A Framework for the Management of Dynamic SLAs in Composite Service Scenarios

Giuseppe Di Modica, Orazio Tomarchio, and Lorenzo Vita

Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania
Viale A. Doria 6, 95125 Catania - Italy
Firstname.Lastname@diit.unict.it

Abstract. The advent of information and communication technology has changed the nature of business-to-business interaction among organizations. The use of electronic contracts with automated support for their management allows an increase of effectiveness and efficiency in contract processing, opening new possibilities for interaction among parties. Service Providers and their customers negotiate utility based Service Level Agreements (SLA) to determine costs and penalties based on the achieved performance levels. The global QoS to be provided to the end customer can be strongly affected by any violation on each single SLA. In order to prevent such violations, SLAs need to be flexible and dynamically adaptable. In this work we focus on the WS-Agreement specification, a Web Service protocol to establish agreements on the QoS level to be guaranteed in the provision of a service. We propose to enhance the flexibility of its approach by integrating new functionality to the protocol that enable the parties of a WS-Agreement to re-negotiate and modify its terms during the service provision.

1 Introduction

Service Oriented Architectures (SOA)[8,9] have recently appeared as an emerging paradigm for automated business integration. In a SOA environment services are implemented on top of resources across different administrative domains, in which IT infrastructure and internal resources are managed according to autonomous policies. When a customer wants to use a service offered by a Service Provider, an agreement is needed, in the same way of a traditional service. A Service Level Agreement [7] is a formal negotiated agreement between a Service Provider and a Service Customer (the service requester). The use of electronic contracts with automated support for their management allows an increase of effectiveness and efficiency in contract processing, opening new possibilities for interaction among parties [2,3].

The high level of interoperability offered by the SOA enables scenarios of world-wide and cross-domains service composition. In a scenario where the service to be provided to a Service Customer has to be built by composing a number of services provided by as many Service Providers, some of the involved parties may act as both customer and provider: they need to access one or more services in order to build and provide a given service, which in turn may be used by another party to build its own service. In such a scenario several one-to-one SLAs have to be signed by the parties contributing to the provision of the final service to the Customer. The overall QoS perceived by the Customer strictly depends on the fulfillment of each of these Agreements. If just one of these Agreements for any reason had to be terminated, the provision of the service to the Customer might have to be stopped. There is a need for a flexible mechanism enabling the run-time re-negotiation of the guarantees on the QoS once violations on such guarantees are expected to occur. The objective would be to avoid the suspension of the service provisioning and the brutal termination of the agreement, which is not convenient for both Service Providers and Service Customers.

In this work we present a framework for the management of flexible and dynamic SLAs in scenarios of service compositions. Our attention has been focused on the WS-Agreement [1] specification, a Web Service protocol to establish agreements on the QoS level to be guaranteed in the provision of a service. A Service Level Agreement compliant to the current WS-Agreement specification is a rigid contract that does not satisfy the requirements characterizing scenarios of service composition. We propose modification to the current WS-Agreement specification in order to introduce a run-time support to the re-negotiation of the QoS levels' guarantees stipulated in the contract by the signing parties.

The rest of the paper is organized in the following way. In the next Section the motivation for this work is discussed. In Section 3 we briefly introduce the WS-Agreement specification. Section 4 will describe the enhancement to the WS-Agreement specification proposed in our approach. Related work is presented in Section 5, and finally we conclude the paper in Section 6.

2 Rationale of the Work

Very often, because of the Service Provider's inability to meet the Agreement's Service Level Objectives (SLOs), the provision of the service that the Agreement applies to is brutally stopped, and the Agreement is terminated. The more services are needed to fulfill that task, the higher the probability that the task is not accomplished.

For instance, let us consider the case of a Customer C that wants to access a service offered by the Service Provider SP_1, which in its turn needs to access a service offered by the Service Provider SP_2 to satisfy the Customer's request. Again, let us assume that the service offered by SP_2 is built on top of another service offered by the Service provider SP_3. For each service provision an Agreement will have to be signed. In the figure 1(a) a graphic representation of this scenario is given.

