VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management

Ivona Brandic, Dejan Music, Philipp Leitner, and Schahram Dustdar

Distributed Systems Group, Institute of Information Systems
Vienna University of Technology, Vienna, Austria
{ivona,dejan,leitner,dustdar}@infosys.tuwien.ac.at

Abstract. Novel computing paradigms like Grid and Cloud computing demand guarantees on non-functional requirements such as application execution time or price. Such requirements are usually negotiated following a specific Quality of Service (QoS) model and are expressed using Service Level Agreements (SLAs). Currently available QoS models assume either that service provider and consumer have matching SLA templates and common understanding of the negotiated terms or provide public templates, which can be downloaded and utilized by the end users. On the one hand, matching SLA templates represent an unrealistic assumption in systems where service consumer and provider meet dynamically and on demand. On the other hand, handling of public templates seems to be a rather challenging issue, especially if the templates do not reflect users' needs. In this paper we present VieSLAF, a novel framework for the specification and management of SLA mappings. Using VieSLAF users may specify, manage, and apply SLA mapping bridging the gap between non-matching SLA templates. Moreover, based on the predefined learning functions and considering accumulated SLA mappings, domain specific public SLA templates can be derived reflecting users' needs.

1 Introduction

Nowadays, well established and traditional resource sharing models are shifted towards novel market-oriented models revolutionizing existing Grid and High Performance Computing (HPC) concepts [8]. In market-oriented resource sharing models users discover resources on demand and pay for the usage of the specific resources. In turn they expect that besides requested *functional* requirements, *non-functional* requirements of the application execution are also fulfilled [1,19,12]. *Non-functional* requirements comprise application execution time, reliability, availability, and similar issues. Non-functional requirements 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 i.e., SLA documents, with all required elements like parties, SLA parameters, metrics, and objectives, but without QoS values [9].

A large body of work deals with SLA based QoS negotiation and integration of QoS concepts into Grid management tools [10,17]. However, most of the

J. Altmann, R. Buyya, and O.F. Rana (Eds.): GECON 2009, LNCS 5745, pp. 60–73, 2009.

existing work relies either on inflexible QoS models assuming that the communication partners have matching *SLA templates* or provide public SLA templates, which can be downloaded and utilized on the users' system. On the one hand, matching SLA templates limit QoS negotiation only between partners where QoS relationship is already established off-line, or to partners who belong to a particular Grid portal [1]. On the other hand, publicly available SLA templates usually do not reflect users' needs. Thus, in order to increase QoS versatility, flexible QoS models are necessary where negotiation is possible even between services which do not have matching SLA templates. The problems with non-matching templates can be exemplified on a very simple example with differing terms of contract on both sides. The term price may be defined as *usage price* or *service price*, etc., leading to inconsistencies during the negotiation process. Another example of not matching templates are SLA templates which slightly differ in their structure.

In this paper we approach the gap between existing QoS methods and novel computing paradigms like Cloud Computing by proposing *VieSLAF*, a framework for the management of *SLA mappings*. Thereby, mappings are defined by XSLT[1] documents where inconsistent parts of one document are mapped to another document e.g, from the consumer's template to the provider's template. Moreover, based on SLA mappings and deployed taxonomies we eliminate semantic inconsistencies between consumer's and provider's templates. The purpose of the submitted SLA mappings is twofold: (1) using *VieSLAF* users may discover services on demand, define mappings to available templates, if necessary and finally start the negotiation with selected services. Therefore, the negotiation is not only limited to services belonging to a special portal or where a relationship is already established off-line; (2) based on *VieSLAF*'s predefined learning functions and accumulated SLA mappings we facilitate user driven definition of public SLA templates.

Based on a case study the presented SLA mapping architecture has been successfully used to manage SLA mappings in context of a Grid workflow management tool [4] and adaptable Cloud services [6]. Additionally to [4,6] where we presented the general approach for SLA mappings, in this paper we present (1) the *VieSLAF* architecture in detail with modules for the measurement of SLA parameters, (2) implementation details of the *VieSLAF* framework; and (3) first experimental results.

The main contributions of this paper are: (1) description of the scenarios for the definition of *SLA mapping* documents; (2) definition of the *VieSLAF* architecture used for the semi-automatic management of *SLA mappings* (3) demonstration of learning functions, which can be used to obtain realistic public templates and (4) evaluation of the *VieSLAF* architecture using an experimental testbed.

