Proactive SLA Negotiation for Service Based Systems

Khaled Mahbub
School of Informatics
City University London, UK
K.Mahbub@soi.city.ac.uk

George Spanoudakis
School of Informatics
City University London, UK
G.Spanoudakis @soi.city.ac.uk

***Abstract*— In this paper we propose a framework for proactive SLA negotiation that integrates this process with dynamic service discovery and, hence, can provide integrated runtime support for both these key activities which are necessary in order to achieve the runtime operation of service based systems with minimised interruptions. More specifically, our framework discovers candidate constituent services for a composite service, establishes an agreed but not enforced SLA and a period during which this pre-agreement can be activated should this become necessary**

Keywords-Service discovery, Service level agreements, Proactive SLA negotiation, service monitoring

I. INTRODUCTION

A service level agreement (SLA) is an explicit contract between the provider and the consumers of a service that defines the quality and, sometimes, functional properties which should be guaranteed during the provision of the service, as well as the penalties that should be applied in case of defaulting [7][10][11]. An SLA is set through a negotiation process between the provider and the consumer of a service [4][12]. This process is particularly complex in the case of composite services since, in order to ensure that the provision of a composite service S is in line with the SLAs required by its clients, the provider of S should also negotiate and establish subordinate SLAs with the providers of the constituent services of S. Furthermore, when a constituent service of S becomes unavailable at runtime or fails to perform according to its SLA, the provider of S should be able to discover alternative replacement services for it and negotiate SLAs with them at runtime.

As it has been suggested in [15], to minimise the runtime interruption in the provision of composite services, the discovery of back up replacement services for their constituents should be proactive, i.e., it should be performed before a constituent service of S becomes unavailable or fails to perform according to its established SLA. Proactiveness is important since service discovery is a time consuming activity and, therefore, carrying it in a reactive mode, is likely to cause significant interruption in the provision of the composite service and violations of its own SLAs. SLA negotiation should also be proactive as it will be necessary to have adequate SLAs for the potential replacement services that have been identified by proactive discovery attempting SLA negotiation just prior to binding to an alternative service is likely to cause significant delay.

Existing work on service level agreements has focused on SLA specification [13][14], negotiation [4][6] and monitoring [8]. The need for runtime SLA negotiation or re-negotiation has been realised in [2][3][5][9], where either the terms of an SLA are revised to accept a constituent service from an existing provider [2][5] or a new SLA is negotiated with a new service provider and an existing SLA is terminated [3]. All these approaches, however, are reactive as they support corrective actions only after an SLA has been violated. Thus they can fail to guarantee uninterrupted runtime provision of composite services.

To address this shortcoming, in this paper we introduce an approach for proactive runtime SLA negotiation. Our approach is based on an extension of a tool for proactive runtime service discovery which is described in [15]. Our approach weaves SLA negotiation into runtime service discovery and provides a clear process model for carrying these two activities in a coordinated manner. It also leverages upon the language for expressing runtime service discovery queries that has been developed in [15] and extends in order to enable the specification of SLA negotiation criteria. Thus, it provides integrated runtime support for both proactive service discovery and SLA negotiation which is necessary in order to achieve runtime composite service provision with minimised interruptions.

Proactive SLA negotiation is weaved into the discovery process and is performed after the execution of service discovery queries to ensure that adequate SLAs can be set for the discovered services. The objective of proactive negotiation is to establish an agreed but not enforced SLA and a period during which the consumer of the service will be able to activate the pre-agreement should this become necessary. Following this, a discovered service can be considered as a candidate constituent service for a composite service S. The negotiation process is also repeated when a pre-agreed SLA comes close to expiry and, therefore, it has to be renegotiated.

The rest of this paper is structured as follows. In Section II, we discuss the architecture of the framework for integrated proactive runtime service discovery and SLA negotiation. In Section III, we describe the negotiation process. In Section IV, we provide an overview of the language for specifying the rules for triggering and carrying out the SLA negotiation process (SLA triggering and SLA negotiation rules). In Section V, we review related work and finally in Section VI, we provide some concluding remarks and outline directions for future work.

II. Overview of Proactive Service Discovery and SLA Negotiation Framework

The architecture of our integrated service discovery and SLA negotiation framework is shown in Figure 1. According to the figure, the framework consists of a runtime service discovery tool, a service listener, an SLA negotiation broker and a monitor. It also interacts with external service registries and event captors.

