Technical University of Vienna
Information Systems Institute
Distributed Systems Group

VieSLAF Framework: Increasing the Versatility of Grid QoS Models by Applying Semi-automatic SLA-Mappings

Ivona Brandic, Dejan Music, Philipp Leitner and Schahram Dustdar
ivona@infosys.tuwien.ac.at
dejan@infosys.tuwien.ac.at
leitner@infosys.tuwien.ac.at
dustdar@infosys.tuwien.ac.at

TUV-1841-2009-02 March 9, 2009

Novel computing paradigms like Grid and Cloud computing facilitate efficient, flexible, and on demand provision of the computational resources. In such models users usually pay for the resource usage and expect that non-functional requirements are satisfied, as for example, in terms of execution time, reliability, and availability. Non-functional requirements are negotiated between service provider and consumer using Service Level Agreements (SLAs) standards. Currently available Quality of Service (QoS) models assume that service provider and consumer have matching SLA templates and common understanding of the negotiated terms. However, this is an unrealistic assumption in systems where service consumer and provider meet each other dynamically and on demand. In this paper we present VieSLAF, a novel framework for the specification and management of SLA mappings bridging the gap between non-matching SLA templates. Using our framework users can browse publicly available template registries, assign their local templates to the remote templates, define SLA mappings if necessary and finally start negotiation with services. After the discussion on VieSLAF architecture we present the solutions for the monitoring of the SLA parameters as well as first experimental results.

Keywords: Cloud Computing, Grid Computing, Service-Oriented Architecture

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

VieSLAF Framework: Increasing the Versatility of Grid QoS Models by Applying Semi-automatic SLA-Mappings

Ivona Brandic, Dejan Music, Philipp Leitner and Schahram Dustdar
Distributed Systems Group
Institute of Information Systems
Vienna University of Technology
Vienna, Austria
Email: {ivona,dejan,leitner,dustdar}@infosys.tuwien.ac.at

***Abstract*—Novel computing paradigms like Grid and Cloud computing facilitate efficient, flexible, and on demand provision of the computational resources. In such models users usually pay for the resource usage and expect that non-functional requirements are satisfied, as for example, in terms of execution time, reliability, and availability. Non-functional requirements are negotiated between service provider and consumer using Service Level Agreements (SLAs) standards. Currently available Quality of Service (QoS) models assume that service provider and consumer have matching SLA templates and common understanding of the negotiated terms. However, this is an unrealistic assumption in systems where service consumer and provider meet each other dynamically and on demand. In this paper we present VieSLAF, a novel framework for the specification and management of SLA mappings bridging the gap between non-matching SLA templates. Using our framework users can browse publicly available template registries, assign their local templates to the remote templates, define SLA mappings if necessary and finally start negotiation with services. After the discussion on VieSLAF architecture we present the solutions for the monitoring of the SLA parameters as well as first experimental results.**

I. Introduction

Nowadays, well established and traditional resource sharing models are shifted towards novel market-oriented resource sharing models revolutionizing existing Grid and High Performance Computing (HPC) concepts [7]. 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* also *non-functional* requirements of the application execution are fulfilled [1], [24]. *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 [18], [10], [21]. However, most of the existing work relies on inflexible QoS models assuming that the communication partner have matching *SLA templates*. Matching SLA templates limits QoS negotiation only between partners where QoS relationship is already established off-line, or to partners who belong to a particular Grid portal [1]. In commercially used Grids and especially in case of computational clouds matching SLAs are an unrealistic assumption since services are discovered dynamically and on demand. 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 different price units on provider's and consumer's side (e.g., Euro and Dollar) or SLA templates which 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 SLA template. 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. A user friendly GUI for the definition and specification of SLA mappings simplifies the management of SLA mappings. Based on a case study the presented SLA mapping architecture has been successfully used to manage SLA mappings in context

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

of a Grid workflow management tool [5]. Additionally to [5] where we presented the general approach of SLA mappings in context of a Grid workflow management tool, 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; (3) and first experimental results.

