Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 189–204. http://www.scpe.org
ISSN 1895-1767
© 2010 SCPE

VIESLAF FRAMEWORK: FACILITATING NEGOTIATIONS IN CLOUDS BY APPLYING SERVICE MEDIATION AND NEGOTIATION BOOTSTRAPPING

IVONA BRANDIC[*], DEJAN MUSIC[*], AND SCHAHRAM DUSTDAR[*]

Abstract. Cloud computing represents a novel and promising computing paradigm where computing resources have to be allocated to software for their execution. Self-manageable Cloud infrastructures are required in order to achieve that level of flexibility on one hand, and to comply to users' requirements specified by means of Service Level Agreements (SLAs) on the other. However, many assumptions in Cloud markets are old fashioned assuming same market conditions as for example in computational Grids. One such assumptions is that service provider and consumer have matching SLA templates and common understanding of the negotiated terms or that they provide public templates, which can be downloaded and utilized by the end users. Moreover, current Cloud negotiation systems have based themselves on common protocols and languages that are known to the participants beforehand. Matching SLA templates and a-priori knowledge about the negotiation terms and protocols between partners are unrealistic assumption in Cloud markets where participants meet on demand and on a case by case basis. In this paper we present VieSLAF, a novel framework for the specification and management of SLA mappings and meta-negotiations facilitating service mediation and negotiation bootstrapping in Clouds. Using VieSLAF users may specify, manage, and apply SLA mappings bridging the gap between non-matching SLA templates without a-priori knowledge about negotiation protocols, required security standards or negotiated terms. We exemplify two case studies where VieSLAF represents an important contribution towards the development of open and liquid Cloud markets.

Key words: grid services, cloud computing, autonomic computing, service negotiation

1. Introduction.

Service-oriented Architectures (SOA) represent a promising approach for implementing ICT systems [9, 24, 30] by packaging the software to services that can be accessed independently of the used programming languages, protocols, and platforms. Despite remarkable adoption of SOA as the key concept for the implementation of ICT systems, the full potential of SOA (e.g., dynamism, adaptivity) is still not exploited [28]. SOA approach and Web service technologies represent large scale abstractions and a candidate concept for the implementation Cloud Computing systems, where massively scalable computing is made available to end users as a service [9, 24]. The key benefits of providing computing power as a service are (a) avoidance of expensive computer systems configured to cope with peak performance, (b) pay-per-use solutions for computing cycles requested on-demand, and (c) avoidance of idle computing resources [16].

Non-functional requirements of a service execution are termed as *Quality of Service (QoS)*, and are expressed and negotiated by means of *Service Level Agreements (SLAs)*. *SLA templates* represent empty SLA documents with all required elements like parties, SLA parameters, metrics and objectives, but without QoS values [12]. However, most existing Cloud frameworks assume that the communication partner know about the negotiation protocols, required security standards and negotiated terms before entering the negotiation and that they have matching SLA templates. These assumptions rely on related technologies (like computational Grids) and cannot be transferred to computational Cloud markets. In case of computational Clouds a priori knowledge about negotiation protocols and strategies as well as matching SLA templates represent unrealistic assumption since services are discovered dynamically and on demand.

In this paper we approach the gap between existing QoS methods and Cloud services by proposing a *Vienna Service Level Agreement Framework (VieSLAF)* architecture for Cloud service management with components for *service mediation* and *negotiation bootstrapping* [7, 8]. Thereby, we introduce so-called *meta-negotiations* to allow two parties to reach an agreement on what specific negotiation protocols, security standards, and documents to use before starting the actual negotiation. Moreover, we discuss the concept of SLA-mappings to bridge the gap between inconsistent SLA templates. The concept of SLA mappings can be exemplified in differences in terminology for a common attribute such as *price*, which may be defined as *usage price* on one side and *service price* on the other, leading to inconsistencies during the negotiation process. VieSLAF framework has been successfully applied to (i) develop Cloud infrastructures for the SLA-based resource virtualization [20] by utilizing our meta negotiation approach and (ii) to facilitate liquidity management in Clouds by using our SLA mapping approach [29]. Besides performance evaluation of the VieSLAF we discuss successful case studies for the application of VieSLAF to establish open and liquid Cloud markets.

[*]Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Vienna, Austria, Emails: {ivona, dejan, dustdar}@infosys.tuwien.ac.at

The main contributions of this paper are (1) description of the scenarios for the definition of *SLA mapping* documents; (ii) development of the architecture for the *meta-negotiations* in Cloud systems; (iii) description of the *meta-negotiation document*; (iv) definition of the *VieSLAF* architecture used for the semi-automatic management of *SLA mappings* and *meta-negotiations* and (iv) demonstration of the usability of the VieSLAF framework for real-world Cloud negotiations.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 gives an overview about the goals of the adaptable, versatile, and dynamic services, in particular goals considering negotiation bootstrapping and service mediation. In Section 4 we discuss the meta-negotiation approach, whereas in Section 5 we present the SLA mapping approach. Section 6 presents the *VieSLAF* architecture. In Section 7 we evaluate SLA mapping and meta negotiation approach and report successful VieSLAF case studies. Section 8 concludes this paper and describes the future work.

2. Related Work.

Currently, a large body of work exists in the area of service negotiation and SLA-based QoS. Most of the related work can be classified into the following four categories: (1) adaptive SLA mechanisms based on OWL, DAML-S and other semantic technologies [13, 26, 38]; (2) SLA based QoS systems, which consider varying service requirements but do not consider non matching SLA templates [1, 34]; (3) systems relying on the principles of autonomic computing [3, 21, 22]; and systems addressing versatile negotiation in Grids/SOAs [25, 27, 18]. Since there is very little work on service mediation and negotiation bootstrapping in Clouds we look in particular into related systems like Grids and SOAs [4].

Work presented in [27] discusses incorporation of SLA-based resource brokering into existing Grid systems. Oldham et al. describe a framework for semantic matching of SLAs based on WSDL-S and OWL [26]. Dobson at al. present a unified quality of service (QoS) ontology applicable to the main scenarios identified such as QoS-based Web services selection, QoS monitoring and QoS adaptation [13]. Zhou et al. survey the current research on QoS and service discovery, including ontologies such as OWL-S and DAML-S. Thereafter, an ontology is proposed, DAML-QoS, which provides detailed QoS information in a DAML format [38]. Hung et al. propose an independent declarative XML language called WS-Negotiation for Web services providers and requestors. WS-Negotiation contains three parts: negotiation message, which describes the format for messages exchanged among negotiation parties, negotiation protocol, which describes the mechanism and rules that negotiation parties should follow, and negotiation decision making, which is an internal and private decision process based on a cost-benefit model or other strategies [17]. Work presented in [1] extends the service abstraction in the Open Grid Services Architecture (OGSA) for QoS properties focusing on the application layer. Thereby, a given service may indicate the QoS properties it can offer or it may search for other services based on specified QoS properties.

Quan et al. discuss the process of mapping a light communication workflow within an SLA context with different kinds of sub-jobs and resources [25]. Dan et al. present a framework for providing customers of Web services differentiated levels of service through the use of automated management and SLAs [12]. Ardagana et al. present an autonomic grid architectures with mechanisms to dynamically re-configure service center infrastructures, which is basically exploited to fulfill varying QoS requirements [3]. Koller et al. discuss autonomous QoS management using a proxy-like approach. The implementation is based on WS-Agreement [36]. Thereby, SLAs can be exploited to define certain QoS parameters that a service has to maintain during its interaction with a specific customer [21]. König at al. investigate the trust issue in electronic negotiations, dealing with trust to a potential transaction partner and selection of such partners based on their past behavior [22].

