Comuzzi, M., Kotsokalis, C., Spanoudakis, G. & Yahyapour, R. (2009). Establishing and Monitoring SLAs in complex Service Based Systems. In: E. Damiani, J. Zhang & R. Chang (Eds.), 2009 IEEE International Conference on Web Services. Los Alamitos, California, I & II. (pp. 783-790). IEEE. ISBN 9780769537092

Establishing and Monitoring SLAs in complex Service Based Systems

Marco Comuzzi[a], Constantinos Kotsokalis[b], George Spanoudakis[a], and Ramin Yahyapour[b]
[a]*City University London,* [b]*Dortmund University of Technology*
{sbbd286, G.Spanoudakis}@soi.city.ac.uk
{constantinos.kotsokalis, ramin.yahyapour}@udo.edu

Abstract

In modern service economies, service provisioning needs to be regulated by complex SLA hierarchies among providers of heterogeneous services, defined at the business, software, and infrastructure layers. Starting from the SLA Management framework defined in the SLA@SOI EU FP7 Integrated Project, we focus on the relationship between establishment and monitoring of such SLAs, showing how the two processes become tightly interleaved in order to provide meaningful mechanisms for SLA management. We first describe the process for SLA establishment adopted within the framework; then, we propose an architecture for monitoring established SLAs, which satisfies the two main requirements introduced by SLA establishment: the availability of historical data for evaluating SLA offers and the assessment of the capability to monitor the terms in a SLA offer.

1. Introduction

IT–supported service provisioning has become relevant in most industries and domains, including, for instance B2B and B2C commerce, banking, telecommunications. Organizations often package their offers as consumable services encapsulating discrete functionalities along the whole typical business/IT service stack [1], what has been named *Software as a Service*. In recent years, virtualization and autonomic computing have also allowed the provisioning of infrastructure resources as well-defined discrete services. Virtualization, in particular, allows an infrastructure provider to package a set of resources, e.g. computing, memory, storage, in an isolated virtual machine, which can be allocated for the execution of higher-level services to accommodate business customers' requirements. Such offerings are referred to as *Infrastructure as a Service* or, more typically, *Cloud Computing* [2].

Therefore, we see the emergence of a vivid service economy, where business customers can purchase high-level business service bundles, relying on software services and on virtual infrastructure services. The establishment of the business relationships and the business/software/infrastructure service chains required to support the expanding service-based economy, however, makes it necessary to provide service consumers of all layers with certainty regarding the quality offered by each service, be it business, software or infrastructure. Such certainty traditionally comes in the form of *contracts*, and *Service Level Agreements* (SLAs) are the instruments to model such contracts in the digital world. SLAs specify the conditions under which a certain service is provided by a provider to a customer. Provisioning of service hierarchies therefore implies similar dynamic and complex SLA hierarchies, established within and across the boundaries of organizations.

A service provisioning infrastructure should allow the establishment of SLA hierarchies through coordinated negotiations among the potential stakeholders. However, SLA establishment can only partially serve the needs of SLA management if not linked to SLA monitoring. This paper explicates the link between SLA negotiation and SBS monitoring in the context of the SLA Management framework developed by the SLA@SOI Project. SLA negotiation and monitoring involve both the service consumer and providers, the latter of which develop models for crafting and evaluating SLA offers and produce monitoring data during the provisioning of SLAs. In this paper, we focus specifically on the service provider side, while the perspective of the service consumer is part of our future work. In particular, we show how, during negotiation, service providers require historical data from monitoring to evaluate SLA offers made by service customers. We also argue that before an SLA is established, the capability to monitor terms at runtime must be confirmed. We then introduce a monitoring framework which satisfies these requirements.

The rest of the paper is organized as follows. Section 2 introduces the SLA@SOI management framework and a motivating example for the work presented in the paper. Section 3 and Section 4 discuss the architecture developed for SLA Establishment and Monitoring. In Section 5 we discuss evaluation issues, whereas related work is revised in Section 6. Finally, conclusions are drawn in Section 7 along with an outlook on future work.

2. Background

We are researching the issues discussed in this paper as part of the EU FP7 Integrated Project (IP) SLA@SOI (http://sla-at-soi.eu/), one of the 6 strategic projects of the Networked European Software and Services Initiative (NESSI, http://www.nessi-europe.com/). NESSI is the cornerstone effort of the European Union to design and implement a coherent and consistent open service

framework, leveraging research in the area of service-based systems to consolidate and trigger innovation in service-oriented economies.

