Normative Management of Web Service Level Agreements

Caroline Herssens[1], Stéphane Faulkner[2], and Ivan J. Jureta[2]

[1] PRECISE, LSM, Université catholique de Louvain, Belgium
[2] PRECISE, LSM, University of Namur, Belgium
`caroline.herssens@uclouvain.be`,
{`stephane.faulkner,ivan.jureta`}`@fundp.ac.be`

Abstract. Service Level Agreements (SLAs) are used in Service-Oriented Computing to define the obligations of the parties involved in a transaction. SLAs define these obligations, including for instance the expected service levels to be delivered by the provider, and the payment expected from the client. The obligations of the parties must be made explicit prior to the transaction, and a mechanism should be available to control the interaction, in order to ensure that the obligations are met. We outline a norm-oriented multiagent system (NoMAS) architecture that is combined with the service-oriented architecture in order to support the definition, management, and control of SLAs between the service clients and service providers.

Keywords: SLA, management, mutual obligations, supervision, norm oriented multi-agent systems.

1 Introduction

We focus in this paper on the critical task of ensuring that the contractual obligations of the parties – the service providers and the service clients – involved in a transaction are respected by these parties within a service-oriented system. Their obligations are typically outlined in a service-level agreement (SLA). An SLA is a contract between the said parties, who specify the quality-of-service (QoS) levels that should be met [17]. QoS is a combination of several quality properties, e.g., availability, reliability, cost, response time [21]. A provider can propose the same service at different quality QoS levels. When a service client requests the execution of a given functionality, it advertises its QoS expectations. The service selected for the service execution will be the one that best satisfy client expectations about QoS properties. Prior to the transaction, the client and the provider enter into a contract by signing an SLA, and thereby specify quality levels to be observed during the service execution [17]. SLAs are used in the QoS management context in order to know what clients requirements to meet, how to manage clients expectations, how to regulate resources and to control costs [26].

The use of SLAs in managing the transaction between a provider and a client requires appropriate conceptual foundations and associated computational mechanisms. SLAs require an architecture if they are to enable the interactions between stakeholders. This architecture must support a specification language used by the stakeholders to communicate their expectations and capabilities. Similarly, a specification language to define the elements of the SLA is needed. Beside the architecture, the SLA management needs an incentive mechanism. To stimulate the correct behavior of stakeholders, these must have mutual obligations. E.g., the provider has the obligation to meet a given QoS level and the client has the obligation to pay according to that quality level. However, the client pays for the service after its execution by the provider. It follows that the client's payment can be adapted to the QoS level delivered by the provider. Finally, stakeholders can behave in an opportunistic manner, i.e., the client can underevaluate the QoS level perceived and the provider can exaggerate the QoS level offered. To prevent such situations, the architecture must introduce a third-party controller to monitor the SLA execution.

The architecture must support the adaptation of SLA during the service execution. This architecture needs to be responsive and flexible. Norm-oriented Multi-Agent Systems (NoMAS) provide these characteristics. Normative agents refer to agents conforming to norms. The core idea of this paper is to adapt the analogy of norms and agents to the issue of SLA and stakeholders. An SLA will be described as a set of norms to be fulfilled by the different agents of the system. The stakeholders of the service execution will be represented by normative agents complying to norms that restrict their behavior. The architecture supports a language enabling the communication between stakeholders. This language allows to express norms to be followed by the normative agents of the MAS. The elements constitutive of the SLA are defined by obligations norms regulating the stakeholders.

Contributions. We propose an architecture based on normative agents in order to: (i) enable the communication between stakeholders involved in the SLA with a common language; (ii) define SLAs that meet provider capabilities and client requirements; (iii) manage the service execution and check the conformance of the quality level expected and observed; (iv) ensure the execution of the mutual obligations according to the SLA contract. We propose to achieve the SLA compliance through two particular mechanisms: *mutual obligations*, which motivate the fulfillment of respective obligations of the involved stakeholders; and a *supervised interaction* with a third-party controller, which monitors and evaluates the SLA execution and penalizes the agents that does not fulfill their obligations.

Organization. Section 2 presents the conceptual foundations of the approach. Section 3 outlines the management architecture and the management of SLA. Section 4 proposes an evaluation of the proposed approach. Section 5 summarizes the related work and the Section 6 concludes this paper.

2 Case Study and Conceptual Foundations

This Section covers the conceptual foundations of our SLA management approach. We first describe the case study used to illustrate the approach proposed in this paper. We briefly introduce the Service Level Agreement concept. We also outline the mutual obligations and the supervised interaction used throughout the approach.