Quan et al. and Ouelhadj et al. discuss incorporation of SLA-based resource brokering into existing Grid systems [25, 27]. Li et al. discusses Rudder framework, which facilitates automatic Grid service composition based on semantic service discovery and space based computing [23]. Hill et al. discusses an architecture that allows changes to the Grid configuration to be automated in response to operator input or sensors placed throughout the Grid based on principles of autonomic computing [18]. Similarly to Hill et al. work discussed in Vambenepe et al. addresses global service management based on principles of autonomic computing [33]. Vu et al. present an extensible and customizable framework for the autonomous discovery of semantic Web services based on their QoS properties [35]. Condor's ClassAds mechanism is used to represent jobs, resources, submitters and other Condor daemons [31].

However, to the best of our knowledge none of the discussed approaches deals with user-driven and semi-automatic definition of SLA mappings enabling negotiations between inconsistent SLA templates. Also, none of the presented approaches address *meta-negotiations (MN)* where participating parties may agree on a specific negotiation protocol, security standards or other negotiation pre-requisites.

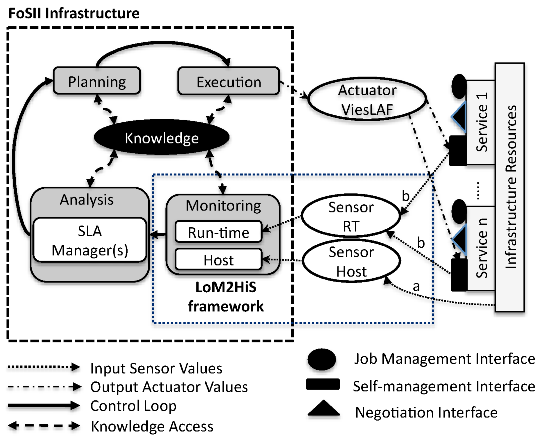


Fig. 3.1. *FoSII infrastructure*

3. Adaptable, Versatile, and Dynamic services. In this section we discuss how service mediation and negotiation bootstrapping can be realized using the concepts of autonomic computing [19]. First, we introduce the *Foundations of Self-governing ICT Infrastructures (FoSII)* project (Section 3.1). Thereafter we discuss how the *VieSLAF* architecture contributes to the implementation of *FoSII* goals (Section 3.2).

3.1. FoSII infrastructure. To facilitate dynamic, versatile, and adaptive IT infrastructures, SOA systems should react to environmental changes, software failures, and other events which may influence the systems' behavior. Therefore, adaptive systems exploiting self-* properties (self-healing, self-controlling, self-managing, etc.) are needed, where human intervention with the system is minimized. In *Foundations of Self-governing ICT Infrastructures (FoSII)* project we propose models and concepts for adaptive services, utilizing autonomic computing concepts [3, 19]. As shown in Figure 3.1 the *FoSII* infrastructure is used to manage the whole Monitoring, Analysis, Planning and Execution (MAPE) lifecycle of self-adaptable Cloud services [5]. Each FoSII service implements three interfaces: (i) negotiation interface necessary for the establishment of SLA agreements, (ii) job-management interface necessary to start the job, upload data, and similar job management actions, and (iii) the self-management interface necessary to devise actions in order to prevent SLA violations.

The self-management interface shown in Figure 3.1 is implemented by each Cloud service and specifies operations for sensing changes of the desired state and for reacting to those changes. The host monitor sensors continuously monitor the infrastructure resource metrics (input sensor values arrow a in Figure 3.1) and provide the autonomic manager with the current resource status. The run-time monitor sensors sense future SLA violation threats (input sensor values arrow b in Figure 3.1) based on resource usage experiences and predefined threat thresholds. The treat thresholds should be retrieved by the knowledge management systems as described later on in the paper. The mapping between the sensed host values and the values of the SLA parameters is described next.

As shown in Figure 3.1 we distinguish between *host monitor* and *runtime monitor.* Resources are monitored by *host monitor* using arbitrary monitoring tools (e.g. Ganglia). Thus, resources metrics include e.g., down-time, up-time, available in and out bandwidth. Based on the predefined mapping rules stored in a database monitored metrics are periodically mapped to the SLA parameters. An example SLA parameter is service availability Av, which is calculated using the resource metrics *downtime* and *uptime* and the according mapping rule looks like the following one:

$$Av = 1 - downtime/uptime \tag{3.1}$$

The mapping rules are defined by the provider using appropriate Domain Specific Languages (DSL). These rules are used to compose, aggregate, or convert the low-level metrics to form the high-level SLA parameter including mappings at different complexity levels e.g., $1 : n$ or $n : m$. Thus, calculated SLA values are compared with the predefined threat threshold in order to react before SLA violations happen. The concept of detecting future SLA violation threats is designed by defining a more restrictive threshold than the SLA violation threshold known as threat threshold. Generation of the *threat threshold* is far from trivial and should be defined and mananaged by the FoSII's knowledge management system.

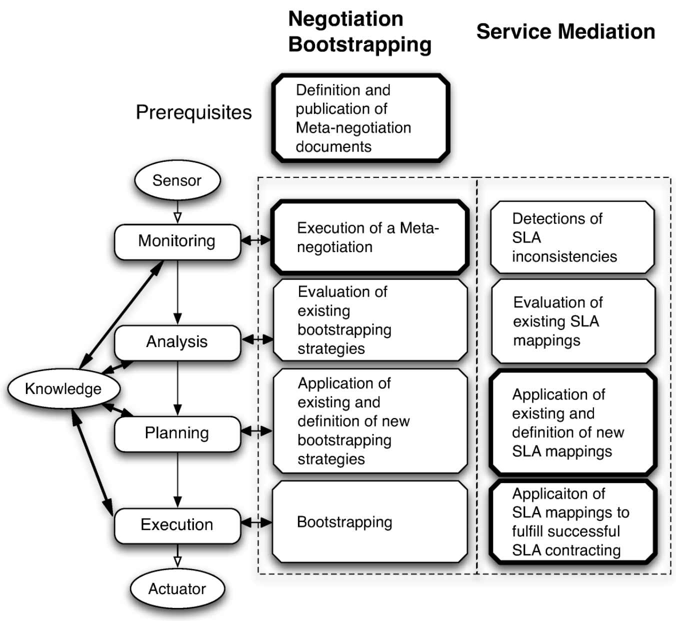

Negotiation Bootstrapping
Service Mediation
Prerequisites
Definition and publication of Meta-negotiation documents
Sensor
Monitoring
Analysis
Knowledge
Planning
Execution
Actuator
Execution of a Meta-negotiation
Evaluation of existing bootstrapping strategies
Application of existing and definition of new bootstrapping strategies
Bootstrapping
Detections of SLA inconsistencies
Evaluation of existing SLA mappings
Application of existing and definition of new SLA mappings
Applicaiton of SLA mappings to fulfill successful SLA contracting

Fig. 3.2. *Negotiation Bootstrapping and Service Mediation as Part of the Autonomic Process*

As described in [11] we implemented a highly scalable framework for mapping of Low-level Resource Metrics to High Level SLA parameters *LoM2HiS framework* facilitating exchange of large numbers of messages. We designed and implemented a communication model based on the Java Messaging Service (JMS) API, which is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients. We use Apache ActiveMQ as a JMS provider that manages the sessions and queues.

As shown in Figure 3.1 *VieSLAF* framework represents an actuator mediating between inconsistent SLA templates and bootstrapping between different protocols. In the following we discuss how the *VieSLAF* contributes to the implementation of the MAPE cycle in self-adaptable Cloud services considering service negotiation phase.

3.2. Negotiation Bootstrapping and Service Mediation. Figure 3.2 depicts how the principles of autonomic computing can be applied to negotiation bootstrapping and service mediation. As a prerequisite of the negotiation bootstrapping users have to specify meta-negotiation document describing the requirements of a negotiation, as for example required negotiation protocols, required security infrastructure, provided document specification languages, etc. During the *monitorig phase* all candidate services are detected which need negotiation bootstrapping, e.g. which do not have matching negotiation protocol with the potential consumer. During the *analysis phase* existing knowledge base is queried and potential bootstrapping strategies are found. In case of missing bootstrapping strategies users can define in a semi-automatic way new strategies (*planning phase*). Finally, during the *execution phase* the negotiation is started by utilizing appropriate bootstrapping strategies.