The involved parties are represented by nodes, while the oriented edges represent the Agreements signed between pairs of parties and the respective roles that the parties play in the Agreements. For instance, the edge Ag_{21} oriented from SP_2 to SP_1 represents the Agreement signed by the parties SP_1 and SP_2,

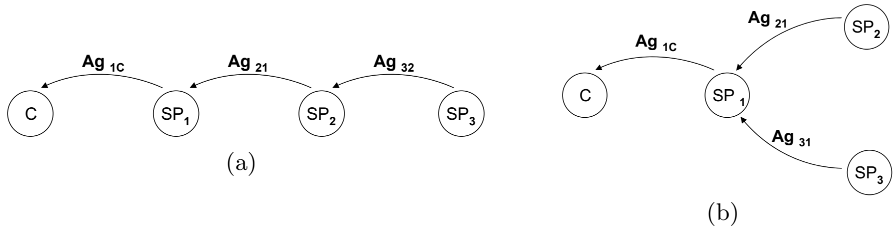

Fig. 1. Simple service-composition scenario: chain (a) and fork (b)

being SP_2 the Service Provider and SP_1 the Service Consumer for that specific contract. The abnormal termination of a given Agreement causes an interruption in the Agreements' chain, thus preventing the forwarding Agreements from being fulfilled. For instance, if SP_2 for any reason did not attain the QoS target or stopped providing its service to SP_1, thus violating the Agreement Ag_{21}, SP_1 would not be able to offer its service to C and then to honor the Agreement Ag_{1C}. In this case, either it searches for another service provider offering the same service as offered by SP_2 or it has to terminate the Agreement Ag_{1C}. Furthermore, in the latter case SP_2 might not need to access the service offered by SP_3 anymore (unless it has pending requests that might be satisfied with that service), and will likely decide to terminate the Agreement Ag_{32}.

Let us consider another scenario, in which a Service Provider needs to use two different services to offer its own service to the customer, as depicted in the figure 1(b). In this case, if one of the services that SP_1 relies on was stopped or did not attain the QoS target, the provision of the final service to the Customer would be compromised. The violation of just one between the Agreements Ag_{21} and Ag_{31} may cause the violation of the Agreement Ag_{1C}.

In a more complex scenario, resulting from the composition of *chains* and *forks* as depicted in the previous two figures, several Service Providers are involved in the building of a service for the Customer and the probability of violation of the Agreement regulating that service is very high. Summarizing, in a service composition tree, the "breaking off" of an edge (Agreement) linking any two nodes (service providers) may propagate throughout the tree affecting any other edge. As long as an Agreement is considered to be a static and inflexible contract that regulates a transaction between two parties, the delivery of a service in service-composition scenarios involving multiple one-to-one Agreements among parties has a few chances to be successfully carried out. A new concept of Agreement need to be adopted. An Agreement should be a dynamic contract capable of reacting and adapting to changes in the context it works on. A high degree of flexibility is needed in order to reduce, during the provision of the service that the contract applies to, any possible risk of violation of the QoS guarantees agreed by the parties at the time of the contract sign. The flexibility that we refer to consists in the possibility of 1) re-negotiating at run-time (i.e., during the service provision) the QoS guarantees, 2) accordingly modifying the terms of the contract and 3) adjusting the service provision to the newly agreed

QoS levels. This process, of course, must preserve the continuity of the service provision, i.e., the service flow is neither interrupted nor suspended while the contract is being re-negotiated and modified.

3 The WS-Agreement Specification

The WS-Agreement specification [1], developed by the GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Scheduling and Resource Management (SRM) Area of the Open Grid Forum (OGF), is a Web Service protocol to establish agreements on the QoS level to be guaranteed in the provision of a service. The Objective of WS-Agreement is to define a language and protocol for:

– Establishing agreements between two parties
– Advertising the capabilities and requirements of service consumers and providers
– Creating agreements based on creation offers
– Monitoring the agreement compliance at runtime

This protocol uses an XML-based language for specifying the nature of an agreement, and *agreement templates* to facilitate the discovery of compatible agreement parties. It is generally aimed to be a "one-shot" interaction, and is not directly intended to support negotiation. The agreement creation process is restricted to a simple request-response protocol: one party creates an agreement document, possibly based on an agreement template, and proposes it to the other party. The responding party evaluates the agreement's offer and assesses its resource situation before accepting or rejecting the offer.