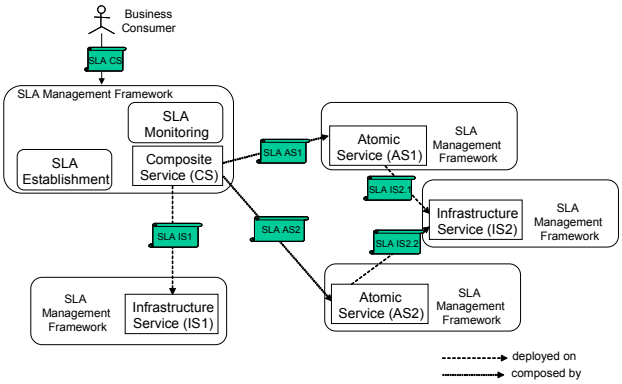

Figure 1: The SLA@SOI SLA management scenario

A general scenario for the SLA management framework is shown in Figure 1. As shown in the figure, a generic Composite Service (CS) is provided to one or more Business Customers. CS is implemented as a *composition* of several atomic Services (AS), namely AS1 and AS2. Both CS and ASs are *deployed* on Infrastructure Services (ISs), provided using virtualization techniques. The provisioning of CS to a customer is regulated by an SLA. From a CS provider perspective, the provisioning of this SLA is based on a complex hierarchy of SLAs, established with atomic and infrastructure services. Thus, the service hierarchy established to implement the composite service, is reflected on an equally complex SLA hierarchy, which governs top-level service consumption and propagates down to the fabric. The proposed framework is generic in order to accommodate different real-world scenarios, including both intra- and inter-domain SLAs.

Independent of the exact use case, the entire set of SLAs that needs to be enforced guarantees the quality of the top-level customer experience, just like service composition enables offering the service to this user in the first place.

3. Dynamic SLAs

When referring to dynamic SLAs, we stress the fact that these are not static, predefined contracts. Instead, they can be a) customized before signing, b) negotiated on their content, and c) renegotiated if the customer and the provider wish to do so. Customization of a SLA refers to the modification of the *SLA template* which is defined and offered by the service provider, as an indication of the acceptable guarantees that may be included in the contract content. We refer to these guarantees as *agreement terms*, adopting the terminology of the Open Grid Forum's Web Services Agreement (WS-Agreement) specification [3]. Negotiation and renegotiation is the phase when the consumer and the provider try to actually reach an agreement on the values for these guarantees and the SLA as a whole, through structured message exchange. During these phases, the two parties are applying their knowledge, assumptions and business axioms, with the purpose of optimizing some utility function that quantifies the value of the contract for them.

3.1. Agreement Terms

As an instrument for showing the explicit relationship between *negotiating* and *monitoring* service guarantees, below we outline some formal definitions of Quality of Service (QoS) properties that are commonly adopted in literature for software services, e.g. [4, 5, 6].

Availability: Assuming service S; time T_1 as the beginning of monitoring time; time T_2 as the time of evaluating availability; monitoring duration $T = T_2-T_1$; b_i as a time when S could not be invoked any more, by all of its (established or potential) customers, due to reasons other than network connectivity, where $T_1 \leq b_i \leq T_2$; e_i as the moment when S became usable again following b_i, where $T_1 \leq e_i \leq T_2$; $d_i = e_i-b_i$; $d = \sum d_i$; we then define availability for service S as $A=(T-d)/T$.

Accessibility: Assuming operation O of service S; time T_1 as the beginning of monitoring time; time T_2 as the time of evaluating accessibility; monitoring duration $T = T_2-T_1$; R_a as the number of all invocations to O during time T; R_d as the number of invocations that were not served (i.e. were dropped) during time T; we then define accessibility for operation O as $C_O=(R_a-R_d)/R_a$.

Throughput: Assuming operation O of service S; time unit t; request arrival rate R = N/t, N=number of requests per time unit t, $N \in$ □; accessibility C=1 for $R = R_1$; accessibility C<1 for $R = R_2$, $R_2 > R_1$; we then define throughput for operation O as $H_O = R_1/t$.

Completion Time and Average Completion Time: Let us assume operation O of service S; request message M_Q of a client to the service S for the invocation of operation O; response message M_R; M_Q received in full on the service end at time t_I; M_R put on the wire in full at time t_O; we then define Completion Time of operation O as $T_{CO} = t_O-t_I$. Assuming a series of Completion Time measurements by the monitoring infrastructure, $T_{CO1}, \ldots, T_{COn}$, we define Average Completion Time as $T_{AO}= (\sum T_{COi})/n$.

Mean Time To Repair: Assuming service S; a moment in time, t_b, that the service becomes unavailable; the respective moment in time, t_e, that it becomes available again; the period (duration) of unavailability, $t = t_e-t_b$; a series of such periods, $T = (t_1, t_2, \ldots, t_n)$ as captured by monitoring infrastructure; the total unavailability time $u = \sum t_i$; we then define MTTR=u/n.

Mean Time To Failure: Assuming service S; a restoration after failure for this service, taking place at time t_b; the consecutive failure of the service, starting at time t_e; the respective period of availability $t = t_e-t_b$; a

series of such periods, $T = (t_1, t_2, \ldots, t_n)$ as captured by monitoring infrastructure; the total duration of service availability, $u = \sum t_i$; we then define MTTF=u/n.