2.1 Case Study

We refer in this paper to a case study coming from the European Space Agency (ESA) program on Earth observation. This program allows researchers to access and use the infrastructure operated and the data collected by the agency[1]. The data and infrastructure of the ESA are accessed through web services. In order to facilitate the discussion and delimit our example, we focus on one part of the overall system. The MERIS/MGVI service is a service able to use the MERIS instrument data provided by the Envisat satellite of the ESA to compute the vegetation indexes for a given period of time and region of the world. A vegetation index measures the amount of vegetation on the Earth's surface. The data on the vegetation index can be obtained for any time range and it is possible to delimit the region of the world that is of interest. This service is subject to one particular QoS characteristic: the latency is initially situated between 4 and 6 hours by day of the selected period. E.g.: if the time range selected is from October 24th 2009 to October 26th 2009, the execution time needed to compute the vegetation index is set between 12 to 18 hours. The length of the selected period impacts then strongly the time needed to fulfill the request. The SLA specification between stakeholders of this service must clearly constrain the execution time prior to all remaining QoS properties. The different concepts presented in this paper will be illustrated with the MERIS/MGVI service and, specially about the execution time of this service.

2.2 Service Level Agreement

A Service Level Agreement is a contract between the service provider and the service client specifying mutually agreed obligations of the provision of a service [6,30]. The SLA concerns the non-functional properties of the service [17], i.e., quality properties. When clients can choose among a set of functionally equivalent web services, Quality of Service (QoS) considerations become the key criteria for service selection. As a consequence, SLA about nonfunctional properties must be defined and managed between service clients and providers [17].

The specification of QoS obligations of a SLA starts from a set of Service Level Objective (SLOs) [26]. A Service Level Objective is a guarantee of a particular state of the SLA parameters in a given time period [13]. All quality properties advertised by the provider are associated to an SLO as illustrated in Example 1. Each SLO has a functional part that refers to the QoS concerned and a guarantee part (italicized in Example 1) applied on the functional part. With SLOs, the SLA covers all quality properties defined in the QoS request of the service client.

[1] `http://gpod.eo.esa.int`

Example 1. The provider of the MERIS/MGVI service shall execute the service *within 5 hours by day of the selected period.*

Example 1 is the SLO stating the maximum execution time of the agreement defined between the provider and the client. As referred in Section 2, the execution time of the MERIS/MGVI service is very important and needs to be clearly defined in the SLA. The SLA definition is communicated between the different stakeholders of the service execution. To assure the interoperability of SLA definitions, their specifications need to be written in a language common to providers and clients. The Web Service Level Agreement (WSLA) language [13] is one of the main standard for specifying SLAs. The Example 2 illustrates how the SLO agreement of the Example 1 is specified with WSLA.

Example 2.

```
<ServiceLevelObjective name=''exectime''>
  <Obliged>provider</Obliged>
  <Validity>
    <Start>2009-10-25T08:00:00.000-05:00</Start>
    <End>2009-10-30T08:00:00.000-05:00</End>
  </Validity>
  <Expression>
    <Predicate xsi:type=''wsla:Less''>
      <SLAParameter>ExecutionTime</SLAParameter>
      <Value>ExecutionTimeThreshold</Value>
    </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
<ServiceLevelObjective>
```

The `ExecutionTimeThreshold` used in Example 1 is a constant that assigns a name to a simple value that can be referred in other definitions [13]. This threshold corresponds to the maximal execution time expected by the client, i.e., 5 hours by day of the selected period to compute.

2.3 Mutual Obligations

Delivering the service at the quality level specified in the SLA is an obligation for the service provider. However, the service client has an obligation to provide all the information needed for the service execution (i.e.: inputs needed for web service execution), but also to pay for the service execution. Interactions between the provider and the client involve mutual obligations [20]. Such bilateral obligations motivate the SLA conformance. Indeed, breaches to some obligations of one party can compromise the fulfillment of obligations of the other party. Both parts have interest in achieving their obligations to meet the contract. Goodin [10] outlines the possible structures of mutual obligations. The SLA of web services can be defined as mutually conditional obligations. With mutual conditional obligations, each party is obliged to discharge his obligations if and only if the other party discharges his obligations. E.g., if the service provided does not meet the contracted execution time, the client has not to pay the amount initially set. The SLA defines mutual obligations compelling the respective behavior of stakeholders.

The service execution is made of bilateral obligations, i.e., unilateral obligations from the provider about the service level execution and unilateral obligations of the client about payment or rating [16,22]. We consider in the remainder of this paper that the client obligation is only about payment. However, other contractual obligations can be used as feedback rating as requests frequency.

The execution of obligations occurs sequentially: obligations of one of the stakeholders are executed before the obligations of the other. E.g., the provider's obligations are executed before the client's and the level of payment can consequently be adapted to the degree, to which the provider conforms to the obligations. Adaptations of the client obligations according to the observed quality level must be specified in the initial SLA. In the classification of mutual obligations [10], SLA contracts are diachronic mutual obligations, because one party is supposed to discharge its obligations before the other party does the same. The consequence is that initial contract must specifies the expected penalties if the defined quality level is not met [16]. The SLA contract implies that the penalty is initially accepted by the provider. Clearly, the efficiency of the relationship existing between the client and the provider improves if specifications of penalties for cases of contract breaches are present.