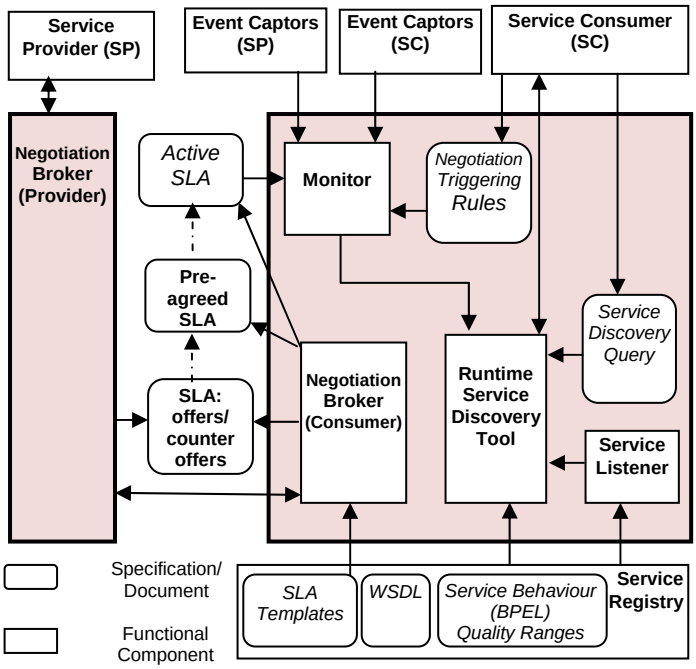

Figure 1. Architecture for proactive (and reactive) SLA negotiation

The *runtime service discovery tool* is used to identify potential alternative services for the services that a composite service uses currently. The discovery process is driven by *service discovery queries*. These queries are associated with each of the constituent services S_c of the composite service S and specify the conditions that should be satisfied by any service that could replace them in the composition. These conditions can refer to the structural (interface), behavioural, contextual, and quality characteristics that services should have in order to be acceptable replacements for S_c and, therefore, provide the criteria for discovering candidate constituent services for S_c. Service discovery queries can be executed in two modes: (a) in a reactive mode where the query is executed when the constituent service S_c it is associated with becomes unavailable or fails to satisfy an agreed SLA and, therefore, a replacement service should be identified for it, or (b) in a proactive mode where the query is executed in parallel with the operation of the composite service S in order to discover and maintain a set of candidate replacement services for it. In the proactive execution mode, the query is executed initially to build a replacement set of services for S (*RS*) and then anytime when an event indicating that the description of some service in *RS* has been changed or a new service that could be a candidate for inclusion in *RS* has emerged.

The *negotiation broker* is the component that manages and executes the negotiation process on behalf of a service consumer (i.e., the composite service) or a service provider. Our architecture assumes that a separate instance of this component is associated with each of the two sides (the service provider and consumer) which participate in the negotiation process. Negotiation brokers are responsible for negotiating and agreeing the guarantee terms of an SLA. The negotiation process can be either reactive or proactive. In proactive negotiation, the negotiation process is carried out according to a two-phase protocol that may result in a provisionally agreed SLA but not activated SLA (see *Pre-agreed SLA* in Figure 1) or negotiation failure. In reactive negotiation, the negotiation process is executed according to a single phase protocol that can result in an agreed and activated SLA (see *Active SLA* in Figure 1) or negotiation failure. In the framework, a *pre-agreed SLA* describes a service level agreement that has been reached but not activated yet. Pre-agreed SLAs have an expiry period within which they will have to be activated or cease to exist. A pre-agreed SLA becomes an *Active SLA*, if the consumer of the service decides to activate it.

The *service registry* contains descriptions of services. These should include at least a specification of the interface of the service (WSDL) and *SLA templates* indicating the terms (e.g. service quality levels, costs etc) under which the provider of service is typically willing to provide it. Additional types of service descriptions that are supported by the framework are models of service behavior expressed in BPEL and further quality characteristics that might not be included in existing SLA templates or complement these templates by specifying the entire range of values for a given characteristics as opposed to the individual quality level points or the sub-ranges of it which might be specified in SLA templates.

The *service listener* polls service registries regularly to identify changes in existing service descriptions or new services that might have become available.

The *monitor* in the architecture of Figure 1 is responsible for monitoring the provision of a service by a given provider and the use of it by a set of service consumer. In general there are two monitors: one associated with the service provider and another associated with the service consumer[1]. A monitor at either of these two sides is typically used to detect if the SLA guarantee terms which should apply to the provision of the service are satisfied, and whether the conditions of the negotiation triggering rules of the relevant party are satisfied in order to generate signals for triggering negotiation (whether proactive or reactive). A

[1] It is, however, also possible that the monitors of two parties of an SLA are realised by a same monitoring service which may be offered by a trusted external third party. Such a shared monitoring service would in general be monitoring different sets of rules for each of the involved parties and based on different sets of events.

monitor at the side of the provider may also check the levels of service usage by the relevant consumer as the latter may be preconditions of SLA guarantee terms (e.g., a service provider may have agreed to an average service throughput only if the rate of service calls by a particular provider does not exceed a given threshold).