ViesLAF has been developed in the context of the Vienna Science and Technology Fund (WWTF) granted project Foundations of Self-Governing ICT Infrastructures (FoSII) [14]. The aim of the FoSII project is to develop an adaptive service infrastructure where services automatically adapt themselves to the occurred failures or environmental changes. Management of the inconsistencies between SLAs and negotiation bootstrapping is also one of the major FoSII goals. *VieSLAF* framework represents the first attempt in achieving these goals.

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* including monitoring service; and (3) evaluation of the *VieSLAF* architecture using an experimental testbed.

The rest of this paper is organized as follows: Section II presents the related work. Section III presents the *VieSLAF* architecture including the used semantic model, methods for SLA mappings and transformations, used registries and features for the SLA monitoring. In Section IV we discuss our first experimental results. Section V concludes this paper and describes the future work.

II. Related Work

Currently, large body of work has been done in the area of Grid service negotiation and SLA-based QoS [19], [8]. Most of the related work can be classified into following three categories: (i) adaptive QoS systems, which do not use standard SLA languages for the generation and negotiation of electronic contracts [23]; (ii) adaptive SLA mechanisms based on OWL, DAML-S and other semantic technologies [21], [10], [29]; (iii) SLA based QoS systems, which consider varying service requirements but do not consider non matching SLA templates [3], [1], [26]. Moreover, several projects are dealing with resource lookup based on functional and non-functional requirements.

Work presented in [22] discusses incorporation of SLA-based resource brokering into existing Grid systems. Glatard et al. discuss a probabilistic model of workflow execution time evaluated in context of EGEE grid infrastructure [11]. Work described in [26] presents an approach for dynamic workflow management and optimisation using near-realtime performance with strategies for choosing an optimal service, based on user-specified criteria, from several semantically equivalent Web services.

Oldham et al. describes a framework for semantic matching of SLAs based on WSDL-S and OWL [21]. 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. surveys 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 [29]. Hung et al. proposes 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 [16].

Ardagana et al. [3] presents an autonomic grid architectures with mechanisms to dynamically re-configure service center infrastructures, which is basically exploited to fullfill varying QoS requirements. 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. Work presented in [8] proposes generalized resource management model where resource interactions are mapped onto a well defined set of platform-independent SLAs. The model is based on Service Negotiation and Acquisition Protocol (SNAP) providing the lifetime management SLAs.

Condor's ClassAds mechanism is used to represent jobs, resources, submitters, and other Condor daemons [23]. Dan et al. [9] presents a framework for providing customers of Web services differentiated levels of service through the use of automated management and SLAs. Zhao et al. discusses how semantic technologies can be used for workflow provenance [28], whereas work described in [13] discusses how semantic technologies may be used by mobile devices which need to locate and select appropriate Grid services in an automatic and flexible way.

Verma provides an overview of service level agreements in IP networks. The work identies three common approaches that are used to satisfy SLAs in IP networks [25]. Jurca et al. proposes a new form of SLAs where the price is determined by the QoS which is actually delivered by service provider. For the monitoring of QoS a novel approach is introduced based on reputation mechanism [17].

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. Our approach for the SLA mappings is presented in the following.

III. SLA Mapping Architecture based on VieSLAF framework

In this section we present the architecture used for the semi-automatic management of *SLA mappings*. First, we discuss the architectural components for the management the *SLA mappings* based on the *VieSLAF* architecture. Thereafter, we

describe the *VieSLAF*'s core components in detail and give a sample architectural case study. Furthermore, we present the extended *VieSLAF* architecture with semantic data model and monitoring service for the SLA parameters.

A. VieSLAF Overview

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 [6]. Based on *VieSLAF* framework service provider 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. As already mentioned *VieSLAF* has been developed in the context of the WWTF funded project Foundations of Self-Governing ICT Infrastructures (FoSII) [14].

B. VieSLAF architecture

Using an appropriate GUI users i.e., service provider and consumer, may browse through templates (as shown in step 1, Figure 1). Thereafter, they can publish services to the registry and define SLA mappings from local WSLAs (user's and provider's SLAs), to remote WSLAs (publicly available SLA templates), and vice versa (steps 2 and 3 in Figure 1). We classify service templates into categories i.e., for each specific domain, as for example medical, telecommunication or financial domain we provide a single template. Using the GUI user may browse and select templates to which he wants to specify SLA mappings. In the following we present the *VieSLAF*'s registry concept.