2.4 Supervised Interaction

Stakeholders of the service execution must achieve their respective obligations to conform to the initial SLA. If they are not supervised, they can adopt an opportunistic behavior, i.e., not fulfill their obligations or fulfilling them at a level lower than expected. E.g., the service client can reduce the payment even if the quality level provided meets is expectation. To prevent such situations, we propose to monitor the service execution with a third-party. This third-party will act as a controlling authority and allows to ensure the correct execution of the SLA. It is a witness of the service execution and stimulates the conformance to SLA for both involved parts. The third-party allows deterrence-based trust, i.e., you trust the other party because there is a very strict rule normative or legal system of rules, and the agent is punished for any violation of rules [7]. The third-party is the controller that controls the compliance of both parts to rules defined in the SLA, it measures the efficiency of the stakeholders transactions and computes their respective reputations [23]. An analogy of the third-party controller is the ebay online auction website[2]. The evaluation system of ebay prevents the opportunistic behavior of the stakeholders of the transaction.

This authority has an additional role in managing the SLA. Namely, it is in charge of collecting and computing metrics. It collects and stores metrics defined in the SLA and computes them to compare observed and expected results. Such metrics are used to establish the trust value of stakeholders involved in transactions. The measurement of quality values is allowed by existing metrics

[2] `http://www.ebay.com`

such as those discussed in [8]. If an SLA is breached, the third-party controller sends notifications to the involved stakeholders. The third-party is independent of the parties involved in the actual transaction, given that its aim is to prevent opportunistic behavior.

3 The Architecture and the Process for SLA Management

To solve the issues of SLA definition and its management during the service execution, we propose to use a normative MAS. Our proposed system will allow the SLA management and stimulates the SLA compliance through the respect of mutual obligations and the supervision of an authority. We first introduce our agent architecture that monitors the SLA through the service stakeholders in Subsection 3.1. We then explain how SLAs are managed with this architecture through the definition of norms associated to the stakeholders in Subsection 3.2.

3.1 SLA Management Architecture

We chose to use a normative multi-agent system to monitor the execution of SLA. A normative multiagent system (MAS) involves normative mechanisms, which allow agents to adopt norms and specify how agents can modify these norms [4]. Norms can increase the efficiency of agent reasoning while their explicit representation supports reasoning about a wide range of behaviour types in a single framework [9]. Agent norms describe the obligations, permissions and prohibitions of a norm addressee to pursue certain activities, either to achieve a state of affairs or to perform an action [18]. The behaviour of an agent is monitored by its norms defining its permissions and obligations. Such a normative system allows deterrence based trust, the agent is punished for any violation of rules of the normative system [7].

The stakeholders of the service execution and the third-party controller are managed by normative agents. Norms condition the behavior of agents, the SLA is defined by obligations and prohibitions restraining the set of possible actions. We manage SLAs with normative agents coordinated within a suitable architecture. Three kinds of normative agents step in this architecture: the provider agents, the client agents, and the cluster agents. These are illustrated in Figure 1.

A cluster agent (A_{Clus}) is dedicated to each existing cluster of web services. A cluster of web services gathers functionally equivalent web services by providing several web services inside a unique wrapper. This wrapper is used by service clients as a standard web service. Services in a same cluster can be offered by different providers. The cluster selects the service that best satisfies the QoS expectations of the client in the cluster with an appropriate selection method [12]. This method relies on QoS advertisements of the provider and QoS expectations of the service client. These advertisements and expectations can have be made with WSLA [13] or another common appropriate language.

A service provider agent (A_P) is dedicated to each existing provider in the service cluster. A provider can offer several services in the cluster, i.e., the same

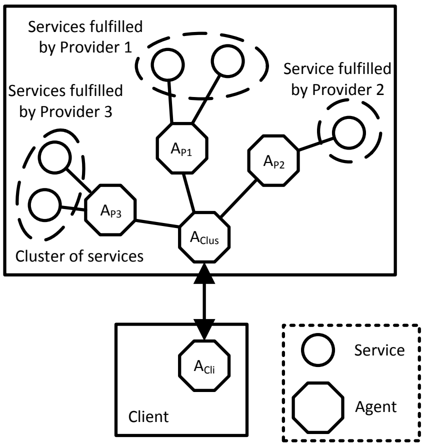
Services fulfilled by Provider 1
Service fulfilled by Provider 2
Services fulfilled by Provider 3
A_P1
A_P2
A_P3
A_Clus
Cluster of services
A_Cli
Client
Service
Agent

Fig. 1. SLA management architecture

functionality at different quality levels. This is illustrated in Figure 1 with the provider 1 and 3 each offering services with the same functionality and different quality characteristics. Providers can also offer services into different clusters, as they provide different functionalities. Provider agents advertise their QoS possibilities to the cluster agent.

A client agent (A_{Cli}) is assigned to each client requesting a service. The service request includes particular QoS expectations of the service client about the service execution.