If a monitor detects (or forecasts) that the conditions of negotiation triggering rules in the negotiation policy of a service provider or consumer are (or will be) violated, it informs the relevant negotiation broker to initiate a negotiation or renegotiation.

The checks performed by the monitors take into account events that are intercepted during the use of services (e.g. service invocations and responses, server loads). These events are intercepted and notified to the framework by different types of *event captors* that may be associated with different services (e.g. SOAP message captors). These events are notified to the monitor for verifying the adherence of services to different SLA guarantee terms and checking whether some SLA negotiation activity should be initiated.

Negotiation Triggering Rules determine the circumstances under which the negotiation of new service level agreements should start (e.g., when a provisionally agreed SLA is about to expire). Separate sets of such rules may be specified by service providers and consumers for this purpose. The negotiation triggering rules are monitored once an SLA is established.

III. SLA NEGOTIATION PROCESS

Figure 2 presents the service discovery process of the framework with the activity of SLA negotiation embedded within it. The integrated discovery and negotiation process is specified as a UML activity diagram. The process starts with the submission of a service discovery query by the composite service (i.e., the consumer of constituent services). As discussed in Section 2, this query can specify different service discovery criteria, namely: (a) structural criteria describing the interface of required services, (b) behavioural criteria describing the functionality of required services, and (c) constraints describing quality characteristics of required service. The initial execution of the service discovery query (see the action state *Execute Query* in Figure 2) results in a list of potential candidate services (*RS*). The candidate services are identified by evaluating the structural, behavioural and quality characteristics specified in a query against the structural, behavioural and quality of service specifications in service registries. The execution of the discovery query also computes distances between a query and candidate services based on the query criteria and ranks the candidate services based on their distances to the query. The list of potential candidate services is updated by executing the service discovery query when the framework is informed via the service listener that a new service has been modified (see the signal accept state *New/Amended Service Description* in Figure 2). This ensures that new or updated services are considered by the process.

Once an initial set of candidate services has been built or updated (see the action state *Create/Update Candidate Service Set*), the framework selects a service that does not have a negotiated SLA from *RS* for negotiation (see the transition guarded by the condition *Exists Service in RS without Negotiated SLA*).

In the negotiation phase (i.e., the action state *Negotiate SLA*), the desired level of service is negotiated with the selected candidate service. In this phase, the *QoS* characteristics of each candidate service are negotiated in order to achieve the best possible SLA for the services that is within the boundary constraints of the two parties. Negotiation during this phase may fail and, if this happens, for a selected candidate service then the service is removed from *RS* and a new negotiation will start with another candidate service in *RS* which does not have a pre-agreed SLA. If the negotiation with a selected service succeeds, however, a provisional SLA is established and the selected candidate service in *RS* is updated to flag the existence of the pre-agreed SLA.

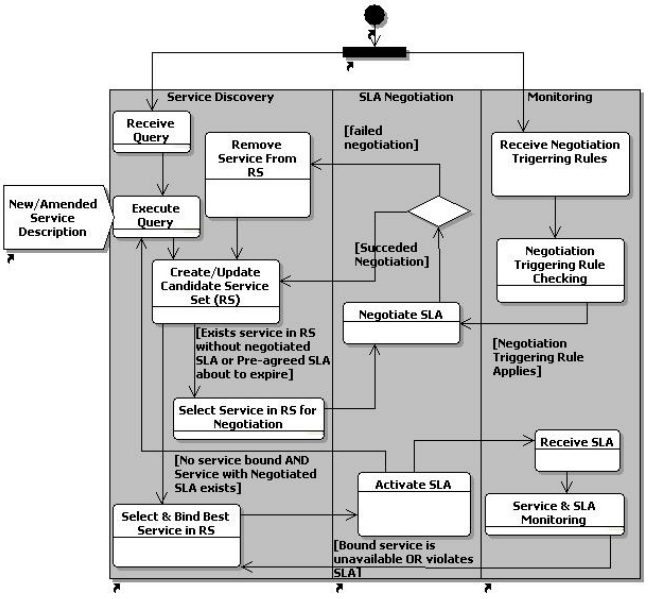

Figure 2. SLA negotiation process

It should be noted that, the negotiated SLAs for the services in *RS* do not come into force immediately. For each pre-agreed SLA, the negotiation process establishes a time period over which the pre-agreed SLA can be automatically brought into force without further negotiation. This will happen if the relevant service is selected for binding to the composite service. If the validity period of a pre-agreed SLA comes close to expiry without the candidate service being bound to the composite service, the framework will proactively re-negotiate the SLA (see the transition guarded by the condition *Pre-agreed SLA about to expire*, from the action state *Create/Update Candidate Service Set* to the

