What's in an Agreement? A Formal Analysis and an extension of WS-Agreement

Marco Aiello,[1] Ganna Frankova[1] and Daniela Malfatti[2]

[1] Dept. of Information and Communication Technologies
University of Trento
Via Sommarive, 14
38100 Trento
Italy
email: {marco.aiello,ganna.frankova}@unitn.it
[2] Corso di Laurea in Informatica
University of Trento
Via Sommarive, 14
38100 Trento
Italy
email: daniela.malfatti@studenti.unitn.it

Abstract. Non-functional properties of services and service compositions are of paramount importance for the success of web services. The negotiation of non-functional properties between web service provider and consumer can be agreed on a priori, by specifying an agreement. WS-Agreement is a recently proposed and emerging protocol for the specification of agreements in the context of web services. Though, WS-Agreement only specifies the XML syntax and the intended meaning of each tag, which naturally leads to posing the question of "What's in an Agreement?" We answer this question by providing a formal definition of an agreement and analyzing the possible evolution of agreements and their terms. From our analysis it turns out that agreements can be made more robust and longer-lived by a simple extension, which we present.

WS-FM topic: Performance Evaluation and Quality of Service of WS

1 Introduction

Web Services (WS) are a set of technologies that allow the construction of massively distributed and loosely coupled applications. The main characteristics of a distributed system based on WS resides in the asynchronicity of the message exchange, the absence of a unique owner of the application, and the openness of the XML based protocols used for the interaction. The present attention to web services is due to the potentials of the technology. WS are proving to enable

the creation of virtual enterprises, to create the possibility for inter-company and intracompany interoperation, to allow the creation of open electronic marketplaces. Not only, web services can be used also to allow for spontaneous networking and interoperation between different appliances in the context of ambient intelligence [2].

One of the most thought provoking issues in web services is that of automatic composing individual operations of services in order to build complex added-value services. The research on composition is well under way, but most of the focus is on functional properties of the composition, that is, how does one automatically compose? How does one enrich the services with semantic self-describing information? How does one discover the available services to use for the composition? If, on the one hand, this is crucial, on the other one, it is not enough. Non-functional properties of the composition are also of paramount importance in defining the usability and success of a composed service. Think for instance of desiring a service that performs a biological computation composing the services offered by a number of web service enabled machines. If the user knows that the composition is correct with respect to his goal, he will be satisfied with the answer he receives, but if the answer takes 3 years to be delivered to the user, the correctness is of little use. Therefore, the quality of a composed service is very important when interacting with an asynchronous system built out of independent components.

With the term Quality of Service (QoS) we refer to the non-functional properties of an individual service, or a composition of services. The term is widely used in the field of networking. Usually it refers to the properties of availability and performance. In the field of web services, the term has a wider meaning. Any non-functional property which affects the definition and execution of a web service falls into the category of QoS, most notably, accessibility, integrity, reliability, regulatory, and security [15]. Dealing with QoS requires the study of a number of problems. One, the design of quality aware systems. Two, the provision of quality of service information at the level of the individual service. Three, ensuring that a promised quality of service is actually provided during execution. In [1], we addressed the first issue by using the Tropos design methodology, and the second one by resorting to WS-Policy to describe QoS properties. In this paper, we consider the second and third issues; in particular, we show how to provide a framework to negotiate the provision of a service according to a predefined QoS, and how to handle changes during the interactions of web services.

WS-Agreement is a XML based language and protocol designed for advertising the capabilities of providers and creating agreements based on initial offers, and for monitoring agreement compliance at runtime. The motivations for the design of WS-Agreement stem out of QoS concerns, especially in the context of load balancing heavy loads on a grid of web service enabled hosts [9]. However, the definition of the protocol is totally general and allows for the negotiation of QoS in any web service enabled distributed system. If, on the one hand, the proposal of WS-Agreement is a step forward for obtaining web service based

systems with QoS guarantees, on the other hand, the protocol proposal is preliminary. The current specification [3] defines a XML syntax for the language and protocol and it gives a vague textual overview of the intended semantics, without defining a set of formal mathematical rules. Furthermore, a reference architecture is proposed to show how WS-Agreement are to be handled, [13]. Nevertheless, a formal analysis of what an agreement is is still missing.

In this paper, we address the question *What's in an Agreement?* In particular, we provide a formal analysis of WS-Agreement by resorting to finite state automata, we provide a set of formal rules that tie together agreement terms and the life-cycle of an agreement. From the analysis, some shortcomings of the protocol become evident. Most notably, breaking one single term of the agreement results in breaking the agreement, while a more graceful degradation is desirable. Therefore, we propose an extension of the protocol for which we provide appropriate semantics, that allows the renegotiation of running agreements by tolerating the break of a term.

Web service QoS issues are gaining attention and have been addressed in a number of recent works. Some approaches are based on the extension of the Web Service Description Language (WSDL) to define not only functional, but also non-functional properties of the service, e.g., [10]. The main idea of the approach is simple: provide syntax to define terms which refer to non-functional properties of operations. The problem with this kind of approach is that the QoS definition is tied to the individual operation, rather than with the service as a whole; furthermore, there is no run-time support. Once a quality is defined, it can not be changed at execution time.