Once the SLA has been negotiated [30] between stakeholders of the service execution it can be defined with an appropriate language. The cluster agent is responsible of the stakeholders conformance to the SLA defined. The cluster agent is also in charge of the collection and evaluation of metrics of the service execution. It collects information about QoS observed at each service transaction. It is able to compute statistical data on the service and able to determine if the service execution met the defined SLA. It is the third-party controller of the service execution, it controls the SLA compliance of the provider and the client agent. Agents of the SLA architecture are normative agents, conforming to norms derived from the SLA initially defined between the client and the provider. This architecture can be easily supported by existing normative MAS frameworks [9,18]. In the remainder, we focus on how normative agents can support the management of SLA. The normative MAS infrastructure and communication is out of scope of this paper.

In the context of our case study, an agent is dedicated to each provider able to propose a service functionally equivalent to the MERIS/MGVI service introduced in the case study. A client agent is dedicated to the requester of the MERIS/MGVI service. A cluster agent is responsible for the supervision of interactions occurring between the client agent and the providers offering the functionality.

3.2 SLA Management Process

The management of SLAs with the proposed architecture covers several steps: (1) the definition of the SLA between the client and the provider; (2) the control of provider obligations, i.e., the service execution; (3) the penalties to apply to the provider if the SLA is not met, and; (4) the control of the client obligations, i.e., payment or evaluation. To fulfill these steps, the agents of the architecture introduced in Subsection 3.1 will take on different roles.

Step 1: Definition of the SLA. The SLA is defined between the service client and the service provider from WSLA specification advertised by the provider. The WSLA specification is extended to include the mutual obligations of stakeholders. The defined SLA must cover expectations about the quality level of the service execution but also the penalties associated to breaches of SLA. These penalties are defined according to the importance of quality properties involved and according to the importance of breaches. Moreover, initial SLA specifications can also be enriched with complex rules, dependent rules or normative rules [24]. Such extensions allow the definition of enriched contracts, e.g., graduated rules are rules sets which specify graduated range for certain parameters so that it can be evaluated whether the measured values exceed, meet or fall below the defined service levels. To define these extended SLAs and include mutual obligations of service providers and clients, usual languages as WSLA [17] or WSOL [28] need to be enriched. To this aim, we choose to express the different SLOs of the initial SLA with obligation norms associated to involved agents of the architecture to benefit from the information added by more complex rules. To express SLO with normative obligations, we refer to the work of Kollingbaum [18,19] about supervised interaction. Each SLO of the SLA contracted between the provider and the client is expressed with the NoA language [18] interpretable by all agents of the architecture. Moreover, complex conditions and penalties associated to SLO failures are also expressed with this language in further steps. The Example 3 illustrates the conversion of the SLO specified with WSLA in Example 2 into an obligation norm of the service provider agent specified with the NoA language.

Example 3.

```
obligation(
ServiceProvider,
achieve ServiceExecutionUnderTimeThreshold (ServiceProvider, Service,
  ExecutionTimeThreshold),
ServiceExecuted (ServiceProvider, Service) and
  ExecutionTime (Service) <= ExecutionTimeThreshold
ServiceExecutionUnderTimeThreshold (ServiceProvider, Service,
  ExecutionTimeThreshold))
```

This obligation states that the provider must achieve the execution of the MERIS/ MGVI service under the time limit (`ServiceExecutionUnderTimeThreshold`) specified in the initial WSLA specification (`ExecutionTimeThreshold`). The

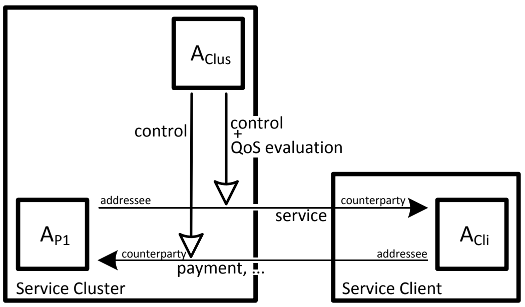

Fig. 2. Roles fulfilled by normative agents

normative agents of the architecture monitor the execution of services through their norms. However, these agents can make a choice whether to obey the norms in specific cases. If the service provider is not able to achieve all SLOs of its contract, it can violate some of them to assure the fulfillment of remaining norms. This situation arises due to unexpected events (i.e.: additional requests, hardware failures) or because the provider amplified its capabilities to be selected. Among all SLOs defined with obligations norms between the client and the provider, some can be met and some can not.

Step 2: Control of provider obligations. Control is enabled through mechanisms of normative agents. Each agent of the architecture fulfills one or several roles in the contract management. The SLA contract is then monitored through these different roles: the addressee commits an obligation defined in the contract; the counter-party is the recipient of the obligation fulfilled, and; the authority is a witness of the contract. The authority is in charge of the correct execution of the contract and imposes sanctions in case of defective behavior of the addressee. The different roles of client, provider and cluster agents are illustrated in Figure 2. The provider illustrated in the service cluster of this example is the provider 1 among those proposed in Figure 1.