action state *Select Service RS for Negotiation*). The remaining validity period threshold that determines when a pre-agreed SLA should be negotiated is selected by the composite service provider.

Following the selection of a service in *RS* for binding at runtime, its SLA is automatically enforced (see the action state *Activate SLA* in Figure 2). When an SLA comes into force, its guarantee terms become subject of monitoring (see the action states *Receive SLA* and *Service & SLA Monitoring* in Figure 2*)*. If the monitoring process detects violation of the SLA or the deployed service becomes unavailable then the service is replaced by the best available service in *RS* (see the transition from the action state *Service & SLA Monitoring* to the action state *Select & Bind Best Service in RS*). The detection of violation of the conditions in negotiation triggering rules (e.g. active SLA about to expire) triggers the negotiation phase to establish a new SLA.

The relationship between the quality criteria expressed in a discovery query and the quality preferences expressed in the negotiation rules is exemplified in Figure 3. The figure shows the case where the discovery and negotiation activity take into account two quality criteria, namely Q1 and Q2 where the service consumer (composite service) seeks to maximise the value of Q1 (e.g., service performance) and minimise the value of Q2 (e.g., service cost). The dotted lines *q1* and *q2* in the figure show the minimum acceptable value for Q1 and the maximum acceptable value for Q2 that the service consumer sets, respectively. These two boundary lines should be expressed by quality constraints in the service discovery query to ensure that no service which does not satisfy them will be considered any further and could participate in a negotiation process that is known to fail (such services cannot become members of the set *RS* in Figure 2).

The zic-zac line in Figure 3 shows the preferences expressed by the negotiation rules of the service consumer. The figure shows a typical case where the consumer is prepared to accept increases in Q2 in exchange of increases in Q1 but cannot agree on any Q2 value that is higher than q2 and any Q1 value that is lower than q1. Thus, the negotiation process will only try to agree an SLA with Q1 and Q2 values in the shaded region of the figure. In our framework we assume a multiphase negotiation protocol where participants are allowed to generate counter offers in response to a given offer until an acceptable goal is reached [13].

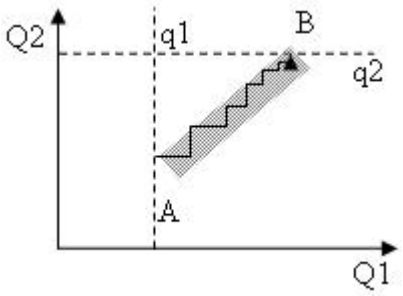

Figure 3. Negotiation rules and query criteria

IV. Specification Languages

In this section we describe the languages that are used in our framework to express (i) Service Discovery Query, (ii) Service Level Agreement, (iii) Negotiation Rules and (iv) Negotiation Triggering Rules.

A. Service Discovery Query Language

Service discovery queries are specified in an XML language introduced in [15]. Figure 4 shows an example of a service discover query expressed in this language. As shown in the figure, a query contains a *structuralQuery*, *BehaviourQuery* and one or more *Constraints*. The *StructuralQuery* specifies the required interface of candidate services. This interface is specified in WSDL. The *BehaviourQuery* specifies the required behavioural characteristics of a service. These characteristics are expressed in temporal logic language which allows the specifications of conditions about: (a) the existence of certain operations in a service specification; (b) the order in which these operations should be executed by a service; (c) other dependencies between operations; (d) pre-conditions; and (e) loops concerning execution of certain operations. As they are not related to the negotiation process, the structural and behavioural parts of a query are not further exemplified in Figure 4. Examples of such parts are, however, given in [15].

```
<dqns:ServiceQuery xmlns:dqns=
  "http://scube.eu/schema/DiscoveryQuery"
  xmlns:slac="http://scube.eu/schema/Constraint"
  xmlns:bqns="http://scube.eu/schema/Behavour_SQL"
   queryID="Q1">
 <dqns:StructuralQuery><!–WSDL->
 </dqns:StructuralQuery>
 <bqns:BehaviourQuery>..</bqns:BehaviourQuery>
 <slac:Constraint>
   <slac:LogicalExpression>
     <slac:Condition relation="GREATER-THAN">
        <slac:Arg1><slac:QualityAttribute
                        name="AVAILABILITY"/>
        </slac:Arg1>
        <slac:Arg2><slac:Constant type="NUMERICAL"
                    unit="PC">75</slac:Constant>
        </slac:Arg2>
      </slac:Condition>
      <slac:LogicalOperator>AND
      </slac:LogicalOperator>
      <slac:Condition relation="LESS-THAN">
        <slac:Arg1><slac:QualityAttribute
                    name="RESPONSE_TIME"/>
        </slac:Arg1>
        <slac:Arg2><slac:Constant type="NUMERICAL"
                    unit="MS">10</slac:Constant>
        </slac:Arg2>
      </slac:Condition>
    </slac:LogicalExpression>
  </slac:Constraint>
</dqns:ServiceQuery>
```