In [18], the authors propose to define WS QoS by using XML schemata that both service consumers and service providers apply to define the agreed QoS parameters. The approach allows for the dynamic selection of WS depending on various QoS requirements. On the negative side, the life-cycle of agreements is not taken into account, and it is not possible to define an expiration for a negotiation. The feasibility of using constraint programming to improve the automation of web services procurement is shown in [16]. A semantic web approach, in which services are searched on the basis of the quality of semantically tagged service attributes is presented in [17]. A predictive QoS model for workflows involving QoS properties is proposed in [5]. In [8], the authors propose a model and architecture to let the consumer rate the qualities of a service. In addition, the industry has proposed a number of standards to address the issue of QoS: IBM Web Service Level Agreement (WSLA) and HP's Web Service Management Language (WSML) are examples of languages used to describe quality metrics of services, [12]. A recent proposal is the specification of a new WS protocol, called Web Services Agreement Specification [3]. In [6], it is presented the Agreement-Based Open Grid Service Management (OGSI-A) model. Its aim is to integrate Grid technologies with Web Service mechanisms and to manage dinamically negotiable applications and services, using WS-Agreement. The WS-Agreement protocol proposal is supported by the definition of a managing architecture: CREMONA–An Architecture and Library for Creating and

Monitoring of WS-Agreement [13]. The Web Services Agreement Specification defines the interaction between a service provider and a consumer, and a protocol for creating an agreement using agreement templates. The above approaches show that frameworks for QoS definition and management are essential to the success of the web service technology, but there are a number of shortcomings that still need to be addressed. First, no one has worked out a formal definition of what the semantics of a QoS negotiation should be. Second, the frameworks should be more flexible at execution time because actual qualities of services may change over time during execution.

The remainder of the paper is organized as follows. In Section 2, we present the WS-Agreement protocol defined in [3] and provide an example. In Section 3, we propose a formal definition of an agreement and of its life-cycle. Section 4 is devoted to the presentation of an extension of WS-Agreement with the goal of improving the duration and tolerance of an agreement in execution. It also contains an example using the extension. Some concluding remarks and hints for future work are presented in Section 5.

2 WS-Agreement

In order to be successful, web service providers have to offer and meet guarantees related to the services they develop. Taking into account that a guarantee depends on actual resource usage, the service consumer must request state-dependent guarantees from the service provider. Additionally, the guarantees on service quality must be monitored and service consumers must be notified in case of failure of meeting the guarantees. An agreement between a service consumer and a service provider specifies the associated guarantees.

The agreement can be formally specified using WS-Agreement Specification [3]. The WS-Agreement specification was developed by the GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Scheduling and Resource Management (SRM) Area of the GGF to standardize means for agreements creation processes.

A WS-Agreement is a XML-based document containing descriptions of the functional and non-functional properties of a service oriented application. It consists of two main components that are the agreement Context and the Agreement Terms. Figure 1 illustrates the main structure of WS-Agreement as defined in [3]. The agreement Context includes the description of the parties involved in the agreement process, and various metadata about the agreement. One of the most relevant components is the duration of the agreement, that is, the time at which the agreement is no longer valid.

Functional and non-functional requirements are specified in the Terms section, that is divided into *Service Description Terms* (SDTs) and *Guarantee Terms*. The first provides information to define the services functionalities that will be delivered under the agreement. An agreement may contain any number of SDTs. An agreement can refer to multiple components of functionalities

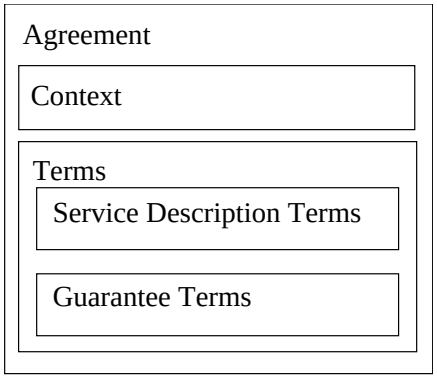

Fig. 1. WS-Agreement structure

within one service, and can refer to several services. Guarantee Terms define an assurance on service quality associated with the service described by the Service Description Terms. An agreement may contain zero or more Guarantee Terms. The main parts of a Guarantee Term are:

/GuaranteeTerm/ServiceScope is the list of service names a guarantee applies to;

/GuaranteeTerm/QualifyingCondition is an optional condition that expresses a precondition under which a guarantee holds;

/GuaranteeTerm/ServiceLevelObjective is a condition that must be met to satisfy the guarantee; and

/GuaranteeTerm/BusinessValueList is a list of business value elements associated with a service level objective.

[7] specifies a definition for guarantee terms in WS-Agreement and provides mechanisms for defining guarantees. An agreement creation process starts when an agreement initiator sends an agreement template to the consumer. The structure of the template is the same as that of an agreement, but an agreement template may also contain a Creation Constraint section, i.e., a section with constraints on possible values of terms for creating an agreement. [4] enables customizations of terms and attributes for the agreement creation. After the consumer fills in the template, he sends it to the initiator as an offer. The initiator decides to accept or reject the offer depending on the availability of resource, the service cost, and other requirements monitored by the service provider. The reply of the initiator is a confirmation or a rejection.

An agreement life-cycle includes the creation, termination and monitoring of agreement states. Figure 2 shows a representation of the life-cycle. When an agreement is created, it does not imply that it is monitored. It remains in an `not_observed` state until a service starts its execution. The semantics of the states is as follows:

Fig. 2. The life-cycle of a WS-Agreement

- `not_observed`: the agreement is created and is in execution, but no service involved in the agreement is running; and
- `observed`: at least one service of the agreement is running;
- `finished`: the agreement has terminated either successfully or not.