The same procedure can be applied to service mediation. During the service negotiation inconsistencies in SLA templates may be discovered (*monitoring phase*). During the *analysis phase* existing SLA mappings are analyzed. During the *planning phase* new SLA mappings can be defined, if existing mappings cannot be applied. Finally, during the *execution phase* the newly defined SLA mappings can be applied.

As indicated with bold borders in Figure 3.2, in this paper we present solutions for the definition and accomplishment of meta-negotiations (Section 5) and for the specification and applications of SLA mappings (Section 4) as described next.

4. Service Mediation with SLA Mappings. In the presented approach each SLA template has to be published into a registry where negotiation partners i. e., provider and consumer, can find each other. The management of SLA mappings and published services is presented in Section 4.1. The transformations between remote and local SLA templates are discussed in Section 4.2. Finally, an example SLA mapping document is presented in Section 4.3.

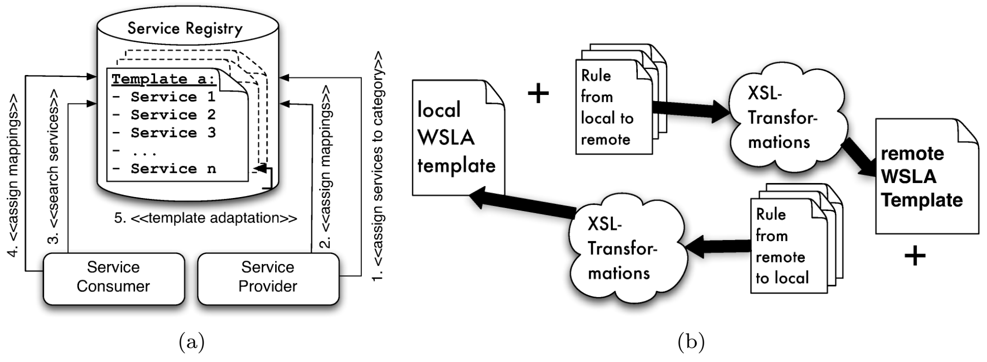


FIG. 4.1. *(a) Management of SLA-Mappings (b) QoS basic scenario*

4.1. Management of SLA Mappings. Figure 4.1(a) depicts the architecture for the management of SLA mappings and participating parties. The registry comprises different *SLA templates* whereby each of them represents a specific application domain e.g., SLA templates for the medical, telco or life science domain. Thus, each service provider may assign his/her service to a particular template (see step 1 in Figure 4.1(a)) and afterwards assign SLA mappings, if necessary (see step 2). Each template *a* may have *n* services assigned. Available templates can be browsed using an appropriate GUI.

Service consumers may search for the services using meta-data and search terms (step 3). After finding appropriate services each service consumer may define mappings to the associated template (step 4). Thereafter, the negotiation between service consumer and service provider may start as described in the next section. SLA mappings should be defined in a dynamic way. Thus, SLA templates can be updated frequently to reflect the actual SLAs used by service provides and consumers based on predefined adaptation rules (step 5). The adaptability functionality facilitates the generation of user driven public SLA templates.

4.2. SLA-Mappings Transformations. Figure 4.1(b) depicts a scenario for defining XSL transformations. As the SLA specification language we use Web Service Level Agreements (WSLAs) [37]. We also developed first bootstrapping strategies for communication across different SLA specification languages [6].

Templates are publicly available and published in a searchable registry. Each participant may download already published templates and compare it in a semi-automated or automated way with the local template. If there are any inconsistencies discovered, the service consumer may write rules (XSL transformation) from his/her local SLA template to the remote template. The rules can also be written by using appropriate visualization tools, for example using a GUI as depicted in Figure 6.1. Thereafter, the rules are stored in the database and can be applied during the runtime to the remote template. Since during the negotiation process transformations are done in two directions, the transformations from the remote SLA template to the local template are necessary as well.

As depicted in Figure 4.1(b), a service consumer is generating an SLA. The locally generated SLA plus the rules defining transformations from local SLA to remote SLA deliver an SLA which is complaint to the remote SLA. In the second case the remote template has to be translated into the local one. In that case the remote SLA plus the rules defining transformations from the remote to local SLA deliver an SLA which is compliant to the local SLA. Thus, the negotiation may be done between non-matching SLAs in both directions: from service consumer to service provider and vice versa.

The service provider can define rules for XSL transformations in the same way as depicted in Figure 4.1(b) from the publicly published SLA templates to the local templates. Thus, both parties, provider and consumer, may match on a publicly available SLA template.

4.3. SLA-Mappings Document (SMD). Figure 4.2 shows a sample rule for XSL transformations where price defined in Euros is transformed to an equivalent price in US Dollars. Please note that for the case of simplicity we use a relatively simple example. Using XSLT more complicated mappings can also be defined. Explanation of this is out of scope of this paper.

As shown in Figure 4.2, the Euro metric is mapped to the Dollar metric. In this example we define the mapping rule returning Dollars by using the *Times* function of *WSLA Specification* (see line 4). The *Times*

```
1.~\dots
2. <xsl:template \dots >
3.  <xsl:element name="Function" \dots >
4.   <xsl:attribute name="type"> <xsl:text>Times</xsl:text> </xsl:attribute>
5.   <xsl:attribute name="resultType"> <xsl:text>double</xsl:text> </xsl:attribute>
6.  <xsl:element name="Operand" \dots >
7.   <xsl:copy> <xsl:copy-of select="@*|node()"/> </xsl:copy>
8.  </xsl:element>
9.  <xsl:element name="Operand" \dots >
10.  <xsl:element name="FloatScalar" \dots > <xsl:text>1.27559</xsl:text> </xsl:element>
11. </xsl:element>
12. </xsl:element>
13.</xsl:template>
14\dots .
```

Fig. 4.2. *Example XSL Transformation*

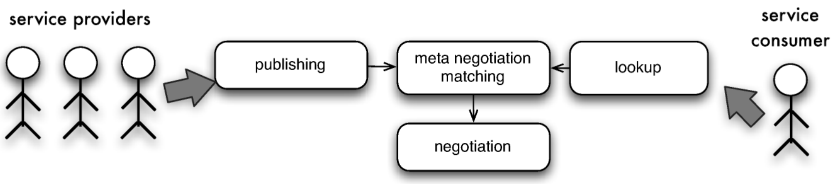


Fig. 5.1. *Meta-negotiation phases*

function multiplies two operands: the first operand is the Dollar amount as selected in line 7, the second operand is the Dollar/Euro quote (1.27559) as specified in line 10. The dollar/euro quote can be retrieved by a Web service and is usually not hard coded.

With similar mapping rules users can map simple syntax values (values of some attributes etc.), but they can even define complex semantic mappings with considerable logic. Thus, even syntactically and semantically different SLA templates can be translated into each other.

5. Negotiation Bootstrapping with Meta-negotiations. In this section, we present an example scenario for the meta-negotiation architecture and describe the document structure for publishing negotiation details into the meta-negotiation registry.

5.1. Scenario. As depicted in Figure 5.1, the meta-negotiation infrastructure can be employed in the following manner:

Publish. A service provider publishes descriptions and conditions of supported negotiation protocols into the registry (see Section 6).

Lookup. Service consumers perform lookup on the registry database by submitting their own documents describing the negotiations that they are looking for.

Match. The registry discovers service providers who support the negotiation processes that a consumer is interested in and returns the documents published by the service providers.

Negotiate. Finally, after an appropriate service provider and a negotiation protocol is selected by a consumer using his/her private selection strategy, negotiations between them may start according to the conditions specified in the provider's document.

Note that in this scenario, the consumer is looking for an appropriate service provider. The reverse may happen as well, wherein a consumer advertises a job or a task to be carried out and many providers bid to complete it. In such cases, the providers would perform the lookup.