Figure 4. Example Service Discovery Query

The *Constraint* part of a discovery query comprises a set of constraints specifying the required *QoS* characteristics of a service. The example query of Figure 4 includes a constraint expressed as a logical combination of two conditions. These are: (a) a condition stating that the availability of acceptable

```
<sla:SLAContract xmlns:sla=
   "http://scube.eu/schema/SLA_Contract"
  xmlns:slac="http://scube.eu/schema/Constraint"
  contractID="SLA-No-2" name="S-Cube-SLA"
  scope="UNDER_NEGOTIATION">
 <sla:SLATerms>
   <sla:Actor>
    <sla:Role>PROVIDER</sla:Role>
    <sla:Type>
     <sla:Company name="XYZ"
       contactInformation="Street_Address">
     </sla:Company>
    </sla:Type>
    <sla:NegotiationStrategy>
              MULTI-PHASE_MULTI-ISSUE
    </sla:NegotiationStrategy>
    </sla:Actor>
    <slac:Constraint>
     <slac:LogicalExpression>
      <slac:Condition relation="EQUAL-TO">
        <slac:Arg1><slac:QualityAttribute
                    name="AVAILABILITY"/>
        </slac:Arg1>
        <slac:Arg2>
         <slac:Constant type="NUMERICAL"
                  unit="PC">80</slac:Constant>
        </slac:Arg2>
       </slac:Condition>
      </slac:LogicalExpression>
      <slac:LogicalOperator>AND
      </slac:LogicalOperator>
      <slac:LogicalExpression>
       <slac:Condition relation="EQUAL-TO">
         <slac:Arg1><slac:QualityAttribute
                    name="RESPONSE_TIME" />
         </slac:Arg1>
         <slac:Arg2>
          <slac:Constant type="NUMERICAL"
                    unit="MS">9</slac:Constant>
         </slac:Arg2>
        </slac:Condition>
      </slac:LogicalExpression>
    </slac:Constraint>
 </sla:SLATerms>
```

```
<sla:SLATerms>
  <sla:Actor>
    <sla:Role>CONSUMER</sla:Role>
    <sla:Type>
      <sla:Company name="City"
          contactInformation="Northampton_Sqr">
      </sla:Company>
     </sla:Type>
     <sla:NegotiationStrategy>
          MULTI-PHASE_MULTI-ISSUE
     </sla:NegotiationStrategy>

   </sla:Actor>
   <slac:Constraint>
    <slac:LogicalExpression>
      <slac:Condition relation="GREATER-THAN">
       <slac:Arg1><slac:QualityAttribute
                        name="AVAILABILITY"/>
       </slac:Arg1>
       <slac:Arg2>
         <slac:Constant type="NUMERICAL"
                  unit="PC">90</slac:Constant>
       </slac:Arg2>
      </slac:Condition>
    </slac:LogicalExpression>
    <slac:LogicalOperator>AND
    </slac:LogicalOperator>
    <slac:LogicalExpression>
      <slac:Condition relation="LESS-THAN">
        <slac:Arg1><slac:QualityAttribute
                  name="RESPONSE_TIME" />
        </slac:Arg1>
        <slac:Arg2>
           <slac:Constant type="NUMERICAL"
                    unit="MS">8</slac:Constant>
        </slac:Arg2>
       </slac:Condition>
      </slac:LogicalExpression>
  </slac:Constraint>
 </sla:SLATerms>
 <sla:Penalty>... . . .</sla:Penalty>
</sla:SLAContract>
```

Figure 5. Example SLA

services should be greater than 75% and (b) a condition stating that the response time of acceptable should be less than 10ms. These constraints, as discussed in Sect. III will be used to find suitable candidate services during the discovery process and set the bottom line for the required *QoS* characteristics of services during the negotiation process.