We illustrate a simple example in order to explain the WS-Agreement specification and its features. A possible application scenario is the request of executing fidelity-card operations[3]. Two parties act in this scenario: a shopping center

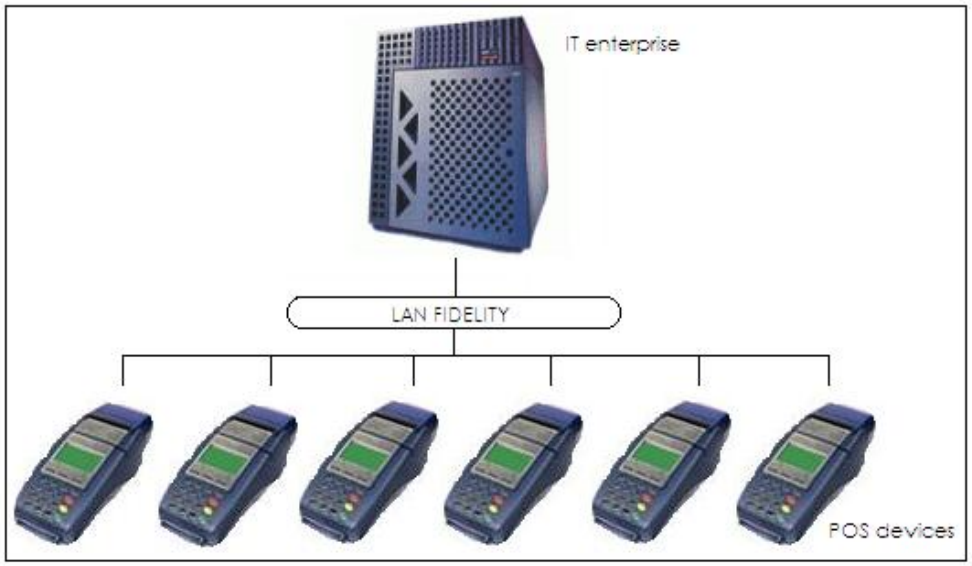

Fig. 3. Fidelity-card scenario

with several Point of Sale (POS) devices, and an IT enterprise with a server farm managing the operations and fidelity database. This scenario is depicted in Figure 3. A store of a shopping center sends a request to the server farm, asking the execution of a typical fidelity-card operation, e.g., fidelity-points, car

[3] Example based on a project of the IT company DeltaDator `http://www.deltadator.it`

park payment, etc. The server farm executes the request and sends a reply to the store, eventually interacting with the bank circuit for payments.

In this scenario, the agreement defines a list of operations to be executed, and an optional set of guarantees for providing a quality of service, i.e., number of operation request for a minute, number of processing operation for a minute, service cost, etc. Let us consider, for instance, the QoS related to the speed of execution: it is necessary to process many operations in a short time. Therefore, provider and consumer assume that the number of requests for minute should be less that 5, the number of processing operations for minute should be more than 12, and the service cost for a single operation should be less that 1 USD.

According to the established agreement between the server farm, the shopping centers, and the stores, fidelity-card operations will be executed taking into account the services required and the guarantees defined in the agreement.

3 What's in an Agreement?

The WS-Agreement specification provides a XML syntax and a textual explanation of what the various XML tags mean and how they should be interpreted. Thank to the syntax, it is possible to prepare machine readable agreements, but a formal notion of agreement is missing. In this section, we formalize the notion of agreement, by defining its main components.

Definition 1 (Term). *A* term t *is a couple* (s, g) *with* $s \in S$ *and* $g \in G$*, where* S *is a set of* n *services and* G *is a set of* m *guarantees.* $T \subseteq S \times G$ *is the set of the terms* t*.*

In words, a term involves the relationship between a service s and a guarantee g, not simply a specific tag of the agreement structure. If the service s appears in the list of services, which the guarantee g is applied to, it means that the couple (s, g) is a term. The number of terms varies between 0 and $n \cdot m$, where 0 means that there is no association between services and guarantees, and $n \cdot m$ indicates the case where each guarantee is associated with all services.

Definition 2 (Agreement). *An* agreement A *is a tuple* $\langle S, G, T \rangle$*, where* S *is a set of* n *services,* G *is a set of* m *guarantees, and* T *is the set of the terms* t*.*

In the following analysis, it is more convenient to consider the agreement as a set of *Terms* rather than a set of related services and guarantees. From the definition of WS-Agreement, we say that an agreement can be in one and only one of three states: `not_observed`, `observed` and `finished`.

Definition 3 (External state). *The* external state A_{es} *of an agreement* A *is an element of the set* {`not_observed`, `observed` *or* `finished`}*.*

We call the above state external, as it is the observable one. We also define an internal state of an agreement, which captures the state of the individual terms.

Definition 4 (Internal state). *The* internal state A_{is} *of an agreement* A *is a sequence of terms' states* $ts_1, \ldots, ts_p$ *of maximum size* $n \cdot m$*, where* $ts_i = (ss_j, gs_k)$ *represents the state of* g_k *guarantee with respect to the state of the* s_j *service. Service and guarantee states range over the following sets, respectively:*

- $ss_j \in$ {`not_ready`, `ready`, `running`, `finished`}, *and*
- $gs_k \in$ {`not_determined`, `fulfilled`, `violated`}.

