2005), available at http://www.tmforum.org

[4] Lee, J., Ben-Natan, R.: Integrating Service Level Agreements: Optimizing Your OSS for SLA Delivery (2002), Wiley

[5] Gartner Symposium – ITxpo 2002, Documentation CDROM, Cannes/France, November 2002

[6] Ludwig, A., Braun, P., Kowalczyk, R., Franczyk, B.: A Framework for Automated Negotiation of Service Level Agreements in Service Grids. Third International Conference on Business Process Management (BPM 2005), Nancy (France), September 2005. Springer-Verlag, 2005.

[7] Jennings, N. R., Parsons, S., Sierra, C. and Faratin, P.: Automated Negotiation. Proc. 5th International Conference on Practical Application of Intelligent Agents and Multi-Agent Systems (PAAM-2000), Manchester, UK (2000) pp. 23-30

[8] Lomuscio, A. R., Wooldridge, M. and Jennings, N. R.: A classification scheme for negotiation in electronic commerce (eds. F. Dignum and C. Sierra). In: Lecture Notes in Computer Science. 1991 (2001), pp. 19-33.
[9] Salle, A., Bartolini, C.: Management by Contract, HPL-2003-186, HP labs, 2004.
[10] Boström, G., Giambiagi, P., Olsson, T.: Quality of Service Evaluation in Virtual Organizations Using SLAs. submitted to 1st Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems, 2006.
[11] Ludwig, H., Dan, A., Kearney, R.: Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements In: IBM Research Report, ICSOC, N 15-19, New York, 2004., available at: http://www.research.ibm.com/people/h/hludwig/publications/rc23265.pdf
[12] Ludwig, H., Dan, A., Kearney, R.: Cremona: Cremona: Creation, Monitoring and Mangeement of WS-Agreement; available at: http://awwebx04.alphaworks.ibm.com/ettk/demos/wstkdoc/cremona/README.htm, 2005
[13] IBM: Emerging Technologies Toolkit (ETTK), v1.0, available at: http://www.alphaworks.ibm.com/tech/ettk
[14] Box, D., Curbera, F., Hondo, M., Kaler, C., Langworthy, D., Nadalin, A., Nagaratnam, N., Nottingham, M., von Riegen, C., Shewchuk, J.: Web Services Policy Framework (WS-Policy).May 28th, 2003.
[15] Czajkowski, K., Ferguson, D.F., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S.,Vambenepe, W: The WS-Resource Framework (2004). http://www-106.ibm.com/developerworks/library/wsresource/ws-wsrf.pdf
[16] Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kübler, D., Ludwig, H., Polan, M., Spreitzer, M., Youssef, A.: Web services on demand: WSLA-driven automated management. IBM Systems Journal, Vol. 43 (1), 2004.
[17] The Object Management Group (OMG). Object Constraint Language (OCL), v2.0., ptc/03-10-14 edition, October 2003
[18] The Object Management Group (OMG). The Unified Modeling Language v.2.0, ptc/04-10-02 edition, October 2004
[19] Lamanna, D., Skene, J., Emmerich, W.: SLAng: A Language for Defining Service Level Agreements In Proc. of The International Workshop on Future Trends of Distributed Computing Systems (FTDCS'2003), San Juan, IEEE Computer Society Press. May 2003.
[20] Ludwig, H., Keller, A., Dan, A., and King, R.: A Service Level Agreement Language for Dynamic Electronic Services. Proceedings of the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002), 2002, pp. 25–32.
[21] OASIS ebXML CPP/A Technical Committee: Collaboration-Protocol Profile and Agreement Specification (2002), available at http://www.oasis-open.org/committees/ebxml-cppa/documents/ebcpp-2.0.pdf
[22] Ludwig, H.: A Conceptual Framework for Building EContracting Infrastructure. In R. Corchuelo, R. Wrembel, A. Ruiz-Cortez (eds.): Technologies Supporting Business Solutions. Nova Publishing, New York, 2003.
[23] Integrated Project “Adaptive Services Grid”, http://asg-platform.org
[24] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1, http://www-106.ibm.com/developerworks/webservices/library/wsbpel/, 05 May 2003.