The above list is not meant to be exhaustive, but serves as a proof of the strong link between the terms under negotiation and the monitoring infrastructure, using QoS terms common in scientific and technical literature. In fact, it shows that it is not possible to define the terms at all, without using monitoring artifacts, such as the time at which monitoring starts, or events captured during service provisioning including, for instance, Web service invocations and responses. Therefore, it is not reasonable to assume that we can calculate negotiable values for these terms without having historical monitoring data to rely on, or otherwise, some software design which defines deterministically the performance of a service for every possible input. As will be discussed later in this paper, this argument is further extended in our monitoring framework: *It is not reasonable to negotiate on a term at all, without confirming with the monitoring subsystem that the term can be monitored*.

As an example for the SLA hierarchy, let us use the scenario of Figure 1 and assume that AS2 follows the execution of AS1, as a sequential workflow, implementing CS1. The latter represents a business process that produces revenue of *M* financial units for the customer, every time it is executed. Suppose that the guarantee offered by the provider CS to the end-customer is that there will not be a revenue loss of more than *N* financial units, due to CS malfunction. Such malfunction may be interpreted as reduced service availability / accessibility, or increased completion time (which results in long queues and departing customers). The SLAs between the service provider of CS and those of AS1 and AS2 will then use these software terms, appropriately calculated and negotiated, to ensure proper execution of CS according to the top-level SLA. Additionally, the SLAs between the service providers of CS, AS1 and AS2 with their corresponding infrastructure providers (i.e., those of IS1 and IS2) will typically include guarantees on the number of virtual machines allocated to these services, the memory provided, etc. Additionally, they may include guarantees on the reaction time for scaling the provided infrastructure, when its load increases over a predefined threshold. This last term is, again, impossible to negotiate if the infrastructure provider cannot monitor virtual machine utilization load, while the reaction time may be indicated by the provider's SLA history.

3.2. Negotiation and Renegotiation

The lifecycle of a single SLA starts with its negotiation. In this phase, the service provider and the customer exchange messages in order to agree on a well-defined set of guarantees governing service consumption by the specific customer. Guarantees may refer to interdependent obligations of both parties. This may include, for instance, the minimum performance of the service (provider side) as long as the invocation rate remains under a certain threshold (customer side). The multi-round negotiation process for establishing an SLA is illustrated in Figure 2 [7].

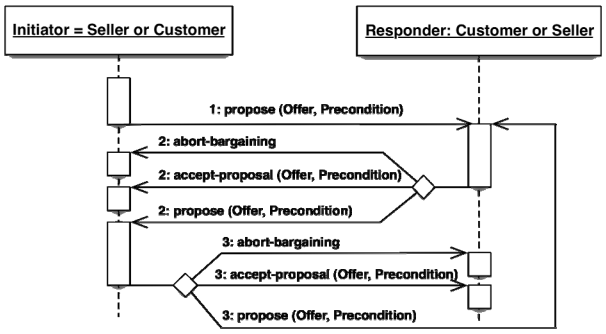

Figure 2: Negotiation process

As already mentioned, during the negotiation phase, both parties are using their knowledge and assumptions for maximizing their profit and the value of the SLA at hand. The exact utility function to be optimized may be different for each party in the negotiation, for each business domain that SLA negotiation may be applied to, or even perhaps for different entities of the same kind in the same domain (e.g., two different cloud-computing providers). Therefore, trying to find a universal solution to the problem of optimal contracting is not possible.

Additionally, assuming an SLA hierarchy as the one shown in Figure 1, it becomes clear that it is not possible to define uniquely and universally an algorithm for the hierarchy's construction and decomposition for any possible agreement term (although, there have been efforts to decompose *performance* terms in a uniform way, e.g. [8]). To address the issue of optimal SLA hierarchy construction, SLA@SOI has employed a large number of different industrial use cases, and will apply the produced framework on them. Through simulation and real-life testing, it is expected to see how different negotiation strategies affect the final contracts in different domains.

One source of existing knowledge, however, that providers should use, is monitoring information from previous consumption of the same service. Overall, we always assume that the provider prefers establishing SLAs with reasonable certainty regarding the offered QoS as a function of the agreement terms, than paying back penalties (as they are defined in the SLA) when deviations occur. Therefore, the provider is expected to utilize historical monitoring information for estimating which terms can indeed be guaranteed with reasonable certainty. Figure 3 illustrates the aforementioned reliance on the monitoring framework from a provider's point of view, showing only one round of these repeated

negotiation steps. The grayed boxes show this relationship explicitly. It should also be noted that Figure 3 assumes that the *agreement initiator* (as defined in WS-Agreement) is the customer, and the *agreement responder* is the provider.

