Enabling re-negotiations of SLA by extending the WS-Agreement specification

Giuseppe Di Modica, Valerio Regalbuto, Orazio Tomarchio, Lorenzo Vita
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania
Viale A. Doria 6, 95125 Catania - Italy
E-mail: {Firstname.Lastname}@diit.unict.it

Abstract

Service Oriented Architecture (SOA) is a recently emerged enterprise architecture promising to perform business processes efficiently and effectively. In this work, we propose to enhance the flexibility of the WS-Agreement specification, a Web Service protocol to establish agreements on the QoS level to be guaranteed in the provision of a service. In fact, this protocol is generally aimed to be a "one-shot" interaction, i.e., the WS-Agreement is created based on a creation offer; the created agreement can not be modified and is effective until all activities pertaining to the agreement are finished or until one of the signing party decides to terminate it. Our main contribution provides for the integration of 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

The advent of information and communication technology has changed the nature of business-to-business interaction among organizations. This means allows organizations to increase both the effectiveness and efficiency of their interaction and consequently improves their business goals. parties. collaboration, the goods, the services or funds exchanged between parties, and details about the way this exchange takes place. 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 [?, ?]. This issue becomes more important in Service Oriented Architectures (SOA)[?, ?], which have recently appeared as an emerging paradigm for automated business integration.

In this context, 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 [?] is though a formal negotiated agreement between a Service Provider and its customer (the service requester). When a customer requests a service to a provider, an SLA is negotiated and a contract is drawn up. The contract involves parameters relating to the service to be provided, both functional and not functional. The high level of interoperability offered by the SOA enables scenarios of world-wide and cross-domains service composition. The end-to-end management of SLA in these scenarios is a tricky task. In a service-oriented scenario a service may be the result of the composition of several services deployed in as many administrative domains, each of which autonomously manages resources quantitatively and qualitatively diverse. In such a scenario, where the resource availability is highly dynamic, promising and guaranteeing specific QoS levels to the customer is a real challenge. The quality of the final service delivered to the customer is strongly affected by those of the services employed to compose it. If just one of the composing services violates at run-time its QoS guarantees, the global QoS delivered to the customer might get definitively compromised, to the detriment of the customer. There is a need for a flexible mechanism enabling the run-time renegotiation of the guarantees on the QoS once violations on such guarantees are expected to occur. This would avoid both the suspension of the service provisioning and the brutal termination of the agreement.

In order to describe the contract between the Service Provider, the customer and an eventual third party, several specifications, defining XML based languages, have been recently proposed. These languages were defined closely to a common language allowing a common understanding of the Service Provider obligations to perform a service according to agreed guarantees. Several languages have been defined (WSLA [?], WS-Agreement [?]), and all of them are a complement to the service description implemented by WSDL [?]. In general, such languages are used within a
framework allowing the management of Web Services and their compositions. Our attention has been focused on the WS-Agreement specification, that has been developed by the GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Scheduling and Resource Management (SRM) Area of the Global Grid Forum (GGF). It does not deal with negotiation mechanisms or protocols, rather it proposes to standardize the activities that follow the negotiation phase, i.e., those involving the signing of a contract between the parties.

In this work we have proposed modification to the current WS-Agreement specification in order to introduce a run-time support to the re-negotiation of the guarantees on the QoS levels as they were agreed in the contract by the signing parties. According to the current specification, if for some reason a Service Provider is not able to meet the Agreement’s objectives, the provision of the service that the Agreement applies to is brutally stopped, and the Agreement is terminated. However, it is plain that the termination of an Agreement is not convenient for both the Service Provider and the Service Consumer. On the one hand, the Service Provider may be interested in carrying out the service provisioning, trying to satisfy at best the Service Customer’s needs, avoiding as much as possible to incur penalties for violations of the promised guarantees; on the other, the Service Consumer that requests a given service may be willing to get the service provisioning successfully completed, even against an agreed lowering of the QoS level, rather than being rewarded for any guarantee’s violation and not being able to control the QoS degradation. A mechanism is needed that, whenever violations on guarantees are about to occur, supports the run-time (i.e., during the service provisioning) re-negotiation and modification of the guarantees. On a successful negotiation, the Agreement that applies to the service would be accordingly modified and the Service Provider would adjust the provided QoS level; if the negotiation failed, the Agreement would not be modified and would still regulate the service provisioning. In either case, the continuity of the service provisioning would be preserved, i.e., the application flow would not be suspended or stopped.

In our work, we do not add a negotiation layer, but introduce a mechanism that enables re-negotiation sessions between the parties and the subsequent modification of the original contract.