5.2. Registry Document. The participants publishing into the registry follow a common document structure that makes it easy to discover matching documents. This document structure is presented in Figure 5.2 and consists of the following main sections. Each document is enclosed within the

`<meta-negotiation>...</meta-negotiation>`

tags. The document contains an `<entity>` elements defining contact information, organization and ID of the participant. The `<ID>` element defines the unique identifier given to the meta-negotiation document by the registry. The publisher can update or delete the document using the identifier. Each meta-negotiation

```
1.  <meta-negotiation
2.    xmlns:xsi="\dots ‘‘xsi:noNamespaceSchemaLocation="\dots ‘‘>
3.    <entity>
4.     <contact name="\dots ‘‘ phoneNumber="\dots ‘‘ />
5.     <organization name= ‘‘University of \dots ‘‘
6.~\dots
7.     <ID name="1234"/>
8.    </entity>
9.    <pre-requisite>
10.    <role name="consumer"/>
11.    <security> <authentication value="GSI location="uri"/> </security>
12.    <negotiation-terms>
13.     <negotiation-term name="beginTime"/>
14.     <negotiation-term name="endTime"/>
15.     <negotiation-term name="price"/>
16.    </negotiation-terms>
17.   </pre-requisite>
18.   <negotiation>
19.    <document name="WSLA" value="uri" version="1.0" />
20.     <document name="WS-Agreements" value="uri" version="1.0" />
21.     <protocol name="alternateOffers" schema="uri" version="1.0" location="uri"/>
22.   </negotiation>
23.   <agreement>
24.    <confirmation name="arbitrationService" value="uri"/>
25.   </agreement>
26. </meta-negotiation>
```

FIG. 5.2. *Example document for meta-negotiation registry*

comprises three distinguishing parts, namely *pre-requisites*, *negotiation* and *agreement* as described in the following paragraphs.

Pre-requisites. The conditions to be satisfied before negotiations are defined within the `<pre-requisite>` element (see Figure 5.2, lines 9–17). Pre-requisites define the *role* a participating party takes in a negotiation, the *security credentials* and the *negotiation terms*. The `<role>` element defines whether the specific party wants to engage in the negotiation as a provider or as a consumer of resources. The `<security>` element specifies the authentication and authorization mechanisms that the party wants to apply before starting the negotiation process. For example, in Figure 5.2, the consumer requires that the other party should be authenticated through the *Grid Security Infrastructure (GSI)* [15] (line 11). The negotiation terms specify QoS attributes that a party is willing to negotiate and are specified in the `<negotiation-term>` element. For example, in Figure 5.2, the negotiation terms of the consumer are *beginTime*, *endTime*, and *price* (lines 13–15).

Negotiation. Details about the negotiation process are defined within the `<negotiation>` element. In Figure 5.2, the consumer supports two document languages and one negotiation protocol. Each document language is specified within `<document>` element. In Figure 5.2, *WSLA* and *WS-Agreements* are specified as supported document languages. Additional attributes specify the *URI* (Uniform Resource Indicator) to the API or WSDL for the documents and their versions supported by the consumer (lines 18–22). In Figure 5.2, *AlternateOffers* is specified as the supported negotiation protocol. In addition to the *name*, *version*, and *schema* attributes, the URI to the WSDL or API of the negotiation protocols is specified by the *location* attribute (line 21).

Agreement. Once the negotiation has concluded and if both parties agree to the terms, then they have to sign an agreement. This agreement may be verified by a third party organization or may be lodged with another institution who will also arbitrate in case of a dispute. These modalities are specified within the `<agreement>` clause of the meta-negotiation document. For example, in Figure 5.2, a third party service, called *arbitrationService*, is specified for confirming the agreement between the two parties.

6. VieSLAF framework.

In this section we present the architecture used for the semi-automatic management of *meta-negotiations* and *SLA mappings*. We discuss a sample architectural case study exemplifying the usage of VieSLAF. Thereafter, we describe each *VieSLAF*'s core component in detail.

6.1. VieSLAF architecture.

As discussed in Section 3 *VieSLAF* framework represents the first prototype for the management of self-governing ICT Infrastructures. The *VieSLAF* framework enables application developers to efficiently develop adaptable service-oriented applications simplifying the handling with numerous Web service specifications. The framework facilitates management of QoS models as for example management of meta-negotiations [8] and SLA mappings [7]. Based on *VieSLAF* framework service provider may easily manage

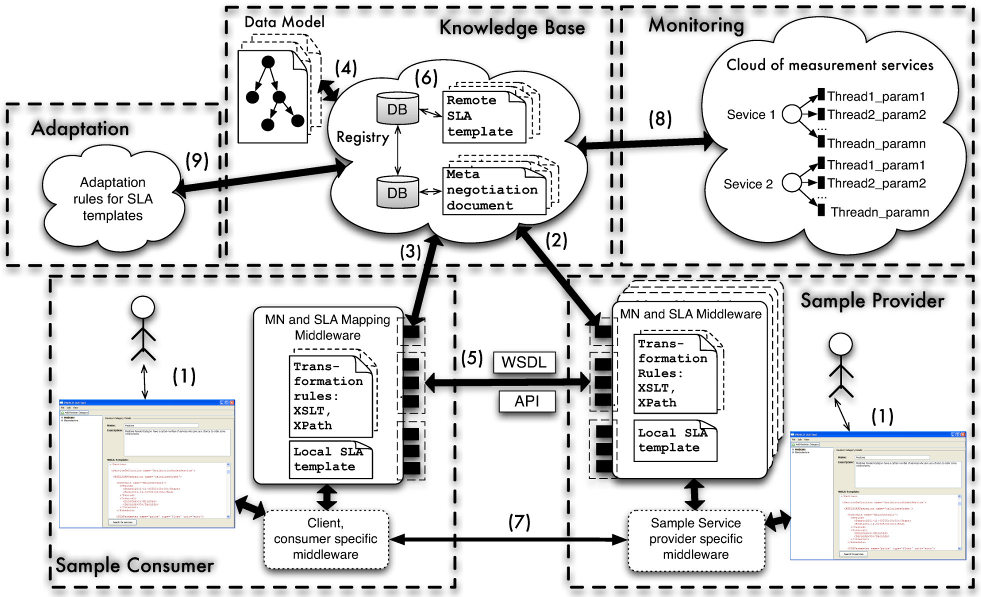


Fig. 6.1. *VieSLAF Architecture*

QoS models and SLA templates and frequently check whether selected services satisfy developer's needs e.g., specified QoS-parameters in SLAs. Furthermore, we discuss basic ideas about the adaptation of SLA templates.

We describe the *VieSLAF* components based on Figure 6.1. As shown in step (1) in Figure 6.1 users may access the registry using a GUI, browse through existing templates and meta-negotiation documents using the MN and SLA mapping middleware. In the next step (2), service provider specify MN documents and SLA mappings using the MN and SLA mapping middleware and submit it to the registry. Thereafter, in step (3), service consumer may query existing meta-negotiation documents, define own SLA mappings to remote templates. MN and SLA mapping middleware on both sides (provider's and consumer's) facilitate management of MNs and SLA mappings. Submitted MN documents and SLA mappings are parsed and mapped to a predefined data model (step 4). After meta-negotiation and preselection of services, service negotiation may start using the negotiation protocols, document languages, and security standards as specified in the MN document (step 5). During the negotiation SLA mappings and XSLT transformations are applied (step 6). After the negotiation, invocation of the service methods may start (step 7). SLA parameters are monitored using the monitoring service (step 8). Based on the submitted SLA mapping publicly available SLA templates are adapted reflecting the majority of local SLA templates (step 9).

