Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services

Ivona Brandic, Dejan Music, Schahram Dustdar
Institute of Information Systems, Vienna University of Technology
Argentinierstraße 8, 1040 Vienna, Austria
{ivona,dejan,dustdar}@infosys.tuwien.ac.at

ABSTRACT

Nowadays, novel computing paradigms as for example Grid or Cloud Computing are gaining more and more on importance. In case of Cloud Computing users pay for the usage of the computing power provided as a service. Beforehand they can negotiate specific functional and non-functional requirements relevant for the application execution. However, providing computing power as a service bears different research challenges. On the one hand dynamic, versatile, and adaptable services are required, which can cope with system failures and environmental changes. On the other hand, human interaction with the system should be minimized. In this paper we present the first results in establishing adaptable, versatile, and dynamic services considering negotiation bootstrapping and service mediation achieved in context of the Foundations of Self-Governing ICT Infrastructures (FoSII) project. We discuss novel meta-negotiation and SLA mapping solutions for Grid/Cloud services bridging the gap between current QoS models and Grid/Cloud middleware and representing important prerequisites for the establishment of autonomic Grid/Cloud services. We present document models for the specification of meta-negotiations and SLA mappings. Thereafter, we discuss the sample architecture for the management of meta-negotiations and SLA mappings.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*; D.0 [**Software**]: General

General Terms

Management, Reliability, Performance

Keywords

Grid services, Cloud Computing, Autonomic Computing

GMAC'09, June 15, 2009, Barcelona, Spain.

1. INTRODUCTION

Service-oriented Architectures (SOA) represent a novel and promising approach for implementing ICT systems [4]. Thereby, software is packaged to services and 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 [20]. SOA approach and Web service technologies represent large scale abstractions and a candidate concept for the implementation of following novel computing paradigms (a) Grid Computing, where sophisticated scientific applications can be accessed as services over Internet [1, 5], (b) Cloud Computing, where massively scalable computing is made available to end users as a service [10]. In all those approaches the access to computing power is provided as a service.

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. The development of novel concepts for dynamic, versatile, and adaptive services represents an open and challenging research issue [16]. In this paper we discuss the first prerequisites in achieving adaptive Grid/Cloud services based on principles of autonomic computing [19].

Major goal of this paper is to facilitate service negotiation in heterogeneous Grids. In order to enable service users to find services which best fit to their needs (considering costs, execution time and other functional and non-functional properties), service users should negotiate and communicate with numerous publicly available services. In this paper we address following research problems: negotiation bootstrapping and service mediation as described next.

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 Grid/Cloud frameworks assume that the communication partner knows about the *negotiation protocols* before entering the negotiation and that they have matching *SLA templates.* In commercially used Grids and especially in case of computational clouds, this is an unrealistic assumption since services are discovered dynamically and on demand.

Thus, so-called *meta-negotiations* are required to allow two parties to reach an agreement on what specific negotiation protocols, security standards, and documents to use before starting the actual negotiation. The necessity for SLA mappings