The interactions between the service provider and the service client (i.e., the service execution and its payment) are restrained by obligation norms associated to these roles. The SLOs specifying the expected QoS level of the MERIS/MGVI service appear as norms. The provider agent is the addressee in these norms, while the client is the counter-party in the transaction. To control the achievement of this contract, the cluster agent acts as an authority. As stated in Subsection 3.1, the cluster agent is responsible for collecting and computing the metrics in order to control the SLA execution. It is then able to determine if the service provider meets the SLOs defined through obligation norms. The cluster agent will impose sanctions when the quality level provided does not meet the level contracted in the SLA. Such sanctions appear as penalties applied to the provider. As stated before, these penalties are part of the initial SLA. In the

MERIS/MGVI instance, the decreasing of payment is proportional to the reduction of quality level provided. Sanctions are expressed by obligations norms to be followed by the cluster agent. When the provider chooses or is forced to breach a norm specifying one of its SLO, the cluster agent captures it and activates a specific penalty. There can be several norms specifying different penalties corresponding to the spreading of the breach. The Example 4 illustrates a specification of one such penalty.

Example 4.

```
sanction(
ServiceCluster,
perform EvaluationTime (ServiceProvider, Service, ExecutionTimeThreshold,
  ExecutionTimeThreshold 2, AmountPenalty),
ServiceExecuted (ServiceProvider, Service) and
  ExecutionTime (Service) > ExecutionTimeThreshold and
  ExecutionTime (Service) <= ExecutionTimeThreshold2
TimePenalty(ServiceProvider, AmountPenalty))
```

When one of the SLOs of the MERIS/MGVI service is not met, a sanction is applied by the cluster agent according to the importance of the breach. As stated in Section 2, the execution time is a critical issue for the MERIS/MGVI service, sanctions to apply must penalize all provider weaknesses about delays. The Example 4 illustrates one sanction: if the execution time observed is above the SLA time limit (`ExecutionTimeThreshold`) but is under the second time limit of the breach scale (`ExecutionTimeThreshold2`), the decreasing of payment (`AmountPenalty`) applied is proportional to the observed level on the breach scale. The cluster agent independently estimates the equality of the quality level provided and the amount to pay. Moreover, according to characteristics of mutual obligations in SLAs, the user must discharge its obligations only if the provider has discharged its owns obligations.

Step 3: Penalties to apply. When the cluster agent observes that the SLA is not fulfilled by the provider agent, it notifies the service client through the application of a sanction. The client agent will then reflect this sanction on its own behavior. The mutual obligations of SLAs are diachronic; the client obligations are adapted to the provider fulfillment of its owns obligations. In the Example 5, the `TimePenalty` is a constant defining the payment reduction of the MERIS/MGVI service initiated by the sanction of the Example 4. There can be several payment reduction to apply, corresponding to different level of breach or to different QoS properties involved in the SLA. According to the importance of the breach, the client agent follows the norm defining the corresponding penalty. The obligation of the Example 5 is the client obligation to pay for the MERIS/MGVI service execution, i.e., the contractual obligation of the client. However, the initial payment amount (`Amount`) is reduced by the penalty (`AmountPenalty`) induced by the time sanction illustrated in Example 4. The payment of the service is an obligation norm in which the client agent is the addressee and the provider agent is the counter-party as illustrated in Figure 2.

Example 5.

```
obligation(
ServiceClient,
achieve ServicePayment (ServiceClient, ServiceProvider,
  Amount - AmountPenalty),
ServiceExecutionUnderExecutionTimeThreshold (ServiceProvider, Service,
  ExecutionTimeThreshold)) and
  TimePenalty(ServiceProvider, AmountPenalty)
achieve ServicePayment (ServiceClient, ServiceProvider,
  Amount - AmountPenalty))
```

Step 4: Control of client obligations. The third-party controller checks the execution of the unilateral obligations of the service provider as detailed in Step 2. Similarly, the third-party controller must verify the obligations of the service client, the client can be subject to different categories of obligations i.e.: its payment to the provider after the service execution. To control the right execution of the payment obligation illustrated in Example 5, the cluster agent acts as the third-party controller. It is the authority of the payment transaction as illustrated in Figure 2. It must check that the right amount has been deposited to the provider. If the client fails to pay or deposits a bad amount, the cluster agent must apply a penalty. The Example 6 illustrates the sanction applied by the cluster agent to the client of the MERIS/MGVI service when the payment obligation is not met. With such sanctions, the cluster agent avoids the non payment of the service client. Indeed, if the payment is not made or if it is insufficient, the client is labeled as a bad payer (`PaymentPenalty(ServiceClient)`) by the third-party controller. The cluster agent can then reject future requests of bad payers on its cluster.