From the definition of *Term*, we see that services and guarantees are related and we can define the internal state of an agreement, but it is necessary to distinguish between terms that have the same service and terms that have the same guarantee.

Proceeding in our goal of answering the question of what is in an agreement, we define the relationship between the internal and external state of an agreement A. First, we note that not all state combinations make sense. For instance, it has no meaning to say that a guarantee is `violated`, when a service is in a `not_ready` state. The only admissible combinations are the following ones.

(1) (`not_ready`, `not_determined`) (2) (`ready`, `not_determined`)
(3) (`running`, `fulfilled`) (4) (`running`, `violated`)
(5) (`finished`, `fulfilled`) (6) (`finished`, `violated`)

In theory, there are 63 possible combinations of states in which terms can be. That is,

$$\sum_{i=1}^{6} \binom{6}{i}$$

all terms could be in state (1), or in state (2),... or in state (6); there could be terms in states (1) and (2), (1) and (3), and so on. But again, considering the definition of WS-Agreement in [3], one concludes that not all 63 combinations make sense. Furthermore, it is possible to extract the possible evolutions of these aggregated internal states.

	terms are in state	state of the agreement	transitions
(A)	(1)	`not_observed`	(B)
(B)	(1)(2)	`not_observed`	(C) (E)
(C)	(1)(2)(3)	`observed`	(D)(E)(F)(G)
(D)	(1)(2)(3)(5)	`observed`	(F)(G)
(E)	(1)(2)(4)	`observed`	(F)(H)
(F)	(1)(2)(3)(4)(5)	`observed`	(H)
(G)	(5)	`finished`	
(H)	(1)(2)(3)(4)(5)(6)	`finished`	

Fig. 4. Transition table for the relation between internal and external states

When an agreement is created its external state is `not_observed`, while all services are `not_ready` and all guarantees are `not_determined`, i.e., state (1). In

the next stage some services will be `ready` while others will still be `not_ready`, i.e., there will be terms in state (1) and (2). In this case, the external state is also `not_observed`. Proceeding in this analysis, one can conclude that there are 8 situations in which terms can be. We summarize these in the table in Figure 4. In the table, we also present the relation between the internal states and the external states, and the set of transitions to go from one set of states to another. The latter transitions are best viewed as an automaton.

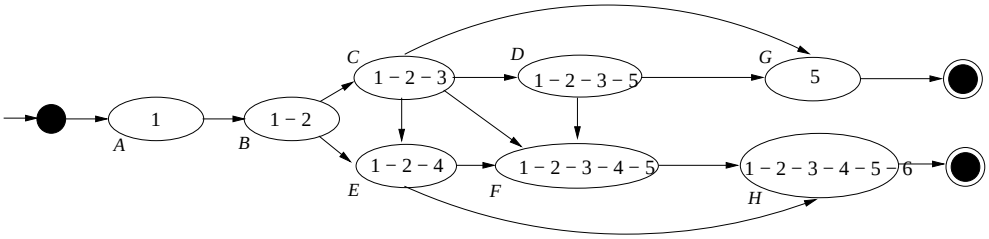

Fig. 5. Automaton representation of the table in Figure 4

Referring to Figure 5, at the beginning all the terms are tied to services which are not running (A). At some point, some services will be ready to start (B). Services which are ready will start execution. This may result in an immediate violation of a term (E), or in executions fulfilling the term (C). If the latter is the case, more and more services will execute. This may result in violations, which bring us to states (E) or (F), or in no violation. Some services may successfully terminate execution, case (D). If all services terminate with no violation, we end successfully in state (G). If any service has a violation at any time, we end in state (E) or (F) and from there, unavoidably, in state (H), which is a failure state.

4 Extension of WS-Agreement

From the semantics and formal analysis just presented, we note that if the agreement arrives into state (E) or (F) there is a failure, and consequently an agreement abortion. To reduce the occurrence of these failures, following the initial proposal in [14], we propose an extension of WS-Agreement specification. Next, we provide syntax and semantics for the extended version of agreements.

4.1 Goals and requirements

The WS-Agreement specification does not contemplate the possibility of changing an agreement at runtime. If a guarantee is not fulfilled because of resource overload or faults in assigning availability to consumers, the agreement must

terminate. For maintaining the service and related supplied guarantees, it is necessary to create another agreement and negotiate the QoS again. This approach wastes resources and computational time, and increases network traffic.

When a service is running, it is much simpler to know the following features:

- if a resource is available or not,
- if a provider can satisfy more requests or not, and
- if more guarantees can be satisfied or not.

At agreement's creation time, indeed, it is more complex to assume the QoS that can be provided to the consumers. For this reason, it is important to change an agreement already created and running: this allows robust and efficient services in terms of satisfying guarantees and QoS.

To allow a renegotiation at runtime, it is necessary to add some elements to an agreement that specify how the agreement can be modified according to execution circumstances. That is, to add a structure to the agreement protocol syntax for defining renegotiation possibilities. When consumer and provider come to an agreement about the services, they also can define some terms of renegotiation that are referred to specific services and guarantees.