At the same time, service-based systems are highly dynamic. As such, conditions constantly change. Infrastructure that was available at the time of negotiation, for example, may become unavailable during the SLA runtime. Furthermore, concurrent use of hardware or virtual resources results in dependencies between the different SLAs of the provider. Thus, it is often necessary to adjust, re-provision, or eventually, renegotiate SLAs. This process is triggered by monitoring, using events indicating the violation of SLA guarantee terms to which the service provider subscribes, as discussed in the following section.

4. SLA Monitoring

As shown in Figure 3, SLA negotiation introduces two requirements for SLA monitoring:

1. Monitoring should allow the collection of SLA violations during the provisioning of a service under the terms of an SLA. On the Provider side, such violations should be made available as historical data to SLA negotiation, for optimization and planning while deciding whether to accept or not a SLA offer made by the customer;
2. Monitoring should be able to assess the *monitorability* of the guarantee terms specified in a SLA offer made by an agreement initiator to an agreement responder. This is necessary since auditing and enforcing an SLA that has non-monitorable guarantee terms would not be feasible.

The first of these requirements is a typical functional requirement for any generic software system monitoring component [9]. However, in loosely coupled and heterogeneous SLA management scenarios, as the one introduced in Section 2, the realization of the requirement requires advanced monitoring mechanisms. The latter should support the clear specification of the monitoring capabilities for the different components of the service based system and their infrastructures, and protocols for monitoring delegation, availability of primitive monitoring information and dissemination of monitoring results. These issues are discussed in more detail later.

The second requirement, regarding the assessment of the monitorability of SLA terms before SLA establishment, is even more challenging and it has not been addressed by previous work on SBS/SLA monitoring. Thus, it represents one of the main contributions of our approach to SLA monitoring.

The architecture of the SLA monitoring framework of SLA@SOI is shown in Figure 4. As shown in the figure, the architecture consists of four main modules, namely an Event Bus, a Monitoring Terms Derivation module, a Terms Verification module, and Monitor Engine.

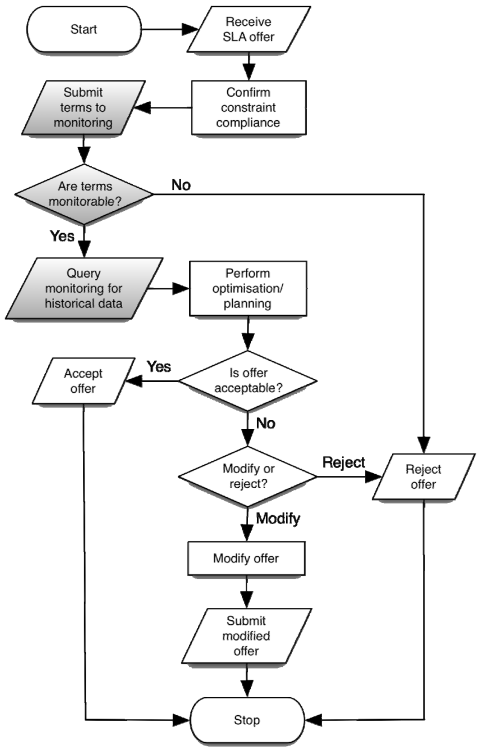

Figure 3: Negotiation from the provider's side

4.1 Components of the monitoring framework

The role and function of the components of the SLA@SOI monitoring framework are as follows:

Event Bus. The architecture of the SLA@SOI monitoring framework is event-based [10], i.e., it relies on capturing runtime information during SLA provisioning at the different services of the managed SBS by suitable event captors and making it available to different components of the monitoring framework as *events*. The exchange of events between the monitor and the event captors (internal to a node or from external nodes) is managed through an Event Bus that realizes a publish/subscribe architecture. In this architecture, event captors are event publishers and monitors are event subscribers and consumers. More specifically, event captors publish their events to the bus with appropriate tags enabling it to distribute them to monitors that have subscribed to them. Based on these events the monitors can detect violations of the terms of SLAs. Note, however, that monitors can also act as event publishers themselves notifying their results as events as well (events of this type will be referred as "monitoring result events" in the following). Thus, it is possible to use the framework to coordinate different monitors in various formations (hierarchical, peer-to-peer etc) as required for the particular SLAs that

need to be monitored and/or other constraints of the overall SBS infrastructure.