B. *Service Level Agreement Specification*

The Service Level Agreements (SLA) in our framework are expressed in an XML language where the foundations of this language are discussed in [14]15] and the schema which defines the SLA specification language for our framework is available in [16]. This language allows to specify not only final SLAs after an agreement has been reached but also SLA offers (and counter offers) that different parties may create during the negotiation process.

Figure 5 shows an example including an SLA offer and an SLA counter-offer made by two actors during an SLA negotiation process. The attribute *scope* in the *SLAContract* element signifies the status (e.g. under negotiation, pre-agreed SLA or active SLA) of the SLA. As shown in the figure, the SLA contract contains two sets of SLA terms (i.e., constraints over *QoS* attributes). Each set of SLA terms is proposed by a participating actor in the negotiation process. In the example, the first set of SLA terms is proposed by the actor "XYZ" (see the attribute *name* of the sub-element *Company* of the element *Actor*). This actor has the role of a service provider in our example as indicated by the element *Role* in the specification of XYZ.

An actor can also specify its own *negotiation strategy*, i.e., the negotiation protocol that it will use to govern the negotiation process and the communication with the other party during it. Whilst the details of the negotiation strategy are hidden from the other participant, information of the

overall protocol that an actor will use should be revealed in order for the two parties to be able to establish whether they are using compatible protocols and it is, therefore, worth engaging in the negotiation process. For the actor XYZ in the example of Figure 5, the negotiation strategy is specified as *MULTI-PHASE_MULTI-ISSUE*. This strategy indicates that XYZ will consider, in principle, counter offers in response to a given offer that it has made until an acceptable goal is reached (*MULTI-PHASE*) and that more than one issue can be the subject of negotiation (*MULTI-ISSUE*).

The SLA required by XYZ in Figure 5 is specified as a logical combination of two conditions. The first of these conditions states that availability of the service offered by the actor is 80%. The second condition states that the response time of the service offered by the actor is 9 milliseconds. Based on this offer, it is clear that XYZ fulfills the boundary conditions of the discovery query of Figure 4 (i.e., AVAILABILITY > 75% and RESPONSE_TIME < 10ms), and it could, therefore, become party to negotiation process where the offer and counter offer of Figure 5 could be generated.

The second set of SLA terms in the example of Figure 5 is proposed by an actor, called “City” which has the *role* of a service *consumer*. Hence, *City* is the service consumer in our example. Furthermore, the service consumer specifies its quality requirements in terms of a set of constraints where each constraint in the set signifies a desired SLA guarantee term.

In this example, the service consumer specifies a constraint that is a logical combination of two conditions: (a) a condition stating that availability should be greater than 90% and (b) a condition stating that the response time should be less than 8ms. It should be noted, that the requestor, in this example has made counter offer for the attributes availability and response time, in response to the offers made by the service provider.

During the negotiation process the SLA contains multiple sets of SLA terms where each set is proposed by a participating actor in the negotiation process. This facilitates a participant in the negotiation process to consider all the offers made by all the participants without storing the offers in local storage. However, after a successful negotiation when an agreement is reached the SLA contains only one set of SLA terms that includes the list of participants that agreed to the constraints, as well as the penalties that will apply if the SLA is violated and the time validity of the agreed (pre-agreed SLA).

C. *Specification of Negotiation Rules and Negotiation Triggering Rules*

In our framework, negotiation rules and negotiation triggering rules are specified in an XML language. The schema of this language can be found in [16]. A negotiation rule in this language has the generic structure:

IF (condition) THEN (action) ELSE (action)