An agreement between a Service Consumer and a Service Provider specifies one or more *Service Level Objectives* (SLO) both as expressions of requirements of the Service Consumer and assurances by the Service Provider on the availability of resources and/or on service qualities. The specification provides a schema for defining the overall structure for an agreement document. It contains information about the agreement parties and a set of "terms". The agreement terms represent contractual obligations and include a description of the service as well as the specific guarantees given. A *service description term* can be a reference to an existing service, a domain specific description of a service or a set of observable properties of the service. A *guarantee term*, on the other hand, specifies non-functional characteristics in service level objectives as an expression over properties of the service, an optional qualifying condition under which objectives are to be met, and an associated business value specifying the importance of meeting these objectives.

The conceptual model for the architecture of system interfaces based on WS-Agreement has two layers: the Agreement Layer and the Service Layer. The agreement layer implements the communication protocol used to exchange information about SLAs between the Service Customer and the Service Provider (create, represent and monitor agreements), and it is responsible for ensuring that the SLA guarantees are enforced by a suitable service provider. This layer handles well-formed requests to the service layer. The Service Layer represents the application-specific layer of the service being provided. The interfaces in this layer are domain-specific, and need not be altered when the Agreement layer is introduced.

4 A Framework for the Dynamic Re-negotiation of SLAs

A Service Level Agreement compliant to the current WS-Agreement specification is a rigid contract whose terms can not be modified during its life cycle.

The protocol for the creation of a WS-Agreement is based on a very simple Request-Response interaction between the parties. In theory there is no time limit to the response time of a WS-Agreement creation offer, thus the proposal originator, once issued the offer, either waits indefinitely for a response or decide to withdraw it. When a WS-Agreement has been created it can not be modified and is effective until all the activities pertaining to the Agreement are finished or until one of the signing party decides to terminate it. The main reason for a Service Provider to terminate an Agreement is its inability to provide a service that maintain the QoS levels that were guaranteed in the Agreement. Therefore, there is a need for a flexible mechanism enabling the run-time re-negotiation of the guarantees on the QoS once violations on such guarantees are expected to occur. On the other hand, a mechanism that permits the re-negotiations of the guarantees on the QoS would also enable Service Providers and Service Customer respectively to offer and to ask for upgradings of the performance of the service provided at run-time.

The new protocol being designed must take into account the dynamics and the new requirements that the scenarios presented in section 2 impose. We remark that in such scenarios there are several actors, in the dual role of provider and consumer of services, that stipulate one-to-one agreements with each others. We must consider that:

- a request for the creation of an Agreement often triggers several requests of Agreement creation in a cascade fashion, giving rise to a "network" of correlated Agreements;
- a request of modification of a given Agreement may trigger modification requests for the Agreements correlated to it;
- the modification to be applied to an Agreement must be such as not to justify the cancellation of the Agreement and the creation from scratch of a new Agreement;
- the party receiving a modification proposal must respond within a useful time that is specified by the proposal originator, otherwise the proposal shall be considered withdrawn;
- the number of modifications that an Agreement can undergo at run-time must be agreed by the parties at sign time;

Focusing on the requirements just described, in this work we enhance the flexibility of the WS-Agreement approach by integrating new functionality to the protocol that enable the parties involved in a scenario of service composition to re-negotiate the levels of quality of the services and modify the guarantee terms of the Agreements while services are being provided. The proposed protocol does not support the process for the re-negotiation of the QoS levels, but provides the means to adjust the terms of an Agreement accordingly to the outcome of an eventual negotiation process. The main modification being introduced to the current specification concerns some integrations to the XML schema of the Agreement and an enhancement of the protocol with new interactions for the run-time modification of the Agreement.

4.1 Modification to the XML Schema

We believe that the parts of an Agreement that might be modified at run-time, while preserving at the same time the nature of the Agreement itself, are those concerning the guarantees on the Service Terms. Specifically, in an Agreement document the terms susceptible of modification are the *Service Level Objectives(SLOs)* in the *Guarantee Terms* section. In particular, we have introduced a new concept of SLO that can be modifiable according to some rules that limit both the time interval of modifiability of the SLO and the number of modifications that the SLO can undergo. Such parameters, once agreed by the parties at signing time, can not be further modified during the Agreement's lifetime. Instead, what can be modified at runtime is the objective itself to be achieved.