6.1.1. Knowledge Base. As shown in Figure 6.1 *knowledge base* is responsible for storing SLA templates, SLA mappings and meta-negotiation documents. For storing of SLA templates and MN documents we implemented registries, representing searchable repositories. Currently, we implemented a MS-SQL 2008 database with a Web service front end that provides the interface for the management of SLA mappings and a PostgreSQL for the management of meta-negotiations. Thus, for scalability issues we rather intent to host the registries using a cloud of databases hosted on a service provider such as Google App Engine [14] or Amazon S3 [2]. The database is manipulated based on the role-model. The registry methods are implemented as Windows Communication Foundation (WCF) services and can be accessed only with the appropriate access rights. We define three roles: *service consumer*, *service provider* and *registry administrator*. *Service consumers* are able to search suitable services for the selected service categories e.g., by using the method *findServices*. Service consumer may also create SLA mappings using the method *createAttributeMapping*. *Service providers* may publish their services and bind it to a specific template category using the method *createService*. Furthermore, both service consumer and provider may submit and query MN documents.

6.2. Meta-negotiation Middleware. The *meta-negotiation middleware* facilitates publishing of the meta-negotiation documents into the registry and the integration of the meta-negotiation framework into the existing client and/or service infrastructure, including, for example, negotiation or security clients. Besides

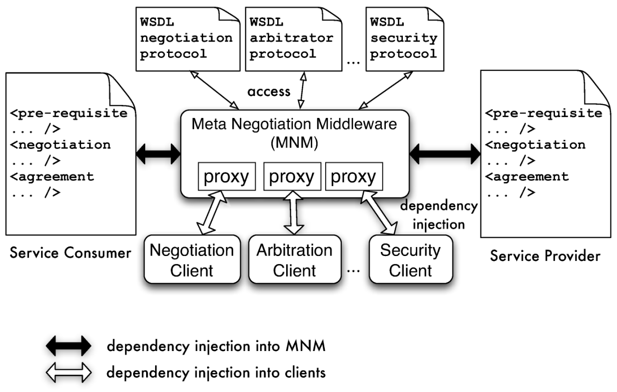


Fig. 6.2. *Meta-negotiation middleware*

acting as a client for publishing and querying meta-negotiation documents (steps 1 and 2 in Figure 6.1), the middleware delivers necessary information for the existing negotiation clients, i. e. information for the establishment of the negotiation sessions (step 4, Figure 6.1) and information necessary to start a negotiation (step 5 in Figure 6.1). As shown in Figure 6.1 each service consumer may negotiate with multiple service providers concurrently. As mentioned in Section 5 even the reverse may happen as well, wherein a consumer advertises a job. In such cases, the providers would negotiate with multiple consumers.

After querying the registry and applying a client-based strategy for the selection of the appropriate service, the information from the service's meta-negotiation document is parsed. Thereafter, meta-negotiation information is incorporated into the existing client software using a dependency injection framework such as Spring[1]. This dependency injection follows an Inversion of Control approach wherein the software is configured at runtime to invoke services that are discovered dynamically rather than known and referenced beforehand. This is suitable for meta-negotiation wherein a participant discovers others at runtime through the registry and has to dynamically adapt based on the interfaces provided by his counterpart (usually through a WSDL document).

Figure 6.2 shows an example of how this would work in practice. On the consumer side, the middleware queries the registry and obtains matching meta-negotiation documents. The middleware parses the meta-negotiation document of the selected provider and dynamically injects the interfaces discovered from the WSDLs in the document for security, negotiation and arbitration services into the existing abstract clients. Currently, we support semi-automatic integration of existing clients into meta-negotiation middleware wherein the existing clients are extended with the XML-based configuration files which are then automatically populated with the discovered interfaces.

6.2.1. SLA Mapping Middleware. As already mentioned in Section 6.1.1 SLA mapping middleware is based on different WCF services. For the sake of brevity, in the following we discuss just a few of them. The *RegistryAdministrationService* provides methods for the manipulation of the database where administrator rights are required e.g., creation of template categories. Another example represents *WSLAMappingService*, which is used for the management of SLA mappings by service consumer and service provider. *WSLAQueryingService* is used to query the SLA mapping database. The database can be queried based on template categories, SLA attributes and similar attributes. Other implemented WCF service are for example services for SLA parsing, XSL transformations, and SLA validation.

Service consumers may search for appropriate services through *WSLAQueryingService* and define appropriate SLA mappings by using the method *createAttributeMapping*. Each query request is checked during the runtime, if the service consumer has also specified any SLA mappings for *SLAElements* and *SLAAttributes* specified in the category's SLA template. Before the requests of service consumers can be completely checked, SLA transformations are applied. The rules necessary for the transformations of attributes and elements can be found in the database and can be applied using the consumer's SLA template. Thereafter, we have the consumer's template completely translated into category's SLA template. Transformations are done by *WSLATransforma-*

[1]http://www.springframework.org/

tor implemented with the .NET 3.5 technology and using LINQ[2]. In the following we explain monitoring and adaptation service in more detail.

Monitoring Service. As depicted in Figure 6.1, we implemented a lightweight concept for monitoring of SLA parameters for all services published in a specific template category. The aim of the monitoring service is to frequently check the status of the SLA parameters of an SLA agreement and deliver information to the service consumer and/or provider. Monitoring starts after publishing a service in a category and is provided through the whole lifetime of the service. Monitoring service is implemented as an internal registry service, similar to other services for parsing, transformation, and validation, that we have already explained in previous sections. Resources are monitored by *host monitor* using arbitrary monitoring tools (e.g. Ganglia). Resources metrics include e.g., down-time, up-time, available storage. Based on the predefined mappings stored in a database, monitored metrics are periodically mapped to the SLA parameters as described in [11].

After publishing service and SLA mappings, SLAs are parsed and it is identified which SLA parameters have to be monitored and how. We distinguish between periodically measured SLA parameters and the parameters which are measured on request. The values of the periodically measured parameters are stored in the so-called *parameter-pool.* The monitoring service provides two methods: a knock-in method for starting the monitoring and a method for receiving the measured SLA parameters from the measurement pool. Whenever a user requests monitoring information of the particular SLA (i) SLAs parameters are requested from the *parameter-pool* in case of periodically measured parameters or (ii) SLA parameters are immediately measured as defined in the parsed and validated SLAs in case of on-request parameters.

Adaptation Service. Remote SLA templates should not be defined in a static way, they should reflect provider's and consumer's needs. We implemented a first prototype of an internal registry's adaptation service. Thereby, mappings supplied by the consumers or the providers are evaluated. Based on the evaluation outcome a new version of the particular SLA template can be automatically defined.

Each SLA mapping can be defined as a *ParameterWish* (add/delete) and stored as an XML chunk. *Registry administrators* have to configure a learning capability property for each template category. Regression models represent one of the promising learning functions. Whenever a new *ParameterWish* is accepted a new revision category of an SLA template is generated. All services and consumers who voted for that specific *wish* are automatically re-published to the new revision. Also all SLA mappings are automatically assigned to the new template revision. Old SLA mappings of the consumers and services are deleted and also all old background threads used for calculation for old SLA template are aborted. The newly generated SLA template is thereafter parsed and new background monitoring threads are created and started for each service. Thus, based on the presented adaptation approach public templates can be derived in a user driven way reflecting the majority of local templates. Application of the learning functions is discussed in more detail in Section 7.3.2.

7. VieSLAF Evaluation and Case Studies.

In this section we evaluate the *VieSLAF* framework. In Section 7.1 we evaluate SLA mappings. In Section we 7.2 we evaluate meta-negotiations. In Section 7.3 we report some successful VieSLAF case studies.

7.1. Evaluation of SLA mappings.

In Section 7.1.1 we measure the overhead produced by SLA mappings compared to Web service invocation without mappings. We describe the experimental testbed and the setup used. Thereafter, we discuss the experimental results. In Section 7.1.2 we discuss stress tests with the varying number of concurrently invoked SLA mappings. In Section 7.1.3 we present results with the varying number of SLA mappings per single Web service invocation.