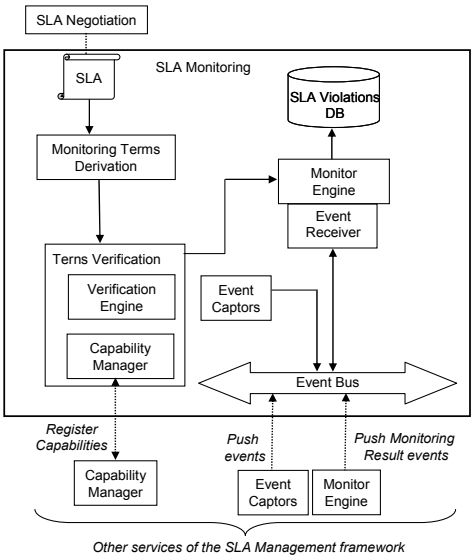

Figure 4: SLA monitoring architecture

Monitoring Terms Derivation Module. The role of this module is to translate the agreed guarantee terms of an SLA into specifications of patterns of events and computations over their features that can be checked at runtime. In the prototype implementation of the framework the language that is used to express the monitorable event patterns is *EC-Assertion*, i.e., an XML language based on Event-Calculus [10]. This is because the default monitor of the SLA@SOI monitoring framework is the *EVE*nt *REaS*oning *T*oolkit (*EVEREST*) [11] that supports this language. Note, however, that the architecture of the monitoring framework allows the integration of other Monitoring Terms Derivation Modules to support different languages for expressing guarantee terms and monitors.

Monitor Engine. Monitoring service based systems has been an area of focus lately and several systems have been proposed for monitoring composite or atomic services, e.g. [12], and service infrastructures, e.g. Ganglia (http://ganglia.info). In our approach, SLA monitoring in each node may adopt a different Monitor Engine. The logic implemented by the Monitoring Terms Derivation module will then change according to the kind of properties/rules required by the adopted Monitor Engine. Detected SLA violations are stored in the SLA Violations DB, which is queried by SLA Negotiation when historical data are required for accepting/refusing a SLA offer.

Terms Verification Module. This module implements the main functionality required for assessing terms' monitorability. It receives as input the Monitoring terms, as obtained from the translation made by the Monitoring Terms Derivation, and assesses whether the terms can be monitored through a call to the Capability Manager. Monitoring capabilities and the Capability Manager functionality are described in Section 4.2.

As discussed previously, the provision of runtime events to the SLA Monitoring framework is based on Event Captors. Event Captors are able to capture events generated by the SLA provisioning environment, and may be implemented differently depending on the entity that they need to provide information for.

Event captors may, for example, be realized as instrumented BPEL processes in the case of composite software services implemented by BPEL service coordination workflows, which during execution can emit the required events [12] and state of the executing workflow. Service invocations and matching responses are typical examples of events that can be captured at the BPEL process execution level. Such events are required, for instance, for monitoring the *Completion Time* agreement term as defined in Section 3.1. In other cases they may be realized as service container/proxies that capture service calls and responses [9]. At the infrastructure layer, specialized event captors may also be deployed. Virtual machines may, for instance, have their own mechanisms for monitoring *Availability*, *MTBF*, or *MTTR*. Alternatively, they may be able to capture events informing the monitor engine when a service becomes unavailable, and when it becomes available again. We therefore implicitly extend the SLA hierarchy to a hierarchy of rules for constructing events, based on which we can monitor higher-layer SLAs using in a straightforward manner the SLAs that constitute them.

Note that, regardless of their implementation, event captors need to timestamp the events that they generate and, depending on the consumer of these events, even synchronize their clocks with the clock of a reference monitor [13]. Time stamping is critical for monitoring SLAs as most of the terms in them need to be expressed in relation to time (see the Completion Time, Throughput, and Accessibility agreement terms defined in Section 3, for example).

4.2 Monitoring capabilities and monitorability assessment

The assessment of the monitorability of SLA terms relies on the definition of the monitoring capabilities of each service involved in the SLA Management Framework. The Monitoring Capabilities of a service are defined as the collection of (i) the Events that can be produced by its local Event Captors and (ii) the Monitoring Result Events that can be produced by its Monitor Engine, that is, the kind of agreement terms a service may locally monitor if requested to do so. The exchange of monitoring capabilities between two services in the SLA management framework is implemented as the exchange of (XML-based) monitoring capabilities

documents among the Capability Managers of the two services.

Because of SLA hierarchies, we envisage the process of exchanging capabilities to be hierarchical.

As an example, based on the scenario of Figure 1, we show how CS can assess the monitorability of the terms in an offer for SLA_CS submitted by the customer. In order to assess the monitorability of the terms in this offer, CS must be made aware of the monitoring capabilities of other services in the SLA hierarchy, i.e. IS1, IS2, AS1, and AS2. However, a service in the SLA management framework can be aware only of its *peers*, that is, the other services with which it is negotiating an SLA. In our example, IS1, AS1, and AS2 are the peers of CS, whereas IS2 is a peer for both AS1 and AS2. Therefore, CS first requests the monitoring capability documents to its peers, i.e. IS1, AS1, and AS2. While IS1 can immediately reply with its monitoring capabilities, since it has no peers down the SLA hierarchy, AS1 and AS2 first issue a request for the monitoring capabilities document to their peer, i.e. IS2. The capability document sent back by AS1 and AS2 to CS includes also the monitoring capabilities of IS2. In this way, after monitoring capabilities documents have been exchanged, CS is aware of the monitoring capabilities of its peers. It should be noted that the exchange of monitoring capabilities triggered by the top-level SLA (i.e. SLA_CS in our example), which is negotiated with the consumer, enables also all the other services to assess the monitorability of terms in other SLAs down the hierarchy. Therefore, each service is able to assess the monitorability of terms in an SLA offer. For instance, AS1 can now assess the monitorability of SLA_AS1 offers, since it is aware of IS1's monitoring capabilities.