Example 6.

```
sanction(
ServiceCluster,
perform CheckPayment ServiceClient, ServiceProvider, Amount, AmountPenalty),
not ServicePayment (ServiceClient, ServiceProvider, Amount - AmountPenalty)
PaymentPenalty(ServiceClient))
```

4 Evaluation

Supervised interaction and mutual obligations ensure that delivery of services is better managed. We conduct some experiments in order to evaluate the effect of these mechanisms. These experiments simulate services transactions between users and providers and measure their utility with and without the utilization of such mechanisms. The utility denotes the abstract quality whereby an object serves our purposes, and becomes entitled to rank as a commodity [15]. We suppose here that the utility increasing is constant for each new transaction initiated. Each transaction initiated by a client involves a cost decreasing of its cumulated utility while each successful transaction increases its cumulated utility. The ratio over the increasing induced by the success of the transaction and the decreasing due to the cost of the transaction must be positive. E.g.: in our simulations, the increasing of utility is set to 1 while the service execution succeeds and the utility decreasing of the service payment is 0.8. The net utility of a service transaction is then 0.2. We generate 200000 transactions from 10000 different providers to 100 different users. Each service is executed 20 times by each service client. To simulate the opportunistic behavior of providers, we define 30% of opportunistic providers that do not fulfill their transactions 70% of time. Without mutual obligations and supervised interaction, the decreasing of client utility involved by the service payment occurs even while services executions fail.

To simulate the supervised interaction effect, we introduce a simple trust model. The trustworthiness of each provider is collected by the third-party controller. The third-party controller monitors all services executions and dismisses providers that fail 10 services executions previously supervised. While services executions occur without supervised interaction, the clients collect themselves information about past executions and dismiss providers that failed 3 of their own previous transactions.

To simulate the interest of mutual obligations, we introduce a variable payment model. The service client can reduce the initial payment while the provider obligations are not met. The utility decreasing of the service client can be less important when the service execution fails. E.g.: in our simulations, the utility decreasing involved by the payment is 0.8 when the service execution succeeds and is reduced to 0.2 while the service execution fails. Without mutual obligations, the payment has to be done and the decreasing of the client utility is fixed to 0.8.

However, the third-party controller offering such monitoring mechanisms has to be payed. We designed two different scenarios to simulate the payment of the third-party controller: a variable and a fixed remuneration. The variable remuneration implies a decreasing of the client utility at each service execution. This variable cost must be proportional to benefit of a transaction. E.g.: if the net utility before the remuneration of the third-party controller is 0.2, the third-party fee of each transaction can be 0.02. The fixed cost allows clients to benefit from third-party mechanisms after a single payment. It implies an important decreasing of the client utility. E.g.: in our simulations, we set the initial utility of the client to -1000 while the third-party controller relies on a fixed cost.

To evaluate benefits from supervised interaction and mutual obligations, we observe the mean cumulated utility of users during 200000 services executions. To highlight the benefits of third-party mechanisms, we design 7 models: (1) services executions without supervised interaction (*s.i.*) and without mutual obligations (*m.o.*); (2) services executions without *s.i.* and with *m.o.* at a variable cost; (3) services executions without *s.i.* and with *m.o.* at a fixed cost; (4) services executions with *s.i.* and without *m.o.* at a variable cost; (5) services executions with *s.i.* and without *m.o.* at a fixed cost; (6) services executions with *s.i.* and *m.o.* at a fixed cost, and; (7) services executions with *s.i.* and *m.o.* at a variable cost. We then measure the difference between the optimal cumulated utility and the cumulated utility of our different models (i.e., the optimal client utility is get while the service client never pays for the services executions that fail).