Conditions in these rules are expressed as atomic conditions over quality attributes of services or logical combinations of atomic conditions. Rule actions can be of three types: (i) *accept* actions which enable the acceptance of the value of one or more attributes in a given SLA offer, (ii) *reject* actions which enable the rejection of the value of one or more QoS attributes in a given SLA offer, and (iii) *set* actions which allow to define a new value or range of values for one or more QoS attribute.

```
<tnsr:NegotiationRule>
 <tnsr:If>
  <tnsr:LogicalExpression>
   <slac:Condition relation="EQUAL-TO">
    <slac:Arg1>
      <slac:QualityAttribute name="AVAILABILITY"
             party="PROVIDER"/>
    </slac:Arg1>
    <slac:Arg2>
      <slac:Constant type="NUMERICAL">90
      </slac:Constant>
    </slac:Arg2>
   </slac:Condition>
  <slac:LogicalOperator>AND</slac:LogicalOperator>
   <slac:Condition relation="LESS-THAN">
    <slac:Arg1>
      <slac:QualityAttribute name="PRICE"
                          party="PROVIDER"/>
    </slac:Arg1>
    <slac:Arg2>
      <slac:ArithmeticExpression>
       <slac:ArithmeticOperand>
          <slac:QualityAttribute name="PRICE"
                       party="CONSUMER"/>
      </slac:ArithmeticOperand>
      <slac:ArithmeticOperator>MULTIPLY
      </slac:ArithmeticOperator>
      <slac:ArithmeticOperand>
        <slac:Constant type="NUMERICAL">0.5
         </slac:Constant>
      </slac:ArithmeticOperand>
     </slac:ArithmeticExpression>
    </slac:Arg2>
   </slac:Condition>
  </tnsr:LogicalExpression>
 </tnsr:If>
 <tnsr:Then>
  <tnsr:Action>
    <tnsr:Accept>
      <tnsr:QualityAttribute name="AVAILABILITY"
                  party ="PROVIDER"/>
      <tnsr:QualityAttribute name="PRICE"
                  party ="PROVIDER" />
     </tnsr:Accept>
    </tnsr:Action>
 </tnsr:Then>
</tnsr:NegotiationRule>
```

Figure 6. Example Negotiation Rule

An example of a negotiation rule is shown in Figure 6. This example expresses a rule used by a service consumer. The rule states that if the service availability offered by a provider is 90% and the offered service price is half of the consumer's expected price then the offer should be accepted.

V. Related Works

An agent based framework for SLA management is presented in [9]. In this framework an initiator agent from

the service consumer's side and a responder agent from the service provider's side take part in the negotiation process. The responder agent advertises the service level capabilities and the initiator agent fetches these advertisements and initializes the SLA negotiation process. Different stages of SLA life cycle e.g. formation, enforcement and recovery is performed through the autonomous interactions among theses agents. In the case of an SLA violation, the initiator agent may either claim compensation and renegotiate with the service provider or select a new service provider. Provision of compensation in case of violation of SLA is also argued in [1]. This approach claims that the penalty clauses in the SLA should not only specify the monetary penalties or impact on potential future agreements between the parties; rather the penalty clauses should include several other issues such as which countries laws will be applied in case a conflict between the provider and the client arise, the impact of the penalty clauses on the choice of service level objectives.

Runtime renegotiation is suggested in [4, 7, 5, 2, 3] to manage SLA violations. In [2] service level objectives are revised and renegotiated at runtime and the deployed service is adjusted to the newly agreed service level objectives. A similar approach which allows changing service level objectives whilst keeping the existing SLA is described in [5]. In [3] a renegotiation protocol is described that allows the service consumer or service provider to initiate renegotiation while the existing SLA is still in forced. In this protocol either party may initiate the renegotiation due to the changes in the business requirements and after a successful renegotiation the existing SLA is superseded by a new contract.

All of these approaches are reactive in nature, i.e. renegotiation starts only after an existing SLA is violated. The outcome of renegotiation is either a revised set of service level objectives allowing the acceptance of a service from an existing provider or a new SLA for a new service provider terminating the existing SLA. All these approaches either affect the quality of the delivered service or fail to guarantee uninterrupted service. Our proposed framework integrates SLA negotiation with dynamic service discovery and, hence, can provide integrated runtime support for both these key activities which are necessary in order to achieve the runtime operation of service based applications with minimised interruptions.

VI. Conclusion and Future Work

This paper proposes a framework that integrates service discovery and monitoring of service in order to facilitate proactive SLA negotiation. The service discovery process is used by service consumers (i.e., composite services and/or service based applications) in order to identify potential alternative services for the services that they currently use. The identification of alternative services is based on various characteristics of published services including structural, behavioural and QoS characteristics.

The framework negotiates with each alternative service to reach an agreement over the QoS level of the service. The negotiation process is carried out according to a two-phase protocol that may result in a provisionally agreed SLA but not activated SLA or negotiation failure. A provisional SLA is a service level agreement that has been agreed by a service provider and a service consumer but has not been activated yet. Such an SLA has an expiry date by which it will either be activated or cease to exist. The monitoring process monitors the runtime behaviour of the service provider and the service requester in order to detect if the agreed SLA is satisfied. If the monitor detects violations of the agreed SLA then the service is replaced by an alternative service. Or if the monitor detects situations that require renegotiation of the SLA then the monitor triggers the framework to initiate renegotiation.