When a service receives an SLA offer, the generated Monitoring Terms are submitted to the Terms Verification module. The Terms Verification module will retrieve the (hierarchically defined) monitoring capabilities from its Capability Manager. Then, for each term, the Terms Verification module verify whether (i) events required for monitoring the term are available or (ii) the monitoring of the term can be delegated to another service in the hierarchy.

In case (i), the term will be monitored locally by the service, consuming the required events that will be published on the bus by Event Captors (local and from other peer services). In case (ii), the monitoring of the term can be delegated to another service down the hierarchy. If the monitoring of a term can not be performed locally, i.e. required events are not available according to the exchanged monitoring capability documents, or delegated to other services, the SLA monitoring will notify the SLA negotiation that the term can not be monitored. Therefore, the agreement offer will be rejected (or modified for further negotiation steps).

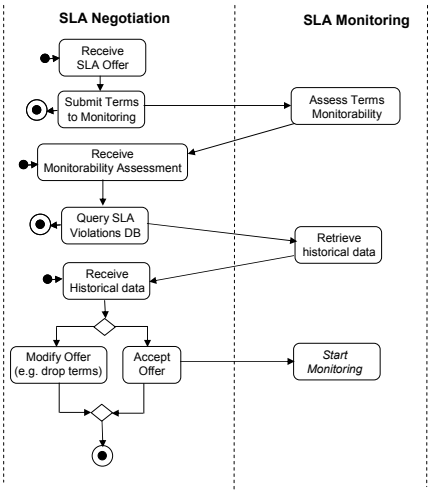

Figure 5: Interactions between SLA negotiation and monitoring

At runtime, when the SLA is provisioned, the Event Bus of the service will subscribe to the events required for monitoring or to the correspondent Monitoring Result event registered by other services, to which the monitoring of some terms has been delegated. A service's Monitor Engine, e.g. CS's in our example, will then start receiving the events to which it has subscribed. Generic events are processed by the Monitor Engine to assess SLA violations, whereas Monitoring Result events are directly stored by the Event Receiver in the SLA Violations DB.

As a conclusion, Figure 5 explicates the negotiation-time offer evaluation flow described in Figure 3, showing how SLA Negotiation acts as a client of SLA Monitoring, which exposes three atomic functionalities, i.e. *Verify Monitorability*, *Retrieve Historical Data*, and *Start Monitoring*. On the one hand, *Verify Monitorability* fulfills the requirement (2) identified in Section 4, i.e. the need for assessing the monitorability of agreement terms in an SLA offer, according to the exchange of monitoring capabilities previously described. On the other hand, *Retrieve Historical Data* and *Start Monitoring* functionalities jointly fulfill requirement (1), i.e. making monitoring data available for the evaluation of SLA offers. The former functionality, in particular, is implemented by a set of queries that SLA negotiation may run on the SLA Violations DB.

5. Evaluation of Design Choices

An initial, rapid prototype of the SLA Management framework and, in particular, SLA Negotiation and Monitoring, is available to support a reference scenario of a retail solution, for which the service and SLA hierarchy is structured as in Figure 1. A second iteration on the software stack is prepared and the framework will be

evaluated in real world business use cases, such as e-government, service aggregator, and financial grids.

For what concerns monitoring, the prototype exploits the core monitor engine described in [11], while, the Event Bus is based on a public implementation of the XMPP-PubSub. The choice to rely on publicly available specifications of the bus has been made to guarantee future interoperability with other external event captors.
In the current implementation, the translation of rules is statically made, In particular, EC rule templates, based on a set of pre-specified set of events, have been defined for each type of agreement term defined in Section 3.1. Templates are instantiated in concrete rules by adding information on service endpoint references and negotiated values of agreement terms contained in the SLA. Services monitoring capabilities are defined by the signatures of events used in monitoring rules templates. In this way, the assessment of monitorability is reduced to the problem of matching the concrete monitoring rules with the signatures of events reported in monitoring capabilities documents of services involved in SLA provisioning. The second iteration of the software stack should remove the coupling between rule templates and event signatures, adopting higher-level definitions of event signatures that could be matched against several formalisms adopted to express concrete monitoring rules/properties.