1) Registry: The registry is a searchable repository for SLA templates and mappings. Currently, this is implemented as a MS-SQL 2008 database with a Web service front end that provides the interface for the management of SLA mappings. However, it is possible to host the registry using a cloud of databases hosted on a service provider such as Google App Engine [12] 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 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, they may define *SLA-mappings* by using the method *createAttributeMapping*. Registry administrators may create, update and delete service categories. Please note that for all *create* methods we have implemented corresponding CRUD[2] methods. Each *template category* is identified with an unique name and is stored in the database as an XML document. Each service is identified with a name and is described with a *WSDL-URI*, *contract*, *binding* and filled *SLA-template* of the selected category. For each service multiple SLA-mapping documents may be defined.

2) SLA-Mapping Middleware: The aim of the SLA mapping middleware is to facilitate access to the registry on the provider's and consumer's side as shown in Figure 1. SLA mapping middleware can be easily plugged into existing middleware as we demonstrated in [5] with Aneka and Gridbus broker and Amadeus workflow framework. Aneka [6] is a resource management system for enterprise Grids composed of machines running Microsoft Windows operating system. Aneka provides facilities for advance reservation of computing nodes and supports flexible scheduling of applications constructed using different parallel programming models such as bag-of-tasks and dataflow computing. The Gridbus Broker [24] maps jobs to appropriate resources considering various restrictions specified by terms of *functional* and *non-functional* requirements.

As already mentioned in Section III-B1 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*, as depicted in step 5 of Figure 1. Each query request is checked during the runtime, if the service consumer has also specified any *SLA-Mappings* for *SLAElements* and *SLAAttributes* specified in category's *SLA-Template* (see steps 5 - 7 in Figure 1). Before the requests of service consumers can be completely checked, SLA transformations are applied (see step 8 in Figure 1). 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, we have the consumer's template completely translated into category's *WSLA-Template*. Transformations are done by *WSLATransformator* implemented with the .NET 3.5 technology and using LINQ[3].

As depicted in step 11 we have also implemented features for the monitoring of the SLA parameters as explained in Section III-D. Step 12, adaptation of SLA templates, is subject of ongoing work, where we develop mechanism for the automatic adaptation of the publicly available SLA templates.