In the following we report the excerpt of the Agreement's XML schema regarding the modifications that have been added.

```
<xs:element name="ModifiableServiceLevelObjective" ..............................
 type="wsag:ModifiableServiceLevelObjectiveType"/>

<xs:complexType
   name="ModifiableServiceLevelObjectiveType">
 <xs:sequence>
  <xs:element name="ModificationWindow"
        type="xs:TimeWindowType"/>
  <xs:element name="MinModificationsTriggerTime"
        type="xs:PercentageType"/>
  <xs:element name="Slo"
        type="xs:SloType"
        maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="SloType">
  <xs:sequence>
   <xs:element name="Objective"
        type="xs:anyType"/>
   <xs:element name="ScheduledTime"
        type="wsag:PercentageType"/>
  </xs:sequence>
</xs:complexType>
.......................................
```

```
<xs:complexType name="TimeWindowType">
 <xs:sequence>
   <xs:element
        name="NotEarlierThan"
        type="PercentageType"/>
   <xs:element
        name="NotLaterThan"
        type="PercentageType"/>
 </xs:sequence>
</xs:complexType>

<xs:simpleType name="PercentageType"
 <xs:restriction base="xs:integer">
   <xs:minInclusive value="0"/>
   <xs:maxInclusive value="100"/>
 </xs:restriction>
</xs:simpleType>
```

First of all, it is worth noticing that an Agreement's XML document conforming to the current specification's Agreement XML schema is conformant to the new schema too. The novelties introduced do not prevent Service Providers and Service Customers from stipulating Agreements based on the old concept of unmodifiable SLOs. At protocol's design time our purpose was, in fact, to maintain the backward compatibility with the original specification. The rest of the modifications that have been introduced do not alter the original protocol's functionality, but rather integrate new ones.

The modifiable SLO is composed of three elements: *ModificationWindow*, *MinModificationsTriggerTime* and *Slo*. In particular, *ModificationWindow* represents the time lapse during which carrying out the modification of the SLO is allowed. The upper bound and the lower bound of the interval are expressed as a percentage of the entire life cycle of the Agreement, which spans from the time that the Agreement proposal is accepted to the Agreement's expiration time (which is specified by the parameter *expirationTime* in the Agreement document). With the *MinModificationsTriggerTime* element the parties agree on the minimum time lapse between two subsequent modifications to be carried out on a given SLO. It has a percentage type, and is to be meant as a fraction of the *ModificationWindow* interval. This value, as showed later, fixes the maximum number of modifications that can be carried out on a SLO The *Slo* term represents the scheduling of a modification of the service level objective. It is composed of the *Objective* element, which contains the specification of the new service level objective, and of the *Scheduled* element, of percentage type, from which it is possible to come to the scheduling time of the modification.

4.2 Enhancement to the WS-Agreement Protocol

We have so far established the requirements and the criteria for the modification of an Agreement. Now we describe the means available to the parties of an Agreement to handle its modification.

The current WS-Agreement protocol provides only one single interaction. The Agreement Initiator (AI from now on) makes an Agreement offer and the Agreement Responder (AR from now on) is called to accept or reject it. We introduce a new form of interaction that enables both the parties to request modifications of the guarantees of an earlier signed Agreement. Either the AI or the AR, therefore, according to the above mentioned criteria, can make offers to modify one or more Agreement's SLOs; such modification becomes effective only after the other party has accepted the proposal. We stress that after that an Agreement's modification offer has been issued, the service provisioning is not interrupted, nor is the Agreement's monitoring suspended. If the responding party accepts the proposal, the Service Provider will have to adjust the QoS to the new agreed levels; if the proposal is rejected, the service provisioning will continue according to the original QoS levels. The rejection of an Agreement's modification offer, in fact, does not invalidate the Agreement itself. The same applies to eventual multiple modification offers: if the (i+1)-th modification proposal is rejected, the guarantees agreed in the context of the i-th proposal still apply.

In the following we detail the new Port Types and Operations that have been introduced for the integration of the described functionality. First of all, in order to be able to accept modification offers, each party must provide the other with an ad-hoc contact point (or End Point Reference - EPR). We have introduced a new operation for the creation of *Modifiable Agreements*. Such operations differ from the one provided in the current WS-Agreement specification (CreateAgreement) by the fact that the two parties exchange each other's EPRs that are specific for receiving Agreement modification proposals.