The rest of this paper is organized as follows: Section 2 presents the related work. In Section 3 we present our SLA mapping approach. In particular we discuss the management of SLA mappings, SLA transformations, and an example

[1] XSL Transformations (XSLT) Version 1.0, http://www.w3.org/TR/xslt.html

SLA mapping document. Section 4 presents the *VieSLAF* architecture including the used semantic model, methods for SLA mappings and transformations, used registries and features for the SLA monitoring and adaptation of SLA templates. In Section 5 we discuss our first experimental results. Section 6 concludes this paper and describes the future work.

2 Related Work

Currently, a large body of work exists in the area of Grid service negotiation and SLA-based QoS. Most of the related work can be classified into the following three categories: (1) adaptive SLA mechanisms based on OWL, DAML-S and other semantic technologies [17,10,23]; (2) SLA based QoS systems, which consider varying service requirements but do not consider non matching SLA templates [1,20]; and (3) systems relying on the principles of autonomic computing [3,14,15].

Work presented in [18] discusses the 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 [17]. 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 [10]. 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 [23]. 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 [13].

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 [16]. Dan et al. present a framework for providing customers of Web services differentiated levels of service through the use of automated management and SLAs [9]. 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 [21]. Thereby, SLAs can be exploited to define certain QoS parameters that a service has to maintain during its interaction with a specific customer [14]. König at al. investigate the trust issue in electronic negotiations,

dealing with how to trust a potential transaction partner and how to choose such partners based on their past behavior [15].

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. Moreover, none of the presented approaches facilitates user driven definition of publicly available SLA templates.

3 The SLA Mapping Approach

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 3.1. The transformations between remote and local SLA templates are discussed in Section 3.2. Finally, an example SLA mapping document is presented in Section 3.3.

3.1 Management of SLA Mappings

Figure 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 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.

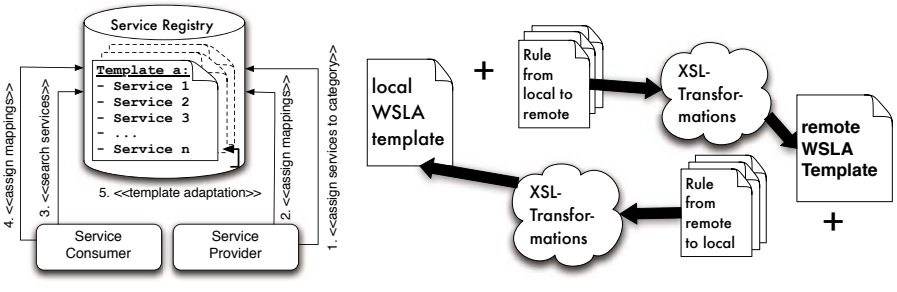


Fig. 1. Management of SLA-Mappings (a) QoS basic scenario (b)

Currently, SLA mappings are defined on an XML level, where users define XSL transformations. A UML based GUI for the management of SLA-mappings is under development [4].

3.2 SLA-Mappings Transformations

Figure 1(b) depicts a scenario for defining XSL transformations. As the SLA specification language we use Web Service Level Agreements (WSLA)s [22]. We also developed first bootstrapping strategies for communication across different SLA specification languages [5].

WSLA templates are publicly available and published in a searchable registry. Each participant may download already published WSLA 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 WSLA 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 3. Thereafter, the rules are stored in the database and can be applied during the runtime to the remote WSLA template. Since during the negotiation process transformations are done in two directions, the transformations from the remote WSLA template to the local WSLA template are necessary as well.

As depicted in Figure 1(b), a service consumer is generating a WSLA. The locally generated WSLA plus the rules defining transformations from local WSLA to remote WSLA deliver a WSLA which is compliant to the remote WSLA. In the second case the remote WSLA template has to be translated into the local one. In that case the remote WSLA plus the rules defining transformations from the remote to local WSLA deliver a WSLA which is compliant to the local WSLA. Thus, the negotiation may be done between non-matching WSLAs 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 1(b) from the publicly published WSLA templates to the local WSLA templates. Thus, both parties, provider and consumer, may match on a publicly available WSLA template.