After the negotiation with service's provider, (step 9) *service*

[2]create, read, update and delete

[3]Language Integrated Query

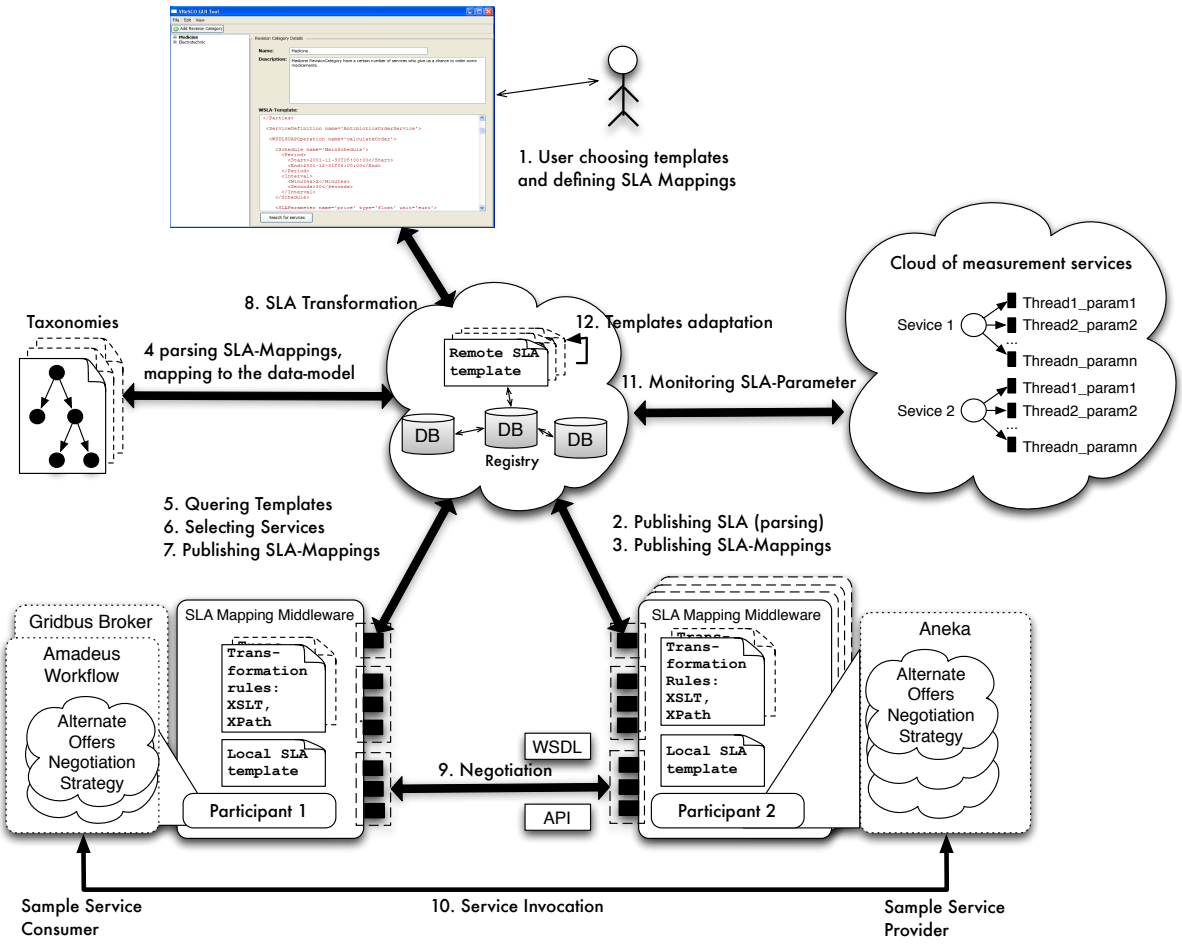

Fig. 1. Extended VieSLAF Architecture with Monitoring and Taxonomies

consumer can communicate directly through the proxy with the service (step 10). As depicted in Figure 1, SLA-templates are mapped to *VieSLAF's* semantic data model based on taxonomies. The *VieSLAF*'s data model is explained in the following.

C. VieSLAF's semantic data model

Figure 1 shows the extended *VieSLAF* architecture with the features for data mapping and monitoring of SLA parameters. As shown in that figure we map data to predefined taxonomies and thus enable semantic mapping of data (see step 4).

The semantic model is automatically filled up with data each time a *Registry Administrator* creates new template category. After this step other users of *VieSLAF's* functionality e.g., service consumers and service providers, can specify *SLA-Mappings* for each *SLAElement* or *SLAAttribute* created for this template. Additionally to already existing WSLA schema elements (e.g., *SLAElement*, *SLAAttribute*) we define *SLA-Mappings*. For each *SLAAttribute* and *SLAElement* element, one or more *SLA-Mappings* can be specified. For each *SLA-Mapping* two rules will be created: rule from local (consumer or provider) SLA template into category SLA template and vice versa.

```
1.  ...
2.  <Metric name='averageResponseTime' type='double' ...>
3.   <Source>MedicineProvider</Source>
4.    <Function xsi:type='Divide' resultType='double'>
5.     <Operand>
6.      <Function xsi:type='Plus' resultType='double'>
7.       <Operand>
8.        <Metric>averageResponseTimeHost1</Metric>
9.       </Operand>
10.      <Operand>
11.       <Metric>averageResponseTimeHost2</Metric>
12.      </Operand>
13.     </Function>
14.    </Operand>
15.    <Operand>
16.     <LongScalar>2</LongScalar>
17.    </Operand>
18.   </Function>
19.  </Metric>
20.  ...
```

Fig. 2. Example Metric

D. Monitoring Service

As depicted in Figure 1 step 11, we implemented a light-way concept for monitoring of SLA parameters for all services published in a category. The aim of the monitoring service is to frequently check the status of the SLA parameters of a SLA agreement and deliver the information to the service consumer

and/or provider. Furthermore, monitoring service monitors values of SLA parameters as specified in *SLA-Template* of the published services. Monitoring starts after publishing of 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. 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*. 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) in case of periodically measured parameters SLAs parameters are requested from the *parameter-pool* or (ii) in case of on-request parameters SLA parameters are immediately measured as defined in the parsed and validated SLAs.