7.1.1. Overhead Test.

In order to test the *VieSLAF* framework we developed a testbed as shown in Figure 7.1(a). As a client machine we used an Acer Aspire Laptop, Intel Core 2 Duo T5600 1.83 GHz, 2 MB L2 Cache, 1GB RAM. For hosting of 10 sample services, calculator services with 5 methods, we used a single core Xenon 3.2Ghz, L1 cache, 4GB RAM Sun blade machine. We use the same machine to host *VieSLAF*s WCF services. The aim of our evaluation is to measure the overhead produced using *VieSLAF*'s *WSLAQueryingService* for search and mappings of the appropriate services.

We created 10 services (S1,..., S10) and 10 accounts for service providers. We also created the registry administrator's role, which manages the creation of template categories with the corresponding SLA templates. The SLA template represents a remote calculator service with five methods: *Add, Subtract, Multiply, Divide and Max.* Both, the provider and the consumers define five *SLAMappings*, which have to be used during the

[2]Language Integrated Query

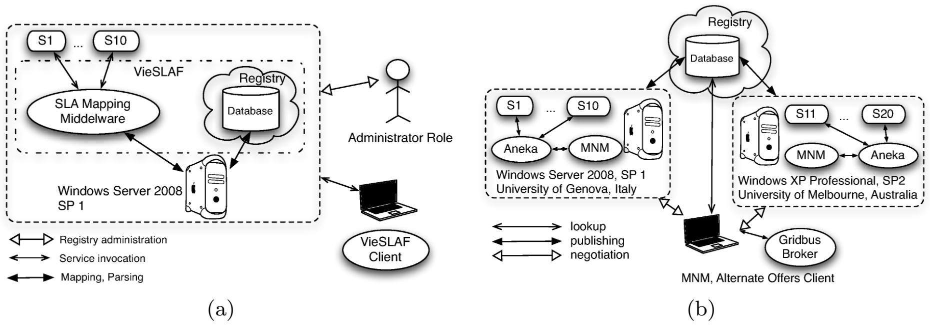


Fig. 7.1. *VieSLAF Testbed (a) for the evaluation of SLA mappings and (b) meta-negotiations*

Table 7.1
SLA Mappings Overhead Compared to Simple Web Service Invocation (Without SLA Mappings)

	Service Search Time				**Total**
	SLA-Mapping			**Remaining Time**	
	Validation	Consumer Map.	Provider Map.		
Time in sec	0.046	0.183	0.151	1.009	1.389
Time [%]	3.32	13.17	10.87	72.64	100.00

runtime. We specify three simple, syntactic mappings where we only change the name of an element or attribute. The other two mappings consider also semantic mappings, where we map between structurally different SLA templates.

Table 7.1 shows the experimental results. The measured values represent the arithmetic mean of 20 service invocations. The overhead measured during the experimental results includes the time needed for validation of SLA documents (column *Validation* in Table 7.1), the time necessary to perform mappings from the local consumers to the remote SLA templates (column *Consumer Mapping*) and the time necessary to transform the remote SLA templates to the local providers (column *Provider Mapping*). Furthermore, we measured the *remaining time* necessary to perform a search. The remaining time includes the round trip time for a search including data transfer between the client and the service and vise versa. As shown in Table 7.1 the time necessary to handle SLA mappings ($Validation + ConsumerMapping + ProviderMapping$) represents 0.38 seconds or 27,36% of the overall search time.

Please note that the intention of the presented experimental results is the proof of concept of the SLA mapping approach. We did not test the scalability issues, since we intend to employ computing Clouds like Google App Engine [14] or Amazon S3 [2] in order to cope with the scalability issues.

7.1.2. Stress Tests. In this section we describe tests on how the *VieSLAF* middleware copes with the multiple SLA mappings executed concurrently with differing complexity. Evaluation is done on an Acer Aspire Laptop, Intel Core 2 Duo T5600 1.83 GHz, 2 MB L2 Cache, 1GB RAM. For the evaluation we have used two different SLA mappings:

- Simple: Invocation of the simple SLA mappings, an example is translation of one attribute to another attribute e.g., *usage price* to *price*.
- Complex: Represents the invocation of the complex SLA mappings, as for example semantic mappings considering two structurally different SLA templates.

We tested *VieSLAF* with different versions of XSLT transformers, namely with *XSLTCompiledTransform*, .Net version 3.0 and with the obsolete *XSLTTransform Class* from .Net 1.1. Figure 7.2(a) shows the measurements with the *XSLTCompiledTransform* Transformer and with the *XSLTTransform* Class. The x axis depicts the number of SLA mappings performed concurrently i. e., number of runs. The y axis depicts the measured time for the execution of SLA mappings in seconds.

Considering the measurement results we can observe that the *XSLTTransform* Class is faster than the *XSLTCompiledTransform* Transformer from the newer .Net version. Complex mappings executed with the *XSLTTransform* Class almost overlap with the simple mappings executed with the *XSLTCompiledTransform*.

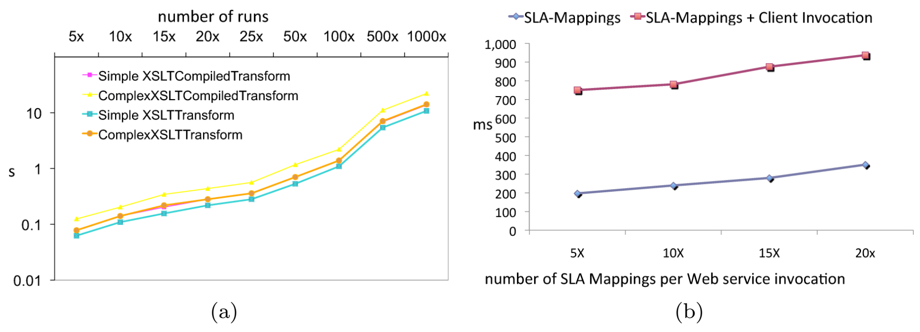


Fig. 7.2. *(a)Stress Tests with* XSLTCompiledTransform *Transformer and* XSLTTransform Class *(b) Measurements with varying number of SLA mappings per Web Service Invocation*

We can observe that in both cases, simple and complex mapping, the performance starts to significantly decrease with the number of SLA mappings > 100. If the number of mappings < 100, the execution time is about or less than 1 second.

7.1.3. Multiple SLA Mapping Tests. In this section we discuss performance results measured during a Web service call with varying numbers of SLA mappings per service. We measured 5, 10, 15 and 20 SLA mappings per Web service call. In order to create a realistic testbed we used SLA mappings which depend on each other: e.g., attribute A is transformed to attribute B, B is transformed to C, C to D, and so on. Thus, we simulate the worst case, where SLA mappings can not be performed concurrently, they have to be performed sequentially.

Evaluation is done on an Acer Aspire Laptop, Intel Core 2 Duo T5600 1.83 GHz, 2 MB L2 Cache, 1GB RAM. Figure 7.2(b) shows measured results. The x axis depicts the number of SLA mappings performed concurrently or sequentially considering attribute dependencies. The y axis depicts the measured time for the execution of SLA mappings in milliseconds. We executed SLA mappings between the remote template and the provider's template (i. e., provider mappings as described in Table 7.1) before the runtime, because these mappings are known before consumer requests. Thus, only mappings between the consumer's template and the remote template are done during the runtime as indicated with the *SLA Mapping* line. The line *SLA Mapping + Client invocation* comprises the time for the invocation of a Web service method including SLA mapping time. The *SLA Mapping + Client invocation* line does not comprise round-trip time, it comprises only the request time.