3.3 SLA-Mappings Document (SMD)

Figure 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 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 5). The *Times function* multiplies two operands: the first operand is the Dollar amount as selected in line 12, the second operand

```
1. ...
2. <xsl:template ...>
3.   <xsl:element name="Function" ...>
4.     <xsl:attribute name="type">
5.       <xsl:text>Times</xsl:text>
6.     </xsl:attribute>
7.     <xsl:attribute name="resultType">
8.       <xsl:text>double</xsl:text>
9.     </xsl:attribute>
10.    <xsl:element name="Operand" ...>
11.      <xsl:copy>
12.        <xsl:copy-of select="@*|node()"/>
13.      </xsl:copy>
14.    </xsl:element>
15.    <xsl:element name="Operand" ...>
16.      <xsl:element name="FloatScalar" ...>
17.        <xsl:text>1.27559</xsl:text>
18.      </xsl:element>
19.    </xsl:element>
20.  </xsl:element>
21.</xsl:template>
22. ...
```

Fig. 2. Example XSL Transformation

is the Dollar/Euro quote (1.27559) as specified in line 17. 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.

4 VieSLAF Framework

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

4.1 VieSLAF Architecture

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 and SLA mappings [7]. Based on the *VieSLAF* framework service providers may easily manage 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 needed for the generation of realistic public SLA templates.

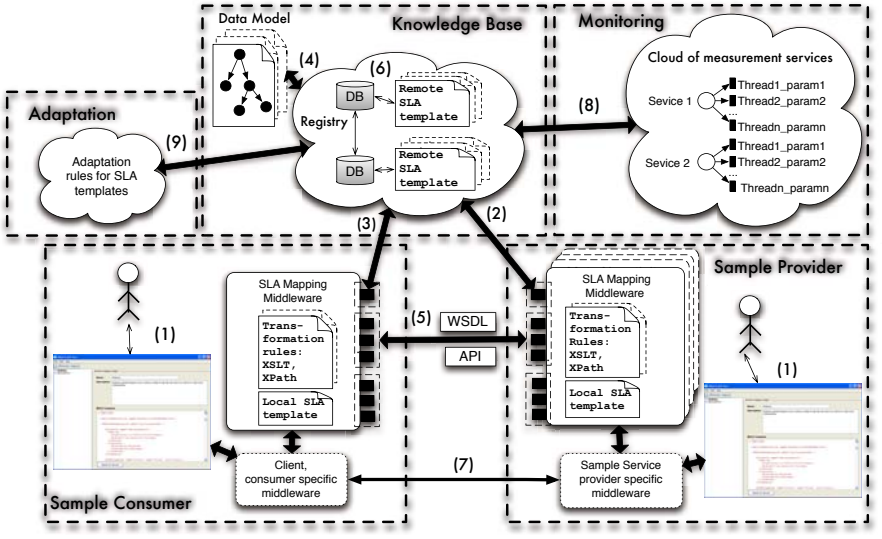


Fig. 3. VieSLAF Architecture

We describe the *VieSLAF* components based on Figure 3. As shown in step (1) in Figure 3 users may access the registry using a GUI, browse through existing templates using the SLA mapping middleware. In the next step (2) service providers specify SLA mappings using the SLA mapping middleware and submit them to the registry. Thereafter, in step (3) service consumers may define their own SLA mappings to the remote templates and submit them to the registry. SLA mapping middleware on both sides (provider's and consumer's) facilitates the management of SLA mappings. Submitted SLA mappings are parsed and mapped to a predefined data model (step 4). Thereafter, service negotiation may start (step 5). During the negotiation SLA mappings and XSLT transformations are applied (step 6). After the negotiation, the 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).

4.2 VieSLAF Components

As shown in Figure 3 the major *VieSLAF* components are the knowledge base, components for monitoring and adaptation and the SLA middleware used by service provider and consumer.

Knowledge Base. As shown in Figure 3 the *knowledge base* is responsible for storing SLA templates and SLA mapping documents. For storing of SLA template documents we implemented registries representing searchable repositories. Currently we have implemented a MS-SQL 2008 database with a Web

service front end that provides the interface for the management of SLA mappings. To handle scalability issues we intend to host the registries using a cloud of databases hosted on a service provider such as Google App Engine [11] or Amazon EC2 [2]. SLA templates are stored in a canonical form enabling comparison of XML documents. The registry methods are implemented as Windows Communication Foundation (WCF) services and can be accessed only with appropriate access rights. The database is manipulated based on the role-model. 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 consumers 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*.

Sample Provider and Sample Consumer. A sample provider and a sample consumer are shown in the lower part of Figure 3. Basically, a service consumer/provider consists of a client/service based middleware, SLA mapping middleware facilitating the access to registries, and a GUI used for browsing remote templates.