According to WSLA specification SLA parameters may be (composed) metrics. Thus, we distinguish between *composite* and *resource* metrics. Examples of composite metrics are maximum response time of a service or average availability of a service. Resource metrics are for example number of service invocations, or system uptime. Functions specify how a composite metric is computed. Figure 2 depicts a composite metric *averageResponseTime* (see line 2), which is calculated as an arithmetic mean of resource metrics *averageResponseTimeHost1* (see line 8) and *averageResponseTimeHost2* (see line 11).

We have currently implemented various functions as for example: *Max, Mean, Median, NumberGreaterThenThreshold, etc.*. In Figure 2 two functions are defined. Firstly, the function *Plus* is specified, which adds *averageResponseTimeHost1* and *averageResponseTimeHost2*. Secondly, the function *Divide* is specified, which divides the amount received by function *Plus* by 2. For functions which are evaluated periodically, an evaluation period is specified by means of a computation schedule. The schedule defines the time intervals during which the functions are executed to compute the metrics. These time intervals may be for example weekly, daily, hourly, etc. Observed values are periodically stored into parameter pool of a specific service and can be queried asynchronously through the monitoring interface.

For the implementation of the monitoring services we used *Abstract Factory Pattern* and *Lazy Singleton Pattern*. Thus, we create only one lazy instance of the monitoring service which is then used by all *WCF-Services* as described in III-B2. Currently, we implemented the monitoring service with *LINQ* technology of .NET 3.5 framework. The monitoring concept

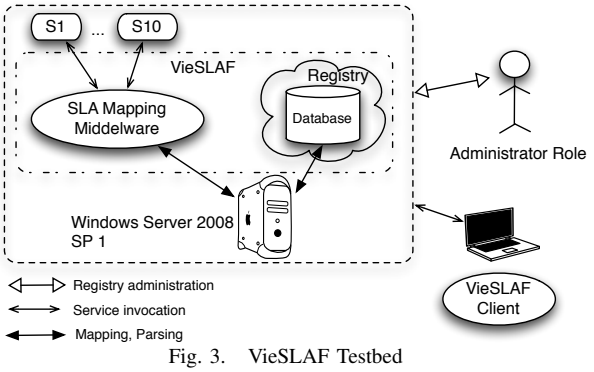

Fig. 3. VieSLAF Testbed

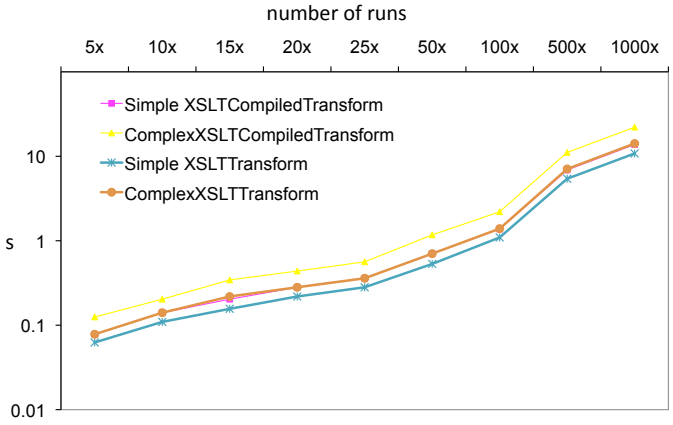

Fig. 4. *Stress Tests with* XSLTCompiledTransform *Transformer* and *XSLTTransform Class*

is implemented in an asynchronous way based on background threads. As shown in Figure 1 for each *SLAparameter* of each service one monitoring thread is started. In the future we plan to monitor also the *Obligation* part of WSLA templates and detect, if there are any violations to the negotiated SLA parameters.

IV. Evaluation

In this section we evaluate the *VieSLAF* framework. In Section IV-A we measure the overhead produced by SLA mappings compared to Web service invocation without mappings. First, we describe the experimental testbed and the used setup. Thereafter, we discuss the experimental results. In Section IV-B we discuss stress tests with varying number of SLA mappings which are invoked concurrently. In Section IV-C we present results with varying number of SLA mappings per Web service invocation.