We refer to a very simple scenario in which the service requested by a Service Customer can be provided by a Service Provider, and no composition of services is required to satisfy the customer’s request. The considerations that follow can be easily extended to more complex scenarios involving two or more Service Providers in different administrative domains, in the logic of service composition, for the provisioning of a given service.

When a Service Provider fails to fulfill the objectives of an Agreement, it incurs penalties. Sometimes, because of the Service Provider’s chronic inability to meet the Agreement’s objectives, the provision of the service that the Agreement applies to is brutally stopped, and the Agreement is terminated. When a Service Provider decides to terminate an Agreement, not only it incurs penalties, but has also wasted resources that might have been better scheduled. Furthermore he loses credibility.

As for the Service Consumer, if the objectives of an Agreement are not met, it has wasted money for a service provision that eventually did not satisfy its expectations; even worse, when an Agreement is terminated he has to re-contract for the service, with no assurance to obtain the same guarantees as he was given in the just terminated Agreement.

It is plain that the termination of an Agreement is not convenient for both the Service Provider and the Service Consumer. On the one hand, in fact, the Service Provider may be interested in carrying out the service provisioning, trying to satisfy at best the Service Customer’s needs, avoiding as much as possible to incur penalties for violations of the promised guarantees; on the other, the Service Consumer that requests a given service may be willing to get the service provisioning successfully completed, even against an agreed lowering of the QoS level, rather than being rewarded for any guarantee’s violation and not being able to control the QoS degradation.

A mechanism is needed that, whenever violations on guarantees are about to occur, supports the run-time (i.e., during the service provisioning) re-negotiation and modification of the guarantees. On a successful negotiation, the Agreement that applies to the service would be accordingly modified and the Service Provider would adjust the provided QoS level; if the negotiation failed, the Agreement would not be modified and would still regulate the service provisioning. In either case, the continuity of the service provisioning would be preserved, i.e., the application flow would not be suspended or stopped.

Put in this way, only Service Providers seem to benefit from this proposal. In fact, Service Providers would have the opportunity to re-schedule resources assignments at run-time whenever needed. Let us now change the perspective. Service Consumers, during the service provisioning, might want to request the adjustment of the current QoS guarantees in order to accommodate new needs that have come up. Our proposal provides that the request of re-negotiation of the QoS levels can also be issued by the Service Consumer. In this case, after a successful negotiation, again, the original Agreement would be modified and the QoS levels would be adjusted "on-the-fly" (i.e., with no service suspension).

The objective of this work is to enhance the flexibility of the WS-Agreement approach by integrating new func-
tionality to the protocol that enable the parties of a WS-Agreement to re-negotiate and modify its QoS guarantees while the service is being provided.

Designing a protocol for the re-negotiation of the guarantees is out of the scope of this work. Out of the scope of this work is also establishing the requirements 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.

Then in Section 3 we introduce the WS-Agreement specification. Section 4 will describe our approach, and the proposed enhancement. Related work is presented in Section 5, while we conclude the paper in Section 6.

2. The WS-Agreement standard

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 and 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 (the Agreement Initiator, generally acting on behalf of a Service Consumer) creates an agreement document, possibly based on an agreement template, and proposes it to the other party (the Agreement Responder, generally acting on behalf of a Service Provider). 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 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.

In the agreement’s structure there are information on 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 typical WS-Agreement life cycle has four phases:

1. Exploration: a Service Provider provides templates describing possible agreement parameters;
2. Creation: Consumer fills in parameters, and makes an offer;
3. Operation: Agreement state is available for monitoring as a ResourceProperty;
4. Termination: Agreement is destroyed explicitly or via soft state (termination time).

3. Enabling WS-Agreement re-negotiation

In this section we give a detailed description of the new features that have been added to the WS-Agreement specification. From now on, with the term Agreement we will refer to the WS-Agreement document signed by two parties and regulating the provisioning of a service.

The main modification being introduced to the current specification concerns:

- Integrations to the XML schema of the Agreement;
- Enhancement of the protocol with new interactions for the run-time modification of the Agreement.

At design time the following questions were posed: Which sections of an Agreement can be modified? When can a party request an Agreement modification? How often can an Agreement undergo modifications? How can a party request a modification to an Agreement?

The purpose of our work is to give flexibility to the management of Agreements, while trying to keep, at the same time, the nature of each specific Agreement from being perverted. It is necessary to devise appropriate criteria for the modification of an Agreement, in a way that the modification to be applied is such as not to justify the cancellation of the existing Agreement and the creation of a new one. 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 Objective(SLO) in the Guarantee Terms section. We decided not to modify the current definition of the ServiceLevelObjective term in the Agreement’s XML schema, rather we have introduced a brand new element, named ModifiableServiceLevelObjective, representing a particular type of SLO that can be modified at run-time. It has been defined as follows:
We can notice the presence of three items:

- **Objective**, used to define the new QoS level that the provider will have to guarantee if the modification proposal was accepted;
- **MaxCount**, used to specify the number of times that a given SLO can be modified during the Agreement’s life cycle;
- **TimeInterval**, representing the time lapse during which requesting the modification of the given SLO is allowed.