By adding appropriate terms, both positive and negative renegotiations can be specified and used to modify a guarantee (both if it has been violated, but also if it is fulfilled and a party wants to change it). It is possible to reduce the value of a currently violated guarantee by increasing another one. It is also possible to increase a currently satisfied guarantee if resources and agreements between the two parties allow it.

The goal of negotiation terms is to have the chance to modify the agreement applying the negotiation terms rather than respecting the original agreement. Applying the negotiation terms means that the services included in the agreement will be performed according to the new guarantees. This feature is allowed only if there is a good monitoring system that interacts with the agreement and service architecture, e.g., as in [13].

4.2 Life-cycle and semantics for the extended agreement

With the extension of the protocol, the agreement life-cycle changes according to the new features. In particular, there are more chances to define the agreement services and guarantees: at the beginning, using the template and filling in the agreement according to agreement Creation Constraints and, during the execution, applying a negotiation term.

The definition of an agreement does not change with respect to Definition 2, the difference lies in the fact that the set of terms T is now extended with special *renegotiation terms*. These terms are defined as in Definition 1, but have a different role, i.e., they specify new conditions that enable modification of guarantees at run-time.

To account for the new type of terms, we need to extend the definition of external and internal state of an agreement. The external states of an extended

agreement are enriched by the `checked` state, the `revisited` state and the `denied` state. We say that an agreement can be in one of six states. `not_observed`, `observed` and `finished` have the same meaning as in WS-Agreement, Figure 2. An agreement is in state `checked` when the monitoring system is checking its services and guarantees. From the `checked` state the agreement can go the three different states: back to `observed` if the agreement is fulfilled, to `revisited` or to `denied`, if the agreement is violated. In the first case, the guarantee can be changed thanks to the negotiation terms and the agreement is fulfilled again; in the second case, it is not possible to renegotiate and the agreement must terminate.

Definition 5 (Extended External state). *The* extended agreement external state A_{xes} *of an agreement A is an element of the set* {`not_observed`, `observed`, `checked`, `revisited`, `denied` *or* `finished`}.

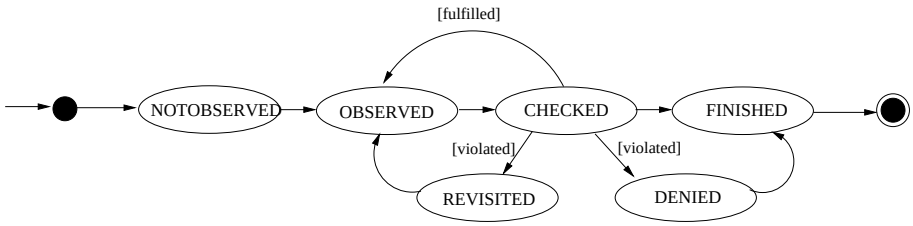

Fig. 6. The life-cylce of the WS-Agreement extension

The transitions between states are illustrated by the automaton in Figure 6, which is an extension of the one presented in Figure 2. The automaton represents the new evolution of an agreement where a guarantee can be modified during the processing of a service. When a guarantee is violated, it can be modified if there is a negotiation term suitable for it, otherwise the agreement must terminate. Also, when a guarantee is fulfilled, it is possible to change the current agreement configuration, applying a negotiation term that increases the QoS.

The internal state definition for the extended agreement is similar to the internal state definition stated before, but a new state for the services is added. It is `stopped` and is needed to define a state of a service where its associated guarantee is violated and the service must terminate or the guarantee can be revisited. It is an intermediate state.

Definition 6 (Extended Internal state). *The* extended internal state A_{xis} *of an agreement A is a sequence of terms' states* $ts_1, \ldots, ts_p$ *of maximum size* $n \cdot m$*, where* $ts_i = (ss_j, gs_k)$ *represents the state of* g_k *guarantee with respect to the* s_j *service. Service and guarantee states range over the following sets, respectively:*

– $ss_j \in$ {not_ready, ready, running, stopped, finished}, *and*
– $gs_k \in$ {not_determined, fulfilled, violated, non_recoverably_violated}.

As for Definition 4, one notes that not all the state combinations make sense. The only possible ones are the combinations itemized in Section 3 plus the following three:

(7) (stopped, fulfilled)
(8) (stopped, violated)
(9) (stopped, non_recoverably_violated)

The state combinations (7), (8) and (9) determine the states when a service is stopped because a guarantee is violated or is being modified. In state (7) a guarantee is fulfilled and we try to improve it applying a positive negotiation term. In (8) and (9) a guarantee is currently violated: in (8) the service is stopped and the guarantee is violated but it is possible to apply a negotiation term and to preserve the agreement again. In (9), instead, the guarantee is irrecoverably violated and the agreement must terminate, there are not any suitable negotiation terms. The relation between internal and external states of an extended

	terms are in state	state of the agreement	transitions
(A)	(1)	not_observed	(B)
(B)	(1)(2)	not_observed	(C) (E)
(C)	(1)(2)(3)	checked	(D)(E)(F)(G)
(D)	(1)(2)(3)(5)	checked	(F)(G)(I)
(E)	(1)(2)(4)	checked	(F)(H)
(F)	(1)(2)(3)(4)(5)	checked	(H)(J)(K)
(G)	(5)	finished	
(H)	(1)(2)(3)(4)(5)(6)	finished	
(I)	(1)(2)(3)(4)(5)(7)	observed	(D)
(J)	(1)(2)(3)(4)(5)(8)	revisited	(D)
(K)	(1)(2)(3)(4)(5)(9)	denied	(F)

Fig. 7. Extension of the transition table for the relation between internal and external states

agreement is an extension of the one presented in the table in Figure 4, and it is presented in Figure 7. The table respects the original agreement evolution and presents some new transitions. Referring to the automaton in Figure 8, when an agreement is checked it is possible to change its terms if a negotiation term allows it. From the state (D), it is possible to improve the current guarantee and apply a positive negotiation term through (I) if a guarantee is fulfilled. If a guarantee is violated, instead, we are in (J) if the guarantee can be recovered or in (K), if the guarantee can not be changed and the agreement is unavoidably violated.