A. Overhead Test

In order to test the *VieSLAF* framework we developed a testbed as shown in Figure 3. 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 service with 5 methods, we used single core Xenon 3.2Ghz, L1 cache, 4GB RAM Sun blade machine.

	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

TABLE I
SLA MAPPINGS OVERHEAD COMPARED TO SIMPLE WEB SERVICE INVOCATION (WITHOUT SLA MAPPINGS)

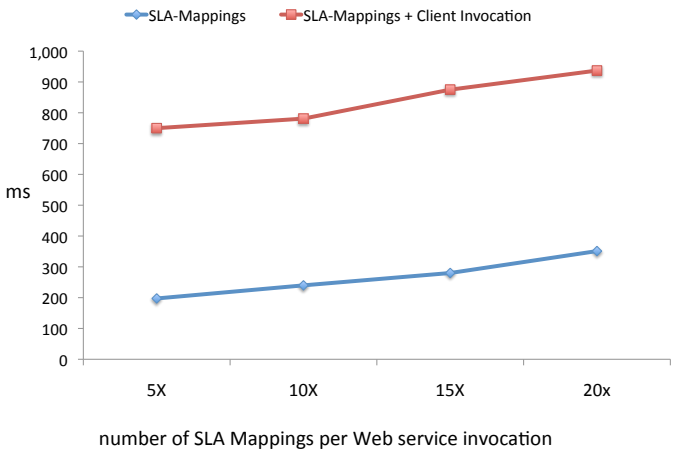

Fig. 5. Measurements with varying number of SLA mappings per Web Service Invocation

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 mapping of the appropriate services.

We created 10 services (S1,..., S10) and 10 accounts for service providers. We also created registry administrators 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, Substract, Multiply, Divide and Max*. Service providers defines five *SLAMappings* which have to be used during the runtime. Also the service consumers specifies five *SLAMappings*. 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. One mapping is used for converting of Euros to Dollars and the other one for milliseconds to seconds.

Table I shows the experimental results. The measured values represent the arithmetic mean of 20 service invocation. The overhead measured during the experimental results includes the time needed for validation of WSLA documents (column *Validation* in Table I), the time necessary to perform SLA-mappings from local consumers to remote SLA templates (column *Consumer Mapping*) and the time necessary to transform remote SLA templates to local providers templates (column *Provider Mapping*). Furthermore, we measured the the *remaining time* necessary to perform search. The remaining time includes the round trip time for a search including data transfer between client and service and vise versa. As shown in Table I 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 attempt to employ Computing Clouds like Google App Engine [12] or Amazon S3 [2] in order to cope with the scalability issues.

B. Stress Tests

With the stress tests described in this section we tested how the *VieSLAF* middleware cope with the multiple SLA mappings, differing in their complexity and which are executed concurrently. Evaluation is done on 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 very 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 already mentioned US Dollar to Euro mapping.

We have tested *VieSLAF* with different versions of XSLT transformer, namely with *XSLTCompiledTransform*, .Net version 3.0 and with the obsolete *XSLTTransform Class* from .Net 1.1. Figure 4 shows the measurements with the *XSLTCompiledTransform* Transformer and with the *XSLTTransform* Class. On the x axis we have the number of SLA mappings performed concurrently i.e., number of runs. On the y axis we have the measured time for the execution of SLA mappings in seconds.

Considering the measurements results we can observe that *XSLTTransform* Class is faster than the *XSLTCompiledTransform* Transformer form the newer .Net version. Complex mappings executed with the *XSLTTransform* Class almost overlap with the simple mappings executed with the *XSLTCompiledTransform*. Also we can observe that in both cases, in simple and complex mapping, the performance starts to significantly decrease with number of SLA mappings > 100. In case when the number of mappings < 100 the execution time is about or less than 1 second.

C. Multiple SLA Mapping Tests

In this section we discuss performance results measured during a Web service call with varying number 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 depends on each other: e.g., attribute A is transformed to attribute B, B is transformed to C, C to D, and so on. Thus, in many cases SLA mappings can not be performed concurrently, they have to be performed sequentially.