On its turn, the **TimeInterval** item is defined as a complex type composed by two integer numbers, to be interpreted as a percentage of the Agreement’s life time; they represent respectively the time before which and after which requesting the modification of that SLO is not allowed. We remind that the life time of an Agreement starts when the Agreement’s state get to the **Observed** state (i.e., when the AR accepts the Agreement offer) and ends up with its expiration (specified by the parameter **expirationTime** in the Agreement document).

It is worth noticing that an Agreement’s XML document conforming to the current standard’s Agreement XML schema is conform to the new schema as well. Our purpose at protocol’s design time was, in fact, to maintain the backward compatibility with the original standard. The rest of modifications that have been introduced do not alter the original protocol’s functionality, but rather integrate new ones.

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 request its modification.

The existing protocol provides only one single interaction by which the Agreement Initiator (AI) makes an Agreement offer and the Agreement Responder (AR) is called to accept or reject it. We introduce a new form of interaction that enables both the parties to request modification 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 acceptance of the proposal from the other party.

We stress that after that an Agreement’s modification offer has been issued, the service provisioning is not interrupted, nor the Agreement’s monitoring is 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.

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 not modified the existing operations for the Agreement’s creation (namely "CreateAgreement" and "CreatePendingAgreement"), rather we have introduced new operations for the creation of **Modifiable Agreements**. Such operations differ from the existing ones for the fact the the two parties exchange each other’s EPRs that are specific for receiving Agreement’s modification proposals.

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

```
<wsag:CreateModifiableAgreement
  InitiatorAgreementEPR
  InitiatorAgreementModificationRequestEPR
  AgreementOffer
  NoncriticalExtension
  any

Result
  CreatedAgreementEPR
  ResponderAgreementModificationRequestEPR
  any
```

We can notice the presence of three items:
Specularly, we added the "CreateModifiablePendingAgreement" operation to the PendingAgreementFactory Port Type:

\[ \text{wsag:CreateModifiablePendingAgreement} \]

\text{Input}
\[ \langle \text{wsag:CreatePendingModifiableAgreementInput} \rangle \]
\text{Result}
\[ \langle \text{wsag:CreatePendingModifiableAgreementResponse} \rangle \]

Let us assume that the AI has made his offer of Agreement’s creation by invoking one of the two operations described above, and let us also assume that the offer has been accepted. From now on, who of the two parties desired to make a proposal of Agreement’s modification, can do it through 1) a "synchronous" modification request, that implies an immediate answer of the other party, or through 2) an "asynchronous" modification request, that instead allows the responding party to delay the answer. After receiving a "synchronous" modification request, the responding party decides whether to accept or reject the proposal and immediately communicates its decision to the requesting party. After issuing an "asynchronous" modification request, the requesting party will have to wait for the responding party’s response; the responding party can take its time to decide whether to accept or reject the proposal, and will communicate it to the requesting party by means of a call back mechanism.

No matter whether the request was "synchronous" or "asynchronous", after the proposal’s acceptance the Agreement is accordingly modified and the Service Provider immediately adjusts the QoS to the new agreed levels. Therefore, in order for the protocol to support the modification request, the new "AgreementHandler" Port Type has been introduced. Its operations, that must be implemented by both the AI and the AR, are detailed as follows:

\[ \text{wsag:ModifyAgreement} \]

\text{Input}
\[ \langle \text{wsag:ModifyAgreementInput} \rangle \]
\text{Result}
\[ \langle \text{wsag:AcceptModificationResponse} \rangle \]

The meaning of the ModifyAgreement operation’s parameters are clear enough. The only input parameter, AgreementOffer, represents the offer of an Agreement where modifications have been introduced to any of the modifiable SLOs. The DeferredModifyAgreement operation has
the mandatory input parameter AgreementModificationAcceptanceEPR, representing the point of contact of the party requesting the modification, to which the responding party will have to notify its decision of acceptance or rejection of the modification proposal. As a consequence of this, a new Port Type has been introduced. It has been named "AgreementModificationAcceptance", must be implemented by both the parties and is in charge of receiving notifications for modification proposals’ acceptance or rejection. The definitions of the exposed operations are the following:

**wsag:Accept**

**Input**

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

**Result**

```xml
<wsag:ModificationAcceptResponse>
  <xs:any> ... </xs:any> *
</wsag:ModificationAcceptResponse>
```

**wsag:Reject**

**Input**

```xml
<wsag:ModificationObservedRejectInput>
  <wsag:NoncriticalExtension/> *
  <xs:any> ... </xs:any> *
</wsag:ModificationObservedRejectInput>
```

**Result**

```xml
<wsag:ModificationObservedRejectResponse>
  <xs:any> ... </xs:any> *
</wsag:ModificationObservedRejectResponse>
```

Because of such integrations, the Agreement State exposed to the AI will observe a new model, depicted in the Figure ??.