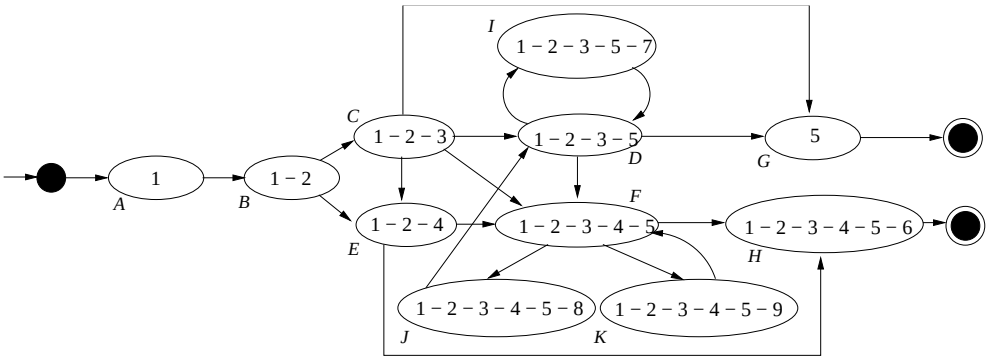

Fig. 8. Automaton representation of the table in Figure 7

4.3 Extended Agreement structure

The proposed extension is reflected in a new component of a WS-Agreement, as shown in Figure 9, and in an appropriate XML syntax.

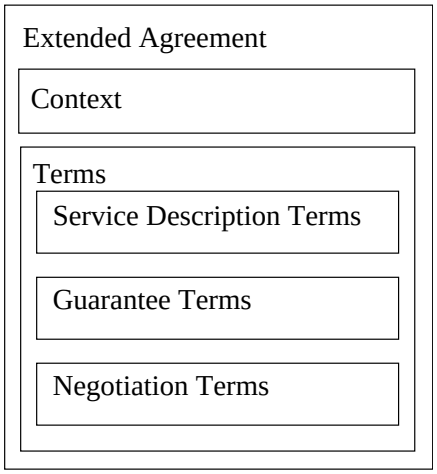

Fig. 9. Extended WS-Agreement structure

The section *Terms* is extended with a new subsection called *Negotiation Terms* that makes references to the services defined in *Service Description Terms* and to the guarantees of *Guarantee Terms*.

The following XML code illustrates the *Negotiation Terms* structure.

```
1     <NegotiationTerm wsag:Name="xs:NCName"
```

```
            Counter="xsd:integer"
            Monitored="xs:boolean">
        <GuaranteeScope>
            ...
        </GuaranteeScope> *
        <NegotiationRange>
            <GuaranteeName>...</GuaranteeName>
            <Minimum>...</Minimum>
            <Maximum>...</Maximum>
        </NegotiationRange> *
        <ServiceLevelObjective>
            ...
        </ServiceLevelObjective>*
        <BusinessValueNegList>
            ...
        </BusinessValueNegList>
    </NegotiationTerm>
```

/NegotiationTerm encloses a description of a negotiation term that can be applied under specific circumstances;

/NegotiationTerm/@wsag:Name represents the name given to a term;

/NegotiationTerm/@Counter represents the maximum number of renegotiations that are allowed with the current term. The attribute is necessary to avoid circumstances where an agreement is fulfilled for too long because it respects the negotiation term instead of the original guarantee. The default value is 1;

/NegotiationTerm/@Monitored is a boolean variable, that identifies across multiple negotiation terms and guarantee terms which one is currently monitored and checked. In other words, the attribute Monitored defines if the guarantee g_i is checked under the values of the Guarantee Term or the values of the Negotiation Term;

/NegotiationTerm/GuaranteeScope is a list of guarantee names referring to the respective wsag:GuaranteeName attributes of one or more of the guarantee terms in the agreement. The negotiation terms can apply to every guarantee in the list and refer to the parameters defined in the respective guarantee terms;

/NegotiationTerms/NegotiationRange represents the range of variable values. It presents a minimum and a maximum value of the corresponding variable that the negotiation term is applied to. If the current value is included in the range, it is possible to negotiate it applying the term;

/NegotiationTerm/ServiceLevelObjective is a list of conditions that must be met to satisfy the negotiation term. The ServiceLevelObjetive is very similar to the homonym term of the guarantees section: it identifies the new conditions that must apply to the renegotiated agreement. They refer to the variables defined in the ServiceDescriptionTerm and used in the GuaranteeTerm defined in the scope; and

/NegotiationTerm/BusinessValueList contains a list of business value parameters associated with a service level objective, each expressing a different feature of the objective.