We thus have added the *CreateModifiableAgreement* operation to the "AgreementFactory" Port Type. Its signature is reported below.

wsag:CreateModifiableAgreement

```
 Input

<wsag:CreateModifiableAgreementInput>
  <wsag:InitiatorAgreementEPR>
    <wsa:EndpointReference>
        wsa:EndpointReferenceType
    </wsa:EndpointReference>
  </wsag:InitiatorAgreementEPR> ?
  <wsag:InitiatorAgModRequestEPR>
    wsa:EndpointReferenceType
  </wsag:InitiatorAgModRequestEPR>
  <wsag:AgreementOffer>
         ...
  </wsag:AgreementOffer>
   <wsag:NoncriticalExtension>
     <xs:any> ... </xs:any>
  </wsag:NoncriticalExtension> *
   <xs:any> ... </xs:any> *
</wsag:CreateModifiableAgreementInput>
```

```
 Result

<wsag:CreateModifiableAgreementResponse>
  <wsag:CreatedAgreementEPR>
      wsa:EndpointReferenceType
  </wsag:CreatedAgreementEPR>
  <wsag:ResponderAgModRequestEPR>
       wsa:EndpointReferenceType
  </wsag:ResponderAgModRequestEPR>
  <xs:any> ... </xs:any> *
</wsag:CreateModifiableAgreementResponse>
```

Similarly, we added the *CreateModifiablePendingAgreement* operation to the "PendingAgreementFactory" Port Type. Its signature differs from that of the *CreateModifiableAgreement* operation for just one input parameter. Since it is not relevant to our work, we omitted to report its details.

For the provision of a given service, a party signs as many Agreements as needed to provide that service. When that party for any reason wants to modify one of these Agreements, it first has to ask itself what the consequences of that modification might be, i.e., what that modification might mean for the rest of the Agreements in that scenario. After the modification of a given Agreement, in fact, the party might not be able to honor the rest of the Agreements signed with its neighbors, and the neighbors, in their turn, might not be able to honor the Agreements signed with their own neighbors, and so on. This means that a proposal of Agreement modification, wherever it is originated, can propagate throughout the Agreement tree, theoretically affecting every Agreement. The acceptance of a given modification proposal is bound to the acceptance of other Agreements' modification proposals in the tree structure.

To support the dynamic modification of Agreements in the WS-Agreement specification, we introduce the "AgreementHandler" Port Type. This Port Type must be implemented by every party interested in taking part in the management of dynamic Agreements. Two operations are exposed through this Port Type: *ModifySLO* and *CondModifySLO*. The *ModifySLO* operation is defined as follows.

wsag:ModifySLO

```
 Input                                                  Result

<wsag:ModifySLOInput>                                  <wsag:SLOModificationResponse>
    <wsag:SLOModificationAcceptanceEPR>                  <xs:any> ... </xs:any> *
        <wsa:EndpointReference>                        </wsag:SLOModificationResponce>
             wsa:EndpointReferenceType
        </wsa:EndpointReference>
    </wsag:SLOModificationAcceptanceEPR>
    <wsag:SLOModificationOffer>
            ...
    </wsag:SLOModificationOffer>
    <wsag:RequestExpirationTime>
            xs:dateTime
    </wsag:RequestExpirationTime>
</wsag:ModifySLOInput>
```

The result parameter is just an acknowledgement for the reception of the modification offer; it does not represent the response to the offer itself. The input parameter SLOModificationOffer represents the offer of an Agreement where modifications have been introduced to any of the modifiable SLO. The input paramenter RequestExpirationTime represents the time-to-live for this specific request. The input parameter SLOModificationAcceptanceEPR represents the contact point of the party requesting the modification, to which the responding party will have to notify its decision of acceptance or rejection of the SLO modification proposal. To support such call-back mechanism, a new Port Type has been introduced. It has been named "SLOModificationAcceptance", must be implemented by both the parties and is in charge of receiving notifications for SLO modification proposals' acceptance or rejection. Two operations are exposed by this Port Type, respectively *Accept* and *Reject*. Here follows the signature of *Accept*:

wsag:Accept

```
 Input                                                Result

<wsag:ModificationAcceptInput>                       <wsag:ModificationAcceptResponse>
   <wsag:NoncriticalExtension/> *                       <xs:any> ... </xs:any> *
   <xs:any> ... </xs:any> *                          </wsag:ModificationAcceptResponce>
</wsag:ModificationAcceptInput>
```

From the time that the *Accept* operation is invoked, the modification of the SLO is effective, i.e., the Agreement has been modified and the Service Provider must adequate the QoS level to the new guarantee. As for the Reject operation, its signature is very similar to that of Accept, thus we omitted to report it.