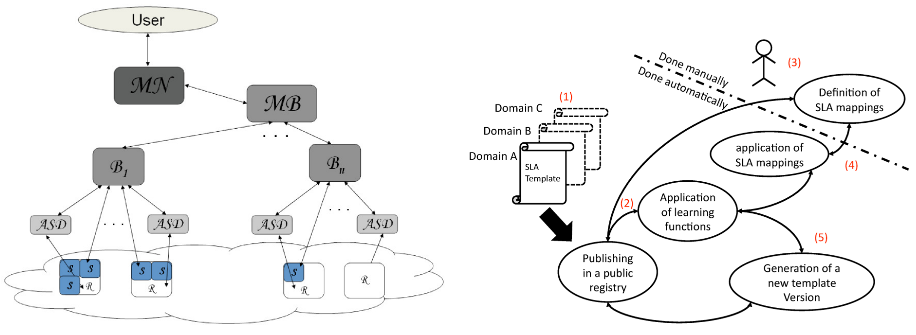
We can conclude that even with the increasing number of SLA mappings and considering the worst case scenario with sequentially performed mappings the SLA mapping time represents about 20% of the overall execution time.

7.2. Evaluation of the Meta-Negotiation Approach. In this section we evaluate the meta-negotiation approach as shown in Figure 7.1(b). We have used the Gridbus broker [32] as an example service consumer and an enterprise Grid constructed using Aneka [10] as a service provider. The aim of this evaluation was to test the overhead of the meta-negotiation infrastructure on the overall negotiation process.

7.2.1. Testbed. As shown in Figure 7.1, we deployed the registry in a machine running Windows Server 2003. The registry was accessible through a Web service interface and used a PostgreSQL database on its backend. Since the aim of these experiments was only to test the meta-negotiation framework, we isolated the Negotiation Service from the resource management system. Hence, it would reject any proposal for node reservation as it would not be able to determine node availability. We deployed 20 such services—*(S1,... S10)* on machines in a student lab in the Department of Computer Science and Software Engineering, University of Melbourne, Australia and *(S11,... S20)* on machines in the Department of Communication Computer and System Sciences, University of Genova, Italy. Negotiations with services located in Melbourne would terminate in single rounds (a proposal followed by a rejection). Services located in Italy would terminate after 2 retries. We published a meta-negotiation document for each service into the registry with different negotiation terms and document languages. The Gridbus broker was started on a machine in the Department of Computer Science, University of Melbourne and queried the registry in order to select an appropriate service provider. It would then open a negotiation process with the selected Aneka Negotiation Service.

Table 7.2
Experimental results of the meta negotiation approach

	Overall Negotiation			Total
	Meta-Negotiation		Negotiation	
	Querying	Parsing		
Time in sec	2.91	0.02	15.10	18.03
Time [%]	16.16	0.01	83.73	100.00



(a) (b)

Fig. 7.3. *(a) Architecture for the SLA-based Resource Virtualization [20] (b) Lifecycle of the SLA Template as used in [29]*

7.2.2. Experimental Results. The results of our evaluation are shown in Table 7.2. As shown in Table 7.2 the time necessary to query the registry represents 2.91 seconds or 16.16% of the overall negotiation time. Query time is calculated as the time necessary to get the list of the IDs, i. e. invocation of the method *query(XMLdocument)*, plus the time necessary to fetch each document, i. e. multiple invocations of the method *getDocument(ID)*. The time necessary to fetch each document represents about 0.2 sec. Thus, in our experiments we fetched about 15 XML documents in average, since $2.91/0.2 \approx 15$. Please note, that all times used in Table 7.1 are average times measured over 10 rounds. Time necessary to parse the selected meta-negotiation document and to inject the WSDL information into the client is 0.02 seconds or 0.01% of the overall negotiation time. Thus, time for the completion of the meta-negotiation is 2.93 seconds or 16.17% of the overall negotiation time. The time for the meta-negotiation is calculated as the the sum of the time necessary to query the registry (2.91 seconds) and the time necessary to parse the selected meta document (0.02 seconds).

The time necessary to negotiate with an *Aneka* service represents 15.10 seconds or 83.73% of the overall negotiation time. We observed that the negotiation time with services located in Italy represents about 15 seconds (due to 2 retries), since the time necessary to negotiate with services located in Melbourne represents about 5 seconds. Thus, in our experiments we have obviously negotiated only with services located in Italy. We started an alternate offers negotiation with only one round. Thus, the overall negotiation time is 18.03 seconds. Overall negotiation time is calculated as the sum of the time necessary to complete the meta-negotiation and time necessary to complete the negotiation.

Considering the fact that the time necessary to complete a meta-negotiation represents only 16.17% of the overall negotiation time, and considering the fact that we have used negotiations with only one round, we can show that the overhead of the meta-negotiations do not significantly influence the overall negotiation time.

With the presented experiments we demonstrated the applicability of our approach to the proposed architecture. Since we plan to use computational clouds in the future, the intention of the presented experiments was not to test the scalability of our approach.

7.3. Case Studies. Besides FoSII project, which was the primary reason for the development of the VieSLAF framework, VieSLAF has been successfully applied to additional case studies. In Section 7.3.1 we discuss SLA-based resource virtualization approach for on-demand service provision. In Section 7.3.2 we discuss the application of VieSLAF framework for the liquidity management in Cloud markets.

7.3.1. An SLA-based Resource Virtualization Approach For On-demand Service Provision. As discussed in [20] VieSLAF's meta-negotiation concept has been successfully utilized for the realization of the

SLA based resource virtualization environment. Thereby, an integrative infrastructure has been provided for on demand service provision based on SLAs. As depicted in Figure 7.3(a) users describe the requirements for an SLA negotiation on a high level using the concept of meta-negotiations (MN). During the meta-negotiation only those services are selected, which understand specific SLA document language and negotiation strategy or provide a specific security infrastructure. After the meta-negotiation process, a meta-broker (MB) selects a broker that is capable of deploying a service with the specified user requirements. Thereafter, the selected broker negotiates with virtual or physical resources (R) using the requested SLA document language and using the specified negotiation strategy. Once the SLA negotiation is concluded, service (S) can be deployed on the selected resource using the Automatic Service Dployer (ASD).

7.3.2. Liquidity Management in Cloud Markets. As discussed in [29] we demonstrated the problems caused in computational Cloud markets by a large number of resource definitions, namely low liquidity for each available resource type. To counteract this problem, we applied SLA mappings, which ensures sufficient liquidity in the market. SLA mapping techniques not only simplify the search for similar offers but also allow us to derive public SLA templates from all existing offerings (i. e. consumer-defined service level contracts or unsigned service level agreements). Figure 7.3(b) depicts the lifecycle of a public template. As indicated through step (1), we assume that for specific domains, specific SLA templates are generated, e.g. medicine, telecommunication. These generated SLA templates are then published in the public registry (step (2)). At the same time, learning functions for the adaptation of these public SLA templates are defined. Thereafter, SLA mappings are defined manually by users (step (3)). During the lifetime of an SLA template adaptation of SLA mappings are done automatically as described in Section 6 (step (4)). Based on the learning function and based on the submitted SLA mappings, a new version of the SLA template can be defined and published in the registry (step (5)).

8. Conclusion and Future Work. In this paper we presented the goals of the Foundations of Self-Governing ICT Infrastructures (FoSII) project and how these goals can be achieved using the principles of autonomic computing. We discussed novel meta-negotiation and SLA mapping solutions for Cloud services bridging the gap between current QoS models and Cloud middleware and representing important prerequisites for the establishment of autonomic Cloud services. We discussed the approaches for meta-negotiation and SLA mapping representing implementation of negotiation bootstrapping and service mediation approaches. Furthermore, we presented the *VieSLAF* framework used for the management of meta-negotiations and SLA mappings. We discussed how SLA templates can be adapted based on the submitted SLA mappings. We presented performance evaluation of the *VieSLAF* representing first proof of concepts. Moreover, we briefly introduced two case studies, namely the SLA-based resource virtualization approach for on-demand service provision and the approach for the liquidity management in Cloud markets. Both case studies showed the impact of the *VieSLAF* framework beyond the aforementioned FoSII project.

In the future we will improve learning function and facilitate different knowledge management methods as for example case based reasoning.

Acknowledgments. The work described in this paper was partially supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII). The authors want to thank Vincent C. Emeakaroha for carefully proofreading the paper.