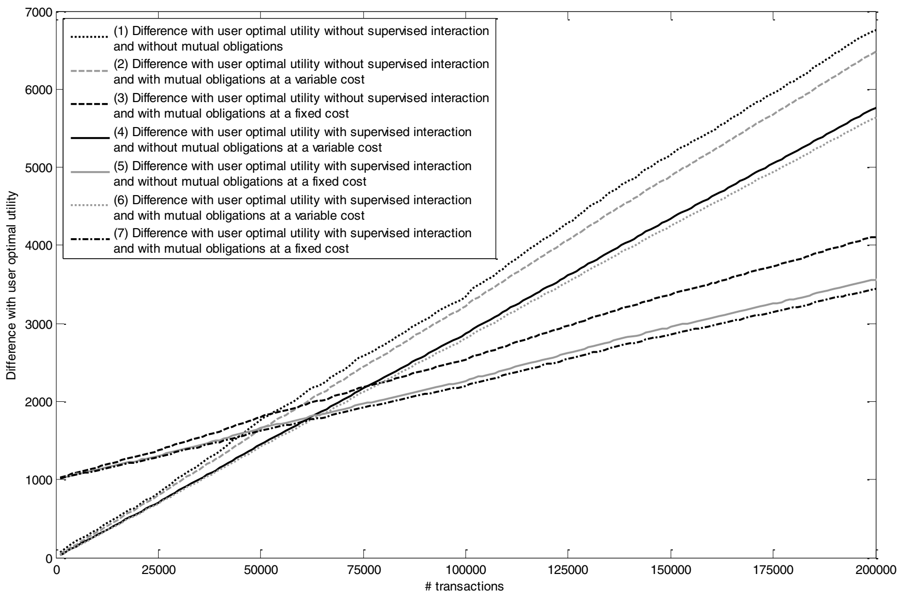

Fig. 3. Simulation Results

The results of our experiments are highlighted in Figure 3. The model nearest to the optimal client utility is the model (7) that provides both supervised interaction and mutual obligations with an initial fixed cost. However, this model becomes the best only when the initial cost is balanced by its profitability (after approximatively 64500 services executions) while at the beginning the most profitable model is the model (6) that provides both supervised interaction and mutual obligations at a variable cost. The profitability of each model is dependent from the third-party controller payment scenario but the utilization of third-party mechanisms always improve the client utility. The less profitable model is the (1) that provides neither supervised interaction nor mutual obligations. Models offering only mutual obligations ((2) and (3)) improve lightly the client utility while models providing only supervised interaction ((4) and (5)) ameliorate strongly the client utility. The combination of both mechanisms (models (6) and (7)) outperforms other models and highlight the interest of normative agents to control SLA of stakeholders transactions. The experiments conducted here to evaluate the client utility can be transposed to the provider utility. We can also simulate opportunistic client that do not fulfill their obligations and evaluate the mean utility of providers.

5 Related Work

QoS properties and SLA management need appropriate architectures to be handled during the service execution. Campbell et al. [5] propose the Quality of

Service Architecture (QoS-A) incorporating the notion of flow, service contract and flow management through QoS properties. Barbosa et al. outline in [3] different architectural configurations to enable the auditing of SLA and to evaluate their efficiencies. The WSLA framework [17] introduces a runtime architecture comprising several SLA monitoring services. Some services may be outsourced to third parties to increase the objectivity in the evaluation of the services. The QoS Mission-Action-Resource (Q-MAR) model [14] and the Grid Quality of Service Management (G-QoSM) framework [2] also propose to distribute the SLA monitoring to the different components of the system. Paschke et al. [24] introduce a Rule-Based Service Level Management (RBSLM) architecture in which SLAs are represented with declarative rules and managed through logical concepts and rule languages. Although all these architectures allow one to observe when a contract is violated, most of them do not prevent such violations and do not clearly define corrective actions to take. In our proposal, the third-party monitors stakeholders behaviors and the mutual obligations of the stakeholders define penalties to apply while the contract is not fulfilled. The BREIN project[3] offers an architecture enabling the management of SLAs through their whole lifecycle [1]. The SLA management is enabled by taking into account the policies of the parties and their respective business goals. The BREIN SLA management offers preventive monitoring to react to upcoming violations and a prioritization of SLAs. Our normative management of SLAs adapt contracts at runtime in response to unexpected violations in order to maximize stakeholders satisfaction.

Agents systems are well fitted to monitor activities requiring negotiation between stakeholders as SLA management or e-commerce mediation [11,27]. Other existing SLA architectures relies on multi agent systems [29]. Yan et al. [30] introduces a MAS architecture supporting the negotiation of services involved in a composition. In comparison with existing MAS architectures, our proposal is supported by normative agents. Normative agents allow to constrain the stakeholders behavior with norms defining the SLA to be achieved. They are particularly relevant to the SLA management issue. Normative agents are also used by Pitt et al. [25]. They propose a framework for QoS management which combines events, metrics and parameters with organizational intelligence offered by norm-governed multi-agent systems. Although their proposal monitors QoS information, they did not tackle the SLA conformance issue. One of the strongest point of our work is that the agreements between clients and providers are defined and monitored through norms associated to roles of agents and not to agents or components of the architecture. These roles allow the architecture to offer more flexibility, e.g., the provider can be easily substituted when unexpected failures occurs.

6 Conclusions and Future Work

We propose in this paper an architecture enabling the management of SLA. This architecture relies on a MAS and supports a normative definition of SLA.

[3] `http://www.eu-brein.com`

The MAS enables the communication between stakeholders involved in the SLA. Each party of the SLA is defined with an obligation norm that constrains stakeholders behaviors. The architecture checks the conformance of the stakeholders to the SLA. To stimulate the proper execution of the SLA, its execution is driven by mutual obligations and supervised by a third-party controller. The architecture benefits from the potential autonomy assured by normative agents. The normative architecture enables the interactions between provider and client and also the evaluation of the quality level of such interactions.

Future work will concern the implementation and the extension of the architecture to support actions to take while an SLA is breached. Rather than penalize the provider or the client, the architecture will propose corrective actions. To ensure the SLA conformance, the architecture will take advantage of the multiple services providing the same functionality inside the cluster.