For a better comprehension of the *ModifySLO* operation, in figure 2(a) we report a very simple sequence diagram referred to the scenario depicted in figure 1(a). The Service Provider SP_3 (or the AI on behalf of it) propose to SP_2 (or the AR on behalf of it) a modification of a SLO term of the Agreement Ag_{32}. This proposal triggers other modification offers in the chain. In this case all the

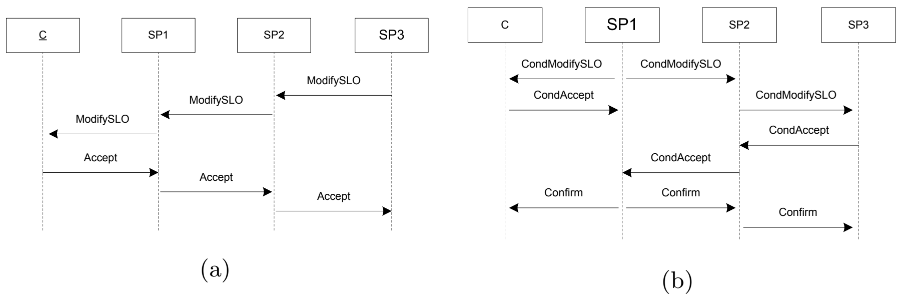

Fig. 2. Two-way (a) and three-way (b) interaction mechanisms

triggered offers are accepted, so the original proposal is accepted as well. If just one of the involved actors rejected a modification offer then the original proposal would not be accepted.

Let us now assume SP_1 be the party needing to modify the terms of one of its Agreements. This time the party originating the modification proposal has two Agreements with two different parties (two edges linking to two nodes). If for instance, according to the SP_1's needs, the modification of a given SLO in the Agreement Ag_{21} would cause the need of a modification of a SLO in the Agreement Ag_{1C}, SP_2 must check that both C and SP_2 agree with those modifications. Since the acceptance of one modification proposal is bound to the acceptance of the other, the ModifySLO operation is not appropriate to handle this situation. We need a *request-response-commit* interaction in order to first gather the willingness of both C and SP_2 to accept the proposed modifications, and then to confirm them (i.e., to commit the Agreements' modifications). To this end, to the "AgreementHandler" Port Type we have added the *ConditionalModifySLO* operation, whose input/result parameters' definition does not differ from those of the *ModifySLO* operation. The parties receiving a proposal of conditional modification, can respond by calling either a *ConditionalAccept* or a *ConditionalReject* (both of which have been added to the "SLOModificationAcceptance" Port Type) on the modification originator. The signature of *ConditionalAccept* is similar to that of *Accept*, but its meaning is different: the party invoking a *ConditionalAccept* communicates that it is just willing to accept the modification proposal, but a confirmation is still needed in order to consider the modification effective. When the modification proposal originator has gathered all the responses from the parties involved in the modification process, it can either confirm the commitment of the modifications, by calling the *Confirm* operation, or stop the entire modification process by invoking the *Cancel* operation (both the *Confirm* and the *Cancel* operations are accessible through the "SLOModificationAcceptance" Port Type). The signature of the *Confirm* operation is shown in the following, while in figure 2(b) we show how the three-way interaction mechanism works.

wsag:Confirm

```
Input

<wsag:ConfirmInput>
   <wsag:NoncriticalExtension/> *
   <xs:any> ... </xs:any> *
</wsag:ConfirmInput>
```

```
Result

<wsag:ConfirmResponse>
   <xs:any> ... </xs:any> *
</wsag:ConfirmResponce>
```

In a previous work of us [6] the reader may find the description of a real B2B scenario that can benefit from the proposal of dynamic SLAs' management presented in this work.