REFERENCES

[1] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail. *G-qosm: Grid service discovery using qos properties.* Computing and Informatics, 21:363–382, 2002.

[2] Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2/`

[3] D. Ardagna, G. Giunta, N. Ingraffia, R. Mirandola, and B. Pernici. *QoS-Driven Web Services Selection in Autonomic Grid Environments.* Grid Computing, High Performance and Distributed Applications (GADA) 2006 Internat. Conference, Montpellier, France, Oct 29 - Nov 3, 2006.

[4] D. Bein, A. K. Datta, S. Yellenki.A Link-Cluster Route Discovery Protocol For ad hoc Networks. Scalable Computing: Practice and Experience Volume 9, Number 1, pp. 21–28, 2008.

[5] I. Brandic. *Towards Self-manageable Cloud Services.* RTSOAA 2009, in conjunction with the 33rd Annual IEEE International Computer Software and Applications Conference, Seattle, USA, July 2009.

[6] I. Brandic, D. Music, S. Dustdar. *Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services.* Grids meet Autonomic Computing Workshop 2009 - GMAC09. In conjunction with the 6th International Conference on Autonomic Computing and Communications Barcelona, Spain, June 15–19, 2009.

[7] I. Brandic, D. Music, Ph. Leitner, S. Dustdar. VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management. The 6th International Workshop on Grid Economics and Business Models 2009 (Gecon09). In conjunction with Euro-Par 2009, 25–28 August 2009, Delft, The Netherlands

[8] I. Brandic, S. Venugopal, Michael Mattess, and Raykumar Buyya. *Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services.* Workshop on Service-Oriented Engineering and Optimizations 2008. In conjunction with International Conference on High Performance Computing 2008 (HiPC 2008), Bangalore, India, December 17–20, 2008.

[9] R. Buyya, Ch. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. *Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility.* Future Generation Computer Systems, 25(6):599-616, June 2009.

[10] X. Chu, K. Nadiminti, Ch. Jin, S. Venugopal, and R. Buyya *Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications.* Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science 2007), Dec. 10-13, 2007, Bangalore, India.

[11] V. C. Emeakaroha, I.Brandic, M. Maurer, S. Dustdar. *Low Level Metrics to High Level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in Cloud environments.* The 2010 High Performance Computing and Simulation Conference (HPCS 2010) June 28—July 2, 2010, Caen, France. *to appear.*

[12] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. *Web services on demand: WSLA-driven automated management.* IBM Systems Journal, 43(1), 2004.

[13] G. Dobson, A. Sanchez-Macian. *Towards Unified QoS/SLA Ontologies.* Proceedings of the 2006 IEEE Services Computing Workshops (SCW 2006), Chicago, Illinois, USA, 18-22 September 2006.

[14] Google App Engine, `http://code.google.com/appengine`

[15] I. Foster, and C. Kesselman, and G. Tsudik, and S. Tuecke. *A Security Architecture for Computational Grids*, Proc. 5th ACM Conference on Computer and Communications Security Conference, San Francisco, CA, USA, ACM Press, New York, USA, 1998.

[16] Foundations of Self-Governing ICT Infrastructures (FoSII) Project, `http://www.wwtf.at/projects/research_projects/details/index.php?PKEY=972_DE_O`

[17] P. C. K. Hung, L. Haifei, and J. Jun-Jang. *WS-Negotiation: an overview of research issues.* Proceedings of the 37th Annual Hawaii International Conference on System Sciences, Big Island, Hawaii, 5-8 January 2004.

[18] Z. Hill, J. C. Rowanhill, A. Nguyen-Tuong, G. S. Wasson, J. C. Knight, J. Basney, M. Humphrey. *Meeting virtual organization performance goals through adaptive grid reconfiguration.* 8th IEEE/ACM International Conference on Grid Computing (Grid 2007), Austin, Texas, USA, September 19-21, 2007.

[19] J. O. Kephart, D.M. Chess, *The vision of autonomic computing.* Computer, 36:(1) pp. 41-50, Jan 2003.

[20] A. Kertesz, G. Kecskemeti, I. Brandic. *An SLA-based Resource Virtualization Approach for On-demand Service Provision.* VTDC 2009, In conjunction with the 6th International Conference on Autonomic Computing and Communications Barcelona, Spain, June 15–19, 2009.

[21] B. Koller, L. Schubert. *Towards autonomous SLA management using a proxy-like approach.* Multiagent Grid Syst. 3(3), 2007, IOS Press, Amsterdam, The Netherlands, The Netherlands.

[22] S. König, S. Hudert, T. Eymann, and M. Paolucci. *Towards Reputation Enhanced Electronic Negotiations for Service Oriented Computing.* In Proceedings of the 11th International Workshop on Trust in Agent Societies (TRUST 2008), Estoril, Portugal, May 12-13, 2008.

[23] Z. Li, M. Parashar: An Infrastructure for Dynamic Composition of Grid Services. 7th IEEE/ACM International Conference on Grid Computing (Grid 2006), Barcelona, Spain, September 28-29, 2006.

[24] D. Nurmi, R. Wolski, Ch. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov. *The Eucalyptus Open-source Cloud-computing System.* Proceedings of Cloud Computing and Its Applications 2008, Chicago, Illinois, October 2008.

[25] D. M. Quan, J. Altmann. *Resource allocation algorithm for light communication grid-based workflows within an SLA context.* International Journal of Parallel, Emergent and Distributed Systems (IJPEDS), 24(1):31-48, 2009.

[26] N. Oldham, K. Verma, A. P. Sheth, and F. Hakimpour. *Semantic WS-agreement partner selection.* Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006.

[27] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar. *A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing.* in Proceedings of the 2005 European Grid Computing Conference (EGC 2005), Amsterdam, The Netherlands, February, 2005.

[28] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges, IEEE Computer, 40(11): 64-71, November 2007

[29] Marcel Risch, Ivona Brandic, Jörn Altmann. Using SLA Mapping to Increase Market Liquidity. Workshop on Non Functional Properties and Service Level Agreements Management in Service Oriented Computing Workshop (NFPSLAM-SOC'09).The 7th International Joint Conference on Service Oriented Computing, November 23-27 2009, Stockholm, Sweden.

[30] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. B.-Y., W. Emmerich, F. Galan. *The RESERVOIR Model and Architecture for Open Federated Cloud Computing.*, IBM Journal of Research and Development, 53(4) (2009)

[31] D. Thain, T. Tannenbaum, and M. Livny. *Distributed Computing in Practice: The Condor Experience.* Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[32] S. Venugopal, R. Buyya, and L. Winton, *A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids*, Concurrency and Computation: Practice and Experience, 18(6): 685-699, Wiley Press, New York, USA, May 2006.

[33] W. Vambenepe, C. Thompson, V. Talwar, S. Rafaeli, B. Murray, D. S. Milojicic, S. Iyer, K. I. Farkas, M. F. Arlitt. *Dealing with Scale and Adaptation of Global Web Services Management.* International Journal of Web Services Research, 4(3): 65-84, 2007.

[34] D. W. Walker, L. Huang, O. F. Rana, and Y. Huang. *Dynamic service selection in workflows using performance data.* Scientific Programming 15(4):235-247, 2007.

[35] L.-H. Vu, F. Porto, K. Aberer, M. Hauswirth. *An Extensible and Personalized Approach to QoS-enabled Service Discovery.*,

Eleventh International Database Engineering and Applications Symposium (IDEAS 2007), Banff, Alberta, Canada, September 6-8, 2007.

[36] Web Services Agreement Specification (WS-Agreement), `http://www.ogf.org/documents/GFD.107.pdf`

[37] Web Service Level Agreement (WSLA), `http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf`

[38] Ch. Zhou, L. T. Chia, and B. S. Lee. *Semantics in service discovery and QoS measurement.* IT Professional, 7(2): 29–34, Mar-Apr 2005.

Edited by: Marcin Paprzycki
Received: March 30, 2010
Accepted: June 09, 2010