References

1. Final brein architecture d4.1.3 v2 - wp 4.1 architectural design. Technical report, BREIN project (2009)
2. Al-Ali, R.J., Rana, O.F., Walker, D.W., Jha, S., Sohail, S.: G- qosm: Grid service discovery using qos properties. J. of Computing and Informatics 21(4), 363–382 (2002)
3. Barbosa, A.C., Sauvé, J., Cirne, W., Carelli, M.: Evaluating architectures for independently auditing service level agreements. Future Gener. Comput. Syst. 22(7), 721–731 (2006)
4. Boella, G., Torre, L., Verhagen, H.: Introduction to normative multiagent systems. Comput. Math. Organ. Theory 12(2-3), 71–79 (2006)
5. Campbell, A., Coulson, G., Hutchison, D.: A quality of service architecture. SIGCOMM Comput. Commun. Rev. 24(2), 6–27 (1994)
6. Cappiello, C., Comuzzi, M., Plebani, P.: On automated generation of web service level agreements. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 264–278. Springer, Heidelberg (2007)
7. Castelfranchi, C., Tan, Y.-H. (eds.): Trust and deception in virtual societies. Kluwer Academic Publishers, Norwell (2001)
8. Cherkasova, L., Fu, Y., Tang, W., Vahdat, A.: Measuring and characterizing end-to-end internet service performance. ACM Trans. Internet Technol. 3(4), 347–391 (2003)
9. Dignum, F., Morley, D.: Towards socially sophisticated bdi agents. In: Proc. of ICMAS '00, p. 111. IEEE Computer Society, Los Alamitos (2000)
10. Goodin, R.: Structures of mutual obligations. J. of Soc. Pol. 31(4), 579–596 (2002)
11. He, M., Jennings, N.R., Leung, H.-F.: On agent-mediated electronic commerce. IEEE Trans. on Know. and D. Eng. 15(4), 985–1003 (2003)
12. Herssens, C., Jureta, I., Faulkner, S.: Capturing and using qos relationships to improve service selection. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 312–327. Springer, Heidelberg (2008)
13. International Business Machines (IBM). Web service level agreement (wsla) language specification (2003)
14. In, H.P., Kim, C., Yau, S.S.: Q-mar: An adaptive qos management model for situation-aware middleware. In: Yang, L.T., Guo, M., Gao, G.R., Jha, N.K. (eds.) EUC 2004. LNCS, vol. 3207, pp. 972–981. Springer, Heidelberg (2004)

15. Jevons, W.S.: Theory of Utility. In: The Theory of Political Economy (1965)
16. Kaminski, H., Perry, M.: Employing Intelligent Agents to Automate SLA Creation. In: Emerging Web Services Technology, pp. 33–46. Springer, Heidelberg (2007)
17. Keller, A., Ludwig, H.: The wsla framework: Specifying and monitoring service level agreements for web services. J. Netw. Syst. Manage. 11(1), 57–81 (2003)
18. Kollingbaum, M.: Norm-governed Practical Reasoning Agents. PhD thesis (2005)
19. Kollingbaum, M.J., Norman, T.J.: Supervised interaction - a form of contract management to create trust between agents. In: Falcone, R., Barber, S.K., Korba, L., Singh, M.P. (eds.) AAMAS 2002. LNCS (LNAI), vol. 2631, pp. 108–122. Springer, Heidelberg (2003)
20. Lockemann, P., Nimis, J., Braubach, L., Pokahr, A., Lamersdorf, W.: Architectural Design. In: Multiagent Engineering, pp. 405–429. Springer, Heidelberg (2006)
21. Menascé, D.A.: Qos issues in web services. IEEE Intern. Comp. 6(6), 72–75 (2002)
22. Morgan, G., Parkin, S., Molina-Jimenez, C., Skene, J.: Monitoring Middleware for Service Level Agreements in Heterogeneous Environments. In: Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government, pp. 79–93. Springer, Heidelberg (2005)
23. Mui, L., Mohtashemi, M., Halberstadt, A.: Notions of reputation in multi-agents systems: a review. In: AAMAS '02, pp. 280–287 (2002)
24. Paschke, A., Dietrich, J., Kuhla, K.: A logic based sla management framework. In: SWPC '05: Semantic Web Policy Workshop at ISWC '05 (2005)
25. Pitt, J., Venkataram, P., Mamdani, A.: Qos management in manets using norm-governed agent societies. In: Dikenelli, O., Gleizes, M.-P., Ricci, A. (eds.) ESAW 2005. LNCS (LNAI), vol. 3963, pp. 221–240. Springer, Heidelberg (2006)
26. Sahai, A., Machiraju, V., Sayal, M., Jin, L.J., Casati, F.: Automated sla monitoring for web services. In: IEEE/IFIP DSOM, pp. 28–41 (2002)
27. Sierra, C., Dignum, F.: Agent-mediated electronic commerce: Scientific and technological roadmap. In: Sierra, C., Dignum, F.P.M. (eds.) AgentLink 2000. LNCS (LNAI), vol. 1991, pp. 1–18. Springer, Heidelberg (2001)
28. Tosic, V., Patel, K., Pagurek, B.: Wsol - web service offerings language. In: Bussler, C.J., McIlraith, S.A., Orlowska, M.E., Pernici, B., Yang, J. (eds.) CAiSE 2002 and WES 2002. LNCS, vol. 2512, pp. 57–67. Springer, Heidelberg (2002)
29. Trzec, K., Huljenic, D.: Intelligent agents for qos management. In: AAMAS '02, pp. 1405–1412 (2002)
30. Yan, J., Kowalczyk, R., Lin, J., Chhetri, M.B., Goh, S.K., Zhang, J.: Autonomous service level agreement negotiation for service composition provision. Future Gener. Comput. Syst. 23(6), 748–759 (2007)