5 Related Work

To date in literature very few works have focused on the management of dynamic SLA in complex B2B scenarios. Most of the research in this area is focused on the elaboration of mechanisms and protocols for the negotiation of the QoS levels. Due to space requirements we cite only a few of them. In [4] the authors propose an extension of WS-Agreement allowing a run-time re-negotiation of the guarantees. Some modifications are proposed in the section wsag:GuaranteeTerm of the agreement schema, and a new section is added to define possible negotiations, to be agreed by the parties before that the Agreement offer is submitted. In this work there is not a real real-time re-negotiation because, after the agreement's acceptance, there is no interaction between the Service Provider and the Service Consumer. In the context of the VIOLA project [11] a simple negotiation protocol has been developed. A Three-phase-commit-protocol (3PCP) is proposed to overcome the limitations of WS-Agreement. This protocol basically aims at solving problems related to the creation of agreements on multiple sites. In [5] the WS-Agreement specification is used as a basis on which negotiations between two parties may be conducted. An agent-based infrastructure takes care of the Agreement offer made by the requesting party; from this offer many one-to-one negotiations are started in order to find the service that best matches the offer. Focusing on the WS-Agreement specification, in [10] the authors specify the guarantee terms of an Agreement as functions rather than just values. The aim of the work was to minimize the number of re-negotiations necessary to reach some consensus on values associated with agreement terms.

6 Conclusion and Future Work

In this work scenarios of B2B have been analyzed in order to identify the requirements that a framework for the managements of one-to-one SLAs must fulfill. Starting from the WS-Agreement specification we have proposed the integration of new functionality to improve the flexibility of the management of Agreements in service composition scenarios. In the future we are planning to further improve the flexibility of an Agreement, also accounting for the possibility of modifying at run-time the Agreement's expiration time. Another objective for our future work is the identification of the features that a service must exhibit in order for a re-negotiation of its QoS guarantees to be feasible at run-time, without requiring the suspension of the service provisioning. Finally, we are planning to implement the described protocol within a real framework for the creation, monitoring and modification of WS-Agreements, and to build upon it a test-bed for evaluating the effectiveness and the performance of the proposed protocol.

References

1. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification (WS-Agreement). OGF GRAAP-WG Recommendation (May 2007)
2. Angelov, S., Grefen, P.: B2B eContract Handling- A Survey of Projects, Papers and Standards. Technical Report TR-CTIT-01-21, University of Twente (2001)
3. Angelov, S., Grefen, P.: The Business Case for B2B E-Contracting. In: Proc. 6th International Conference on Electronic Commerce (ICEC 2004), , Delft, The Netherlands (October 2004)
4. Frankova, G., Malfatti, D., Aiello, M.: Semantics and Extensions of WS-Agreement. Journal of Software 1(1) (July 2006)
5. Joita, L., Rana, O.F., Chacín, P., Chao, I., Ardaiz, O.: Application Deployment using Catallactic Grid Middleware. In: 3rd International Workshop on Middleware for Grid Computing (MGC 2005), ACM/IFIP/USENIX International Middleware Conference, Grenoble (France) (November 2005)
6. Di Modica, G., Regalbuto, V., Tomarchio, O., Vita, L.: Dynamic re-negotiations of SLA in service composition scenarios. In: 33rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2007), Lübeck, Germany. IEEE Computer Society, Los Alamitos (2007)
7. Overton, C.: On the Theory and Practice of Internet SLAs. Journal of Computer Resource Measurement (106), 32–45 (2002)
8. Papazoglou, M.P.: Service-Oriented Computing: Concepts, Characteristics and Directions. In: 4th International Conference on Web Information Systems Engineering (WISE 2003), Rome, Italy, pp. 3–12. IEEE Computer Society, Los Alamitos (2003)
9. Papazoglou, M.P., van den Heuvel, W.-J.: Service Oriented Architectures: approaches, technologies and research issues. VLDB Journal 16(3), 389–415 (2007)
10. Sakellariou, R., Yarmolenko, V.: On the Flexibility of WS-Agreement for Job Submission. In: 3rd International Workshop on Middleware for Grid Computing (MGC 2005), ACM/IFIP/USENIX International Middleware Conference, Grenoble, France (November 2005)
11. Waeldrich, O., Ziegler, W.: A WS-Agreement based negotiation protocol. Technical report, Fraunhofer Institute SCAI (2006), `http://www.fz-juelich.de/zam/grid/VIOLA/`