With regard to negotiation, the current prototype takes advantage of monitoring as explained above, to verify that specific terms can actually be monitored. At the same time, monitoring information from previous SLAs provides simple averages that indicate whether a SLA offer should be accepted or not, based on the service performance logged in the past. What is currently missing from this prototype is the capability for multi-round negotiation, which is necessary in environments such as the one under discussion. For the time being, a WS-Agreement implementation has been adopted, providing single-round interactions with the offers followed by responses declaring only acceptance or rejection. The project is actively participating in the Open Grid Forum and seeks to affect WS-Agreement with regard to full negotiation capabilities, which will eventually be implemented as part of the framework.

6. Related Work

SLA negotiation and SLA monitoring have been heavily researched in the past, but the two research streams have usually been kept separated. In some cases, they have been brought together in more unified architectures, but never viewed in such a way where negotiation relies on monitoring and vice versa, in a fully dynamic context taking into account multi-layered SLA hierarchies.

For what concerns runtime monitoring of SBS, intrusive monitoring relies on alternating the execution of the service and monitoring activities at runtime. This can be done directly in the BPEL engine, interleaving monitoring code with the process executable code [9]. System properties' monitorability can not be achieved with intrusive monitoring, since the properties to be monitored and the actions required for monitoring must be interleaved with service execution code and, therefore, known a priori by the system designer. Non-intrusive monitoring [10, 15, 12, 16] requires the establishment of mechanisms for capturing runtime information on service execution, e.g. service operation calls and responses. In this way, the business logic of the SBS process and the monitoring logic remain separate. The cited approaches to non-intrusive monitoring take for granted the availability of events required for monitoring and do not consider the issue of monitorability of rules/properties submitted to a generic monitor engine. The concept of local monitors attached to services has been introduced in [27]. However, the proposed approach considers the static allocation of properties monitoring based on a predefined service network topology.

A multitude of research papers discuss the topic of SLA negotiation with some reference to monitoring, but without exploring it explicitly in the context of a complete, multi-layer service economy. [17] is using a "Situation Assessment Module" to evaluate the feasibility of a SLA based on monitoring info, but only looking at isolated SLAs. Conversely, [18] and [19] are looking into SLA hierarchies and negotiation in this context, without any reference to consultation with monitoring though. In [20] the authors refer to using events for evaluating the validity of offers, but without further discussion on using monitoring for provider-side optimization of the negotiation process. In [21] a negotiation framework is presented and decision strategies are mentioned, but without any explicit links to monitoring information.

Several projects have also focused on SLA definition, establishment, and provisioning both in the context of Web and Grid services. Project NextGRID is probably the one closest to what SLA@SOI is also discussing. NextGRID foresaw the need for SLA hierarchies [22], however the monitoring and profiling infrastructure does not take it into account [23]. Adaptive Services Grid (ASG) designed an architecture where negotiation uses profiling data, but not monitoring data from previous violations. Also, the monitoring rules and parameters are static and pre-defined [24]. Finally, inter-dependencies of SLAs are not discussed at all. The TrustCOM project looked deeply into the subject of SLA negotiation and monitoring, and also produced a reference implementation. However, SLA hierarchies and dependencies are not taken into account, and the problem is solved for isolated agreements only [25]. The same holds for AssessGrid, which concentrated on SLAs and risk management [26]. Also, AssessGrid has a focus on

Grid computing, therefore assuming certain system organization and architecture, while our approach has a wider view on autonomic service providers and the respective service economies.

7. Conclusions and future work

After illustrating and analyzing the explicit link between SLA negotiation and SLA monitoring, we presented a novel architecture for establishing and monitoring SLA hierarchies spanning through multiple domains and layers of a service economy: Business, software and infrastructure services. We showed why this relationship cannot be disregarded, especially in such complex hierarchies, and how a SLA hierarchy reflects on the monitoring hierarchy.

Besides applying the framework to industrial use cases and addressing the open design issues discussed in Section 5, we also plan to broaden our SLA management scenario by considering requirements for SLA negotiation and monitoring on the service consumer side, i.e. focusing on mechanisms for SLA offer negotiation on the consumer side and on how consumer-generated monitoring data may be integrated in the service provider SLA monitoring framework presented in this paper.

Acknowledgments

The research has been supported by the EU Commission under the SLA@SOI Project (grant agreement n. 216556).

References

[1] Papazoglou, M.P., Service-Oriented Computing: Concepts, Characteristics and Directions. Proc. 4th Conference on Web Information Systems Engineering, 2003.

[2] L. Wang, G. von Laszewski, M. Kunze, J. Tao. Cloud computing: A Perspective study. Proc. Grid Computing Environments (GCE) workshop, 2008.