4.4 Example

Referring to the POS scenario introduced in Section 2, we can see how the extended version of WS-Agreement behaves. We assume that the shopping center and the IT enterprise establish an agreement in order to define interactions and the qualities of the service provided. In the agreement they specify some service terms and guarantee terms for fidelity-card operations. Following the operation and the interaction's model stated in the WS-Agreement specification, consumer and provider negotiate resources and qualities of the services.

For instance, besides the agreement about services and guarantees, with the extension it is possible to add some negotiation terms that give the freedom to change the agreement at runtime.

The main and exclusive service defined in the agreement is the execution of fidelity-card operation. Associated with this service we specify two variables that are bandwidth and memory, which can be checked on the service provider side by a monitoring system. Depending on this variable, it is simple to identify some service's properties like the number of operation's execution per minute, the number of request per minute and the service cost. We specify the metric of the variable and in the section dedicated to the guarantee statement we assign ranges of values that should be met to fulfill the current agreement.

Let us consider an agreement example adapting the WS-Agreement structure to our example.

```
<wsrp:GetResourcePropertyResponse>
    <wsag:Name>AgreementExample</wsag:Name>
    <wsag:Context/>
    <wsag:Terms>
    <wsag:All>
        ....
        <wsag:All>
            <wsag:ServiceDescriptionTerm wsag:Name="bandWidth"
                    wsag:ServiceName="Operation">
            </wsag:ServiceDescriptionTerm>
            <wsag:ServiceDescriptionTerm wsag:Name="memorySize"
                    wsag:ServiceName="Operation">
            </wsag:ServiceDescriptionTerm>
        </wsag:All>

        <wsag:ServiceProperties wsag:ServiceName="Operation">
            <wsag:VariableSet>
                <wsag:Variable wsag:Name="requestMinute"
                        wsag:Metric="time:duration">
```

```
                <wsag:Location>
                    ...
                </wsag:Location>
            </wsag:Variable>
            <wsag:Variable wsag:Name="numberOfOperationMin"
                    wsag:Metric="time:duration">
                <wsag:Location>
                    ...
                </wsag:Location>
            </wsag:Variable>
            <wsag:Variable wsag:Name="serviceCost"
                    wsag:Metric="float">
                <wsag:Location>
                    ...
                </wsag:Location>
            </wsag:Variable>
        </wsag:VariableSet>
    </wsag:ServiceProperties>

    <wsag:GuaranteeTerm wsag:Name="operationRequestMinute"
            Monitored="True" Negotiability="True">
        ...
        <wsag:ServiceLevelObjective>
            requestMinute IS_LESS_INCLUSIVE 5
        </wsag:ServiceLevelObjective>
        ...
    </wsag:GuaranteeTerm>

    <wsag:GuaranteeTerm wsag:Name="operationMinuteCount"
            Monitored="True" Negotiability="True">
        ...
        <wsag:ServiceLevelObjective>
            numberOfOperationMinute IS_MORE_INCLUSIVE 12
        </wsag:ServiceLevelObjective>
        ...
    </wsag:GuaranteeTerm>

    <wsag:GuaranteeTerm wsag:Name="operationCost"
            Monitored="True" Negotiability="True">
        ...
        <wsag:ServiceLevelObjective>
            serviceCost IS_LESS_INCLUSIVE 1
        </wsag:ServiceLevelObjective>
        ...
    </wsag:GuaranteeTerm>
```

```
83
84          <wsag:NegotiationTerm wsag:Name="Neg1"
85                  Counter="2" Monitored="False">
86              <wsag:GuaranteeScope>
87                  <wsag:GuaranteeName>
88                      operationMinuteCount
89                  </wsag:GuaranteeName>
90                  <wsag:GuaranteeName>
91                      operationCost
92                  </wsag:GuaranteeName>
93                  <wsag:GuaranteeName>
94                      operationRequestMinute
95                  </wsag:GuaranteeName>
96              </wsag:GuaranteeScope>
97              <NegotiationRange>
98                  <wsag:GuaranteeName>
99                      operationRequestMinute
100                 </wsag:GuaranteeName>
101                 <--! requestMinute values-->
102                 <Minimum>4</Minimum>
103                 <Maximun>6</Maximun>
104             </NegotiationRange>
105             <wsag:ServiceLevelObjective>
106                 <ServiceLevelObjectiveAssertion>
107                     numberOfOperationMin IS_MORE_INCLUSIVE 24
108                 </ServiceLevelObjectiveAssertion>
109                 <ServiceLevelObjectiveAssertion>
110                     serviceCost IS_LESS_INCLUSIVE 2
111                 </ServiceLevelObjectiveAssertion>
112             </wsag:ServiceLevelObjective>
113             <wsag:BusinessValueList>
114                 ....
115             </wsag:BusinessValueList>
116         </wsag:NegotiationTerm>
117     </wsag:All>
118     </wsag:Terms>
119 <wsrp:GetResourcePropertyResponse>
```

Service consumer and service provider start their interactions taking into account the established agreement described above. In this scenario it is possible that the monitoring system at provider side notices that the consumer sends more requests per minute than the number stated in the agreement, exceeding the maximum value, 4 (defined in the guarantee at line 43). For instance, the provider can not fulfill all the requests from the consumer as previously agreed. Thanks to the proposed extension, it is possible to renegotiate the current guarantee. In the *NegotiationTerms* (lines 84 to 107), there is a term referring to the current

guarantee that gives the freedom to increase the number of requests per minute up to 24, if service cost is increased of 2 USD. Applying this negotiation term, defined and agreed on by both service consumer and provider at agreement creation's time, the consumer will pay more, but can ask more executions per minute: in this case an increase of performance means an increase of service cost.