The presented framework has also opened broad scope of future investigations. For example the framework can be extended to support proactive negotiation for hierarchical SLA i.e. a complex SLA can be decomposed into several SLAs and negotiated separately to come to a final agreement. Also in the presented framework, negotiation rules are specified by the participating parties before the negotiation starts and followed in the negotiation process. The framework can be extended to support dynamic adaptation of the negotiation rules, i.e. the participants will be able to dynamically change the negotiation rules during the negotiation process.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under Grant Agreement 215483 (S-Cube).

References

[1] Omer Rana, Martin Warnier, Thomas B. Quillinan, Fances Brazier and Dana Cojocarasu, "Managing Violations in Service Level Agreements", the Proceedings of the Usage of Service Level Agreements in Grids Workshop, 2007

[2] Di Modica G.,Tomarhio O. and Lorenzo V., "A framework for the management of dynamic SLAs in composite service scenarios", Service-Oriented Computing - ICSOC 2007 Workshops: ICSOC 2007, International Workshops, Vienna, Austria, September 17, 2007

[3] Michael Parkin, Peer Hasselmeyer, Bastian Koller, Philipp Wieder: An SLA Re-negotiation Protocol. In proceedings of the 2nd Workshop on Non Functional Properties and Service Level Agreements in Service Oriented Computing at ECOWS 2008. Dublin, Ireland. 12 November 2008.

[4] Waeldrich, O., Ziegler, W., "A WS-Agreement based Negotiation Protocol", Technical Report, Fraunhofer Institute SCAI, VIOLA - Vertically Integrated Optical Testbed for Large Application in DFN. 2006

[5] Rizos Sakellariou and Viktor Yarmolenko, "On the Flexibility of WS-Agreement for Job Submission", Proceedings of the 3rd International Workshop on Middleware for Grid Computing MGC '05, vol. 117, 1-6 (Nov. 2005)

[6] Catalin L. Dumitrescu and Ian Foster, "GRUBER: A Grid Resource Usage SLA Broker", International Euro-Par conference, Lisbon , PORTUGAL (30/08/2005)

[7] Philipp Wieder, Jan Seidel, Oliver Wäldrich, Wolfgang Ziegler, and Ramin Yahyapour, "Using SLA for Resource Management and

Scheduling - A Survey", Book Chapter, Grid Middleware and Services Challenges and Solutions, Springer, 2008

[8] Raimondi, F. and Skene, J. and Chen, L. and Emmerich, W., "Efficient monitoring of web service SLAs", Technical report. Research Notes (RN/07/01). UCL, London, UK. 2007

[9] Q. He, J. Yan, R. Kowalczyk, H. Jin, Y. Yang, "Lifetime Service Level Agreement Management with Autonomous Agents for Services Provision". Information Sciences, Elsevier, 2009

[10] Peer Hasselmeyer, Changtao Qu, Lutz Schubert, Bastian Koller, and Philipp Wieder. Towards Autonomous Brokered SLA Negotiation. In: Proceedings of the eChallenges e-2006 Conference, Barcelona, Spain, October 2006.

[11] Paul Karaenke and Stefan Kirn, "Service Level Agreements: Evaluation from a Business Application Perspective", Proceedings of eChallenges 2007

[12] Pichot, A.; Wieder, P.; Ziegler, W.; Wäldrich, O. "Dynamic SLA-negotiation based on WS-Agreement", CoreGRID - Network of Excellence, 2007, CoreGrid Technical Report; 82, TR-0082

[13] V. Robu, D.J.A. Somefun, and J. A. La Poutre. "Modeling complex multi-issue negotiations using utility graphs", In Proc. of the 4th Int. Conf. on Autonomous Agents & Multi Agent Systems (AAMAS'05), Utrecht, ACM Press, 2005

[14] Kyriakos Kritikos and Barbara Pernici, editors. "Initial Concepts for Specifying End-to-End Quality Characteristics and Negotiating SLAs". S-Cube project deliverable, June 2009. S-Cube project deliverable: CD-JRA-1.3.3. http://www.s-cube-network.eu/achievements-results/s-cube-deliverables.

[15] A. Zisman, G. Spanoudakis, and J. Dooley. A Framework for Dynamic Service Discovery, 23rd Int. IEEE/ACM Conf. on Automated Software Engineering, 2008.

[16] SLA Specifications: http://www.soi.city.ac.uk/~am697/sla/SLA-Specification.zip