Evaluation is done on Acer Aspire Laptop, Intel Core 2 Duo T5600 1.83 GHz, 2 MB L2 Cache, 1GB RAM. Figure 5 shows measured results. We executed SLA mappings between remote template and provider's template (i.e., Provider Mappings as described in Table I) before the runtime, because these mappings are known before consumer requests. Thus, only mappings between consumer's template and 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 request time.

We can conclude that even with increasing number of SLA mappings the time necessary for the SLA mappings represents about 20% of the overall execution time.

V. 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 mapping 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 user (provider and consumer) may search for appropriate services, define SLA mappings, if necessary, and finally start service negotiation and execution. Using *VieSLAF* Grid service users can even monitor SLA parameters during the execution of the service calls. Finally, we discussed our first proof of concept based on the experimental results.

We are currently developing adaptation methods for the SLA templates. Based on the submitted SLA mappings and pre-defined learning functions public SLA templates should adapt themselves. Thus, public SLA templates should reflect the majority of the local SLA templates. Currently, in our approach we consider only WSLA document language. A challenging research issue represents the bootstrapping of other SLA document languages (e.g. WS-Agreement), where even partners which understand different SLA document languages may communicate with each other.

Acknowledgment

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) and by the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-Cube).

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 Simple Storage Services (S3), http://aws.amazon.com/s3/

[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 International Conference, Montpellier, France, Oct 29 - Nov 3, 2006.

[4] J. Blythe, E. Deelman, Y. Gil. *Automatically Composed Workflows for Grid Environments*. IEEE Intelligent Systems 19(4): 16–23 2004.

[5] I. Brandic, D. Music, S. Dustdar, S. Venugopal, and R. Buyya. Advanced QoS Methods for Grid Workflows Based on Meta-Negotiations and SLA-Mappings. The 3rd Workshop on Workflows in Support of Large-Scale Science. In conjunction with Supercomputing 2008, Austin, TX, USA, November 17, 2008.

[6] I. Brandic, S. Venugopal, Michael Mattess, and Rajkumar Buyya, *Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services*. Technical Report, GRIDS-TR-2008-9, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Aug. 8, 2008.

[7] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems, ISSN: 0167-739X, Elsevier Science, Amsterdam, The Netherlands, 2009, in press, accepted on Dec. 3, 2008.

[8] K. Czajkowski, I. Foster, C. Kesselman, V. Sander and S. Tuecke, *SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems*. 8th Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh Scotland, July 2002.

[9] 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.

[10] 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.

[11] T. Glatard, J. Montagnat, X. Pennec. *A Probabilistic Model to Analyse Workflow Performance on Production Grids*. 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), pp.510-517, Lyon, France, 19-22 May 2008.

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

[13] T. Guan, E. Zaluska, D. De Roure. *A Semantic Service Matching Middleware for Mobile Devices Discovering Grid Services*. Advances in Grid and Pervasive Computing, Third International Conference, GPC 2008, pp. 422-433 Kunming, China, May 25-28, 2008.

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

[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] P.C.K. Hung, L. Haifei, 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.

[17] R. Jurca, B. Faltings. *Reputation-based Service Level Agreements for Web Services*. In Proceedings of 3rd International Conference on Service Oriented Computing, pp. 396-409, Amsterdam, The Netherlands, December 12-15, 2005.

[18] Daniel A. Menasce, Emiliano Casalicchio: QoS in Grid Computing. IEEE Internet Computing 8(4): 85-87, 2004.

[19] A. Paschke, J. Dietrich, K. Kuhla: A Logic Based SLA Management Framework, Semantic Web and Policy Workshop (SWPW), 4th Semantic Web Conference (ISWC 2005), Galway, Ireland, 2005.

[20] I. J. Taylor, E. Deelman, D. B. Gannon. *Workflows for e-Science*. Springer Verlag, 2005.

[21] N. Oldham, K. Verma, A. P. Sheth, 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.

[22] 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.

[23] 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.

[24] 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.

[25] Dinesh C. Verma. Service Level Agreements on IP Networks. Proceedings of the IEEE, Vol. 92, No. 9, September 2004.

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

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

[28] J. Zhao, C. Goble, R. Stevens, D.Turi. *Mining Taverna's semantic web of provenance* Concurrency and Computation: Practice & Experience 20(5), John Wiley & Sons, Inc., New Jersey, April 2008.

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