In this simple execution on the running example, we see that using the extension it is possible to maintain the current agreement, mediating guarantees that are currently violated and guarantee that are widely fulfilled. Instead, using the original version the agreement must terminate as soon as a guarantee is violated.

5 Concluding Remarks

Describing and invoking an individual functionality of a web service is becoming more and more common practice. Most of the major players on the Web already offer their functionalities via web services (e.g., Google, Amazon, and eBay). One of the next steps is moving from functional properties of basic services to non-functional properties of composed services. The non-functional properties need to be specified by the services, but also to be negotiated among services.

WS-Agreement is a protocol that defines a syntax to specify a number of guarantee terms within an agreement. We looked into the protocol specification with the goal of providing a formalization of the notion of an agreement and proposing a formal representation for the internal and external states in which an agreement can be. From this analysis we discovered that an agreement can be made more long-lived, if we extend the protocol by tolerating some guarantee violations. We presented the details of the proposed extension in formal terms and provided a simple example showing the potentials of the extension.

This work prods for more investigation of agreements and of their management. First, one may consider further properties of the agreement protocol by resorting to a formal specification language (in the spirit of [11] which have used TLA+ for the WS-Transaction protocol). Second, one can consider guarantee terms to be in more states than simply not determined, fulfilled or violated. One could introduce states that specify a guarantee term to be fulfilled but close to being violated, for instance. Finally, one may want to experiment with a reference implementation of the extended version—in the spirit of [13]—and check quantitatively the advantages of such an extension.

Acknowledgments

Marco thanks Asit Dan and Heiko Ludwig for useful discussion on WS-Agreement while visiting IBM TJ Watson.

References

1. M. Aiello and P. Giorgini. Applying the Tropos methodology for analysing web services requirements and reasoning about Qualities of Services. *CEPIS Upgrade - The European journal of the informatics professional*, 5(4), 2004.

2. M. Aiello, M. Zanoni, and A. Zolet. Exploring web-service notification: Building a scalable domotic infrastructure. *DrDobbs Journal: Software Tools for the Professional Developer*, 371:48–51, 2005.
3. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Grid Resource allocation Agreement Protocol (GRAAP) WG, 2004.
4. A. Andrieux, A. Dan, K. Keahey, H. Ludwig, and J. Rofrano. Negotiability constraints in WS-Agreement. Technical report, Grid Resource Allocation Agreement Protocol (GRAAP) Working Group Meetings, 2004.
5. J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 2004. To appear.
6. K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, and M. Xu. Agreement-based Grid Service Management (OGSI-Agreement). Technical report, Global Grid Forum, GRAAP-WG Author Contribution, 2003.
7. A. Dan, K. Keahey, H. Ludwig, and J. Rofrano. Guarantee Terms in WS-Agreement. Technical report, Grid Resource Allocation Agreement Protocol (GRAAP) Working Group Meetings, 2004.
8. V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian. A quality of service management framework based on user expectations. In *Service-Oriented Computing (ICSOC)*, pages 104–114. LNCS 2910, Springer, 2003.
9. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *IEEE Computer*, 35(6), 2002.
10. D. Gouscos, M. Kalikakis, and P. Georgiadis. An approach to modeling web service QoS and provision price. In *1st Web Services Quality Workshop (WQW2003) at WISE*, 2003.
11. J. Johnson, D. Langworthy, L. Lamport, and F. Vogt. Formal specification of a web services protocol. In *Proc. of 1st Int. Ws. Web Services and Formal Methods (WS-FM 2004)*, volume 105 of *ENTCS*, Pisa, Italy, 2004. Elsevier Science Publishers.
12. H. Ludwig. Web services QoS: External SLAs and internal policies or: How do we deliver what we promise? In *1st Web Services Quality Workshop (WQW2003) at WISE*, 2003.
13. H. Ludwig, A. Dan, and R. Kearney. CREMONA: an architecture and library for creation and monitoring of ws-agreements. In M. Aiello, M. Aoyama, F. Curbera, and M. Papazoglou, editors, *ICSOC*, pages 65–74. ACM, 2004.
14. D. Malfatti. A framework for the monitoring of the QoS by extending WS-Agreement. Master's thesis, Corso di Laurea in Informatica, Università degli Studi di Trento, 2005. In Italian.
15. A. Mani and A. Nagarajan. Understanding quality of service for web services, 2002. `http://www-106.ibm.com/developerworks/library/ws-quality.html`.
16. O. Martn-Daz, A. Ruiz Corts, A. Durn, D. Benavides, and M. Toro. Automating the procurement of web services. In *Service-Oriented Computing (ICSOC)*, pages 91–103. LNCS 2910, Springer, 2003.
17. M. P. Singh and A. Soydan Bilgin. A DAML-based repository for QoS-aware semantic web service selection. In *IEEE International Conference on Web Services (ICWS 2004)*, 2004.
18. M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller. A concept for QoS integration in web services. In *1st Web Services Quality Workshop (WQW2003) at WISE*, 2003.