When a "synchronous" modification request is issued, after its acceptance or rejection the Agreements remains in the state *Observed* (in the figure, a self transition). As said before, in fact, the Agreement’s modification process does not imply a suspension of the service provisioning, thus the monitoring of the Agreement’s guarantees still goes on (state "Observed"), no matter whether the modification request is accepted or not. Of course, in ther case that the modification proposal is accepted, the guarantees being monitored are the new agreed ones.

When an "asynchronous" modification request is issued, the state of the Agreement changes from *Observed* to *ObservedAndModifying*, meaning that the modification request has been received and is being considered for acceptance, but the acceptance/rejection decision is deferred. From this state, the Agreement state may fall back to the *Observed* state, meaning that the decision has been notified, be it an notification of acceptance or a notification of rejection.

From the state *ObservedAndModifying* the Agreement may also change to the *ObservedAndModifyingAndTerminating* state if the AI decides to terminate the Agreement. From that state, finally, the Agreement state may change to *Terminated*, whose meaning is well known.

For a better comprehension of the modification that have been made to the current WS-Agreement specification, we present some use cases. In Figure ?? the AI makes an offer of modifiable Agreement, that is accepted by the AR. Afterwards, the AI issues an Agreement’s synchronous modification request, that is accepted by the AR.

![Figure 2. Agreement Creation and synchronous modification - initiator side](attachment:image.png)

In Figure ?? the AI makes an offer of modifiable Agreement that is accepted by the AR. Afterwards, the AI issues...
an Agreement’s asynchronous modification request, whose notification of acceptance is delayed by the AR.

Figure 3. Agreement Creation and asynchronous modification - initiator side

In Figure ?? the AI makes an offer of modifiable Agreement that is accepted by the AR. Afterwards, the AR issues an Agreement’s asynchronous modification request, whose notification of acceptance is delayed by the AI.

4. Related works

In [?] 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. If the provider violates the agreement, it will try to modify the agreement’s guarantee terms using the information in the negotiation section of the agreement. 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 [?] a simple negotiation protocol has been developed. A Three-phase-commit-protocol (3PCP) is proposed to overcome the limitations of WS-Agreement. It is based on the creation of different types of agreements within a negotiation process, namely a Declaration of Intention Agreement, a Preparation Agreement, and a Commitment Agreement. All of these agreements are normal WS-Agreements, following a certain naming convention. This protocol basically aims at solving problems related to the creation of agreements on multiple sites.

In [?] a negotiation approach for mobile agents is proposed. Agents acquire time-limited resource contracts through negotiation with one or more mediators instead of individual hosting systems. While the WS-Agreement interaction protocol allows only for a single ”request - accept” interaction, in this work a new model is proposed. It is introduced an accept/reject interaction sequence that allows the requesting party to explicitly accept or reject an offer created by the providing party. This allows agents to negotiate with multiple domain coordinators simultaneously and to accept the best offer from the set of the received offers.

CREMONA [?] is a WS-Agreement framework implemented by IBM. Cremona (Creation and Monitoring of Agreements) proposes an architecture for the WS-Agreement-implementing middleware. In addition, the Cremona Java Library implements the WS-Agreement interfaces, provides management functionality for agreement templates and instances, and defines abstractions of service-providing systems that can be implemented in a domain-specific environment.
5. Conclusion and future work

In this work integrations to the WS-Agreement specification have been proposed in order to support the modification of the guarantees of Agreements during the service provisioning. New functionality have been added to the protocol with the introduction of a new type of Agreement (the "modifiable" Agreement) and new Port Types and Operations for handling the Agreement’s modifications.

In the future we are planning to further improve the flexibility of an Agreement by taking into account the possibility of requesting "planned" modifications of agreements, i.e. modifications that become effective not just after their acceptance but later at a given moment in time; we shall also account for the possibility of modifying at run-time the Agreement’s expiration time. Furthermore, we are also planning to implement the described protocol within a framework for the creation, monitoring and modification of WS-Agreements.

Acknowledgment

This work makes use of results produced by the PI2S2 Project managed by the Consorzio COMETA, a project co-funded by the Italian Ministry of University and Research (MIUR) within the Piano Operativo Nazionale "Ricerca Scientifica, Sviluppo Tecnologico, Alta Formazione” (PON 2000-2006).

References