[3] A. Andrieux et al.; Web Services Agreement Specification (WS-Agreement). The Open Grid Forum, March 2007. http://www.ogf.org/documents/GFD.107.pdf

[4] Dobson, G. and Sanchez-Macian, A.; Towards Unified QoS/SLA Ontologies. Services Computing Workshops, SCW '06, Sept. 2006

[5] Maximilien, E.M. and Singh, M.P.; A framework and ontology for dynamic Web services selection. IEEE Internet Computing, 8(5), Sept.-Oct. 2004, pp. 84 – 93.

[6] D. Colling, T. Ferrari, Y. Hassoun, C. Huang, C. Kotsokalis, A.S. McGough, E. Ronchieri, Y. Patel and P. Tsanakas. On Quality of Service Support for Grid Computing. Grid Enabled Remote Instrumentation, Springer US, 2009.

[7] D. Somefun, E. Gerding, S. Bohte, J. La Poutré. Automated Negotiation and Bundling of Information Goods. Agent-Mediated Electronic Commerce V, pp.1-17, 2004

[8] Y. Chen, S. Iyer, X. Liu, D. Milojicic, A. Sahai. SLA Decomposition: Translating Service Level Objectives to System Level Thresholds. Int. Conf. on Autonomic Computing, 2008.

[9] L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes, Proc. ICSOC 2005.

[10] Spanoudakis G., Mahbub K.: Non Intrusive Monitoring of Service Based Systems, International Journal of Cooperative Information Systems, 15 (3), pp. 325-358, 2006.

[11] Spanoudakis G, Kloukinas C. Mahbub K.: The SERENITY Runtime Monitoring Framework, In *Security and Dependability for Ambient Intelligence*, Information Security Series, Springer, pp. 213-238 (to appear)

[12] F. Barbon, P. Traverso, M. Pistore, M. Trainotti, Run-Time Monitoring of Instances and Classes of Web Service Compositions, Proc. IEEE ICWS 2006.

[13] Kloukinas, C., Spanoudakis, G., and Mahbub, K. Estimating Event Lifetimes for Distributed Runtime Verification, Proc. SEKE 2008.

[14] Keller, A. and Ludwig, H. 2004. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. J. of Network and Systems Management, 11(1), 57-81.

[15] W.M.P. Van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek, Conformance checking of Service Behavior, ACM TOIT, 8 (3), May 2008.

[16] O. Moser, F. Rosenberg, and S. Dustdar, Non-intrusive monitoring and service adaptation for WS-BPEL, WWW 2008.

[17] N. R. Jennings, T. J. Norman, P. Faratin, P. O'Brien, B. Odgers. Autonomous Agents For Business Process Management. Applied Artificial Intelligence, 2000, 14, 145-189.

[18] M.B. Chhetri, J. Lin, S. Goh, J.Y. Zhang, R. Kowalczyk, J. Yan. A Coordinated Architecture for the Agent-based Service Level Agreement Negotiation of Web Service Composition. Proc. Australian Software Engineering Conference, 2006.

[19] J. Brzostowski, M.B. Chhetri, R. Kowalczyk. Three Decision-making Mechanisms to facilitate Negotiation of Service Level Agreements for Web Service Compositions. Proc. Joint Conference of the INFORMS Section on Group Decision and Negotiation, 2007, pp. 37-44.

[20] M.R. Ayatollahzadeh Shirazi, A.A. Barfouroush. A Conceptual Framework for Modeling Automated Negotiations in Multiagent Systems. Negotiation Journal, 2008, 24(1), pp. 45-70.

[21] E. Di Nitto, M. Di Penta, A. Gambi, G. Ripa, M. Villani. Negotiation of Service Level Agreements: An Architecture and a Search-Based Approach. Proc. ICSOC 2007, pp. 295-306.

[22] D. Snelling, A. Anjomshoaa, F. Wray, A. Basermann, M. Fisher, M. Surridge, P. Wieder. NextGRID Architectural Concepts. Towards Next Generation Grids: Proc. CoreGRID Symposium, 2007.

[23] K. Tserpes, D. Kyriazis, A. Menychtas, T. Varvarigou, F. Silvestri, D. Laforenza. An Open Architecture for QoS Information in Business Grids. Towards Next Generation Grids: Proc. CoreGRID Symposium, 2007.

[24] K. Jank, Reference Architecture. Adaptive Services Grid Deliverable D6.V-1, 2005.

[25] The TrustCOM project. Deliverable 64: Final TrustCoM Reference implementation and associated tools and user manual. June 2007 (v3.0).

[26] J. Padgett, I. Gourlay, K. Djemame (eds). AssessGrid D1.3: System Architecture Specification and Developed Scenarios (v0.30). December 2006.

[27] Machiraju, V., Sahai, A., and van Moorsel, A. Web Services Management Network: An Overlay Network for Federated Service Management, Proc. IFIP/IEEE 8th Int. Symposium on Integrated Management, 2003.