SLA Mapping Middleware. As already mentioned 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 the *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 at runtime, if the service consumer has also specified any *SLA-mappings* for *SLAElements* and *SLAAttributes* specified in category's *SLA-Template*. SLA transformations are applied before the requests of the service consumers can be completely checked. The rules necessary for the transformations of attributes and elements can be found in the database and can be applied using the consumer's *WSLA-Template*. Thereafter, the consumer's template is completely translated into a category's *WSLA-Template*. Transformations are done by *WSLATransformator* implemented with the .NET 3.5 technology and using LINQ[2].

Monitoring Service. As depicted in Figure 3, we implemented a lightweight concept for the monitoring of SLA parameters for all services published in a

[2] Language Integrated Query.

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 the information to the service consumer and/or provider. Furthermore, the monitoring service monitors values of SLA parameters as specified in the *SLA-Template* of the published services. Monitoring starts after publishing a service in a category and is provided through the whole lifetime of the service. The 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. In the following we describe how the monitoring process can be started i.e., all the steps necessary to setup monitoring.

After the publishing of the 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. Thus, we implemented a first prototype of an internal registry's adaptation service, which can be used by consumers and providers as shown in Figure 3 in order to derive realistic public SLA templates. Users can specify *SLAParameters* which should be added into *SLA-Template* or choose some *SLAParameters* which they do not need and want to delete.

Each *ParameterWish* (add/delete) is saved as an XML chunk that contains all *SLAParameter*s with metrics which should be added/deleted from a specific *SLA-Template*. *Registry administrators* have to configure a learning capability property for each template category. The property defines how many requests for a specific *ParameterWish* have to be defined in order to add/delete *ParameterWish* to/from an *SLA-Template*. 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.

5 Evaluation

In this section we evaluate the *VieSLAF* framework. In Section 5.1 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 5.2 we discuss stress tests with the varying number of concurrently invoked SLA mappings. In Section 5.3 we present results with the varying number of SLA mappings per single Web service invocation.

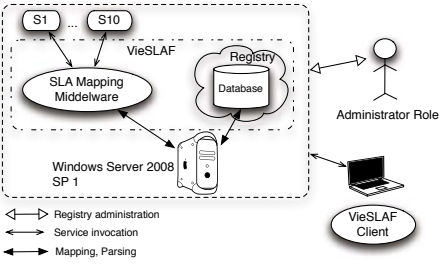


Fig. 4. VieSLAF Testbed

5.1 Overhead Test

In order to test the *VieSLAF* framework we developed a testbed as shown in Figure 4. 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, 2MB 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 SLA 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 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 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 WSLA documents (column *Validation* in Table 1), the time necessary to perform SLA-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

Table 1. SLA Mappings Overhead Compared to Simple Web Service Invocation (Without SLA Mappings)

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

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 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 [11] or Amazon EC2 [2] in order to cope with the scalability issues.

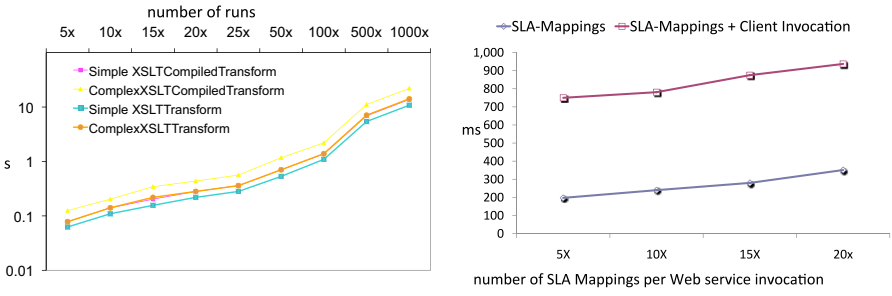
5.2 Stress Tests

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 5(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*. We can observe that in both cases, simple and complex mapping, the



(a) (b)

Fig. 5. Stress Tests with *XSLTCompiledTransform* Transformer and *XSLTTransform Class* (a) Measurements with varying number of SLA mappings per Web Service Invocation (b)

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.

5.3 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 5(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 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.

6 Conclusion and Future Work

In this paper we presented the *VieSLAF* framework used for the management of SLA mappings. SLA mappings are necessary in service oriented Grids and computational Clouds where service consumer and provider usually do not have matching SLA templates. Thus, based on SLA mappings even those partners with slightly different templates may negotiate with each other and increase the number of potential negotiation partners. We have demonstrated how Grid service users (provider and consumer) may search for appropriate services, define SLA mappings, if necessary, and finally start service negotiation and execution. Using *VieSLAF* users can even monitor SLA parameters during the execution of the service calls. Thereafter, we presented how the SLA mappings and the predefined learning functions can be used to adapt SLA templates. Adaptability functions facilitate generation of user driven public SLA templates. Finally, we discussed our first proof of concept based on the experimental results.

In the future we plan to extend our work on adaptable Cloud services and test our approach with real life applications.

Acknowledgments

The work described in this paper was partially supported by the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-Cube) and by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII).

References

1. Al-Ali, R.J., Rana, O.F., Walker, D.W., Jha, S., Sohail, S.: 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. Ardagna, D., Giunta, G., Ingraffia, N., Mirandola, R., Pernici, B.: QoS-driven web services selection in autonomic grid environments. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4276, pp. 1273–1289. Springer, Heidelberg (2006)
4. Brandic, I., Music, D., Dustdar, S., Venugopal, S., Buyya, R.: Advanced QoS Methods for Grid Workflows Based on Meta-Negotiations and SLA-Mappings. In: The 3rd Workshop on Workflows in Support of Large-Scale Science. In conjunction with Supercomputing 2008, Austin, TX, USA, November 17 (2008)
5. Brandic, I., Music, D., Dustdar, S.: Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services. In: Grids meet Autonomic Computing Workshop 2009 - GMAC 2009. In conjunction with the 6th International Conference on Autonomic Computing and Communications Barcelona, Spain, June 15-19 (2009)
6. Brandic, I.: Towards Self-manageable Cloud Services. In: The Second IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications (RTSOAA 2009). In conjunction with the 33rd Annual IEEE International Computer Software and Applications Conference, Seattle, Washington, USA, July 20-24 (2009)

7. Brandic, I., Venugopal, S., Mattess, M., Buyya, R.: Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services. In: 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)
8. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: 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 (2009)
9. Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., Youssef, A.: Web services on demand: WSLA-driven automated management. IBM Systems Journal 43(1) (2004)
10. Dobson, G., Sanchez-Macian, A.: Towards Unified QoS/SLA Ontologies. In: Proceedings of the 2006 IEEE Services Computing Workshops (SCW 2006), Chicago, Illinois, USA, September 18-22 (2006)
11. Google App Engine, `http://code.google.com/appengine`
12. Foundations of Self-Governing ICT Infrastructures (FoSII) Project, `http://www.wwtf.at/projects/research_projects/details/index.php?PKEY=972_DE_0`
13. Hung, P.C.K., Haifei, L., Jun-Jang, J.: WS-Negotiation: an overview of research issues. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences, Big Island, Hawaii, January 5-8 (2004)
14. Koller, B., Schubert, L.: Towards autonomous SLA management using a proxy-like approach. Multiagent Grid Syst. 3(3) (2007)
15. König, S., Hudert, S., Eymann, T., Paolucci, M.: Towards reputation enhanced electronic negotiations for service oriented computing. In: Falcone, R., Barber, S.K., Sabater-Mir, J., Singh, M.P. (eds.) Trust 2008. LNCS, vol. 5396, pp. 273–291. Springer, Heidelberg (2008)
16. Quan, D.M., Altmann, J.: 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)
17. Oldham, N., Verma, K., Sheth, A.P., Hakimpour, F.: Semantic WS-agreement partner selection. In: Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26 (2006)
18. Ouelhadj, D., Garibaldi, J.M., MacLaren, J., Sakellariou, R., Krishnakumar, K.T.: A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 651–660. Springer, Heidelberg (2005)
19. Venugopal, S., Buyya, R., Winton, L.: A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids. In: Concurrency and Computation: Practice and Experience, vol. 18(6), pp. 685–699. Wiley Press, New York (2006)
20. Walker, D.W., Huang, L., Rana, O.F., Huang, Y.: Dynamic service selection in workflows using performance data. Scientific Programming 15(4), 235–247 (2007)
21. Web Services Agreement Specification (WS-Agreement), `http://www.ogf.org/documents/GFD.107.pdf`
22. Web Service Level Agreement (WSLA), `http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf`
23. Zhou, C., Chia, L.T., Lee, B.S.: Semantics in service discovery and QoS measurement. IT Professional 7(2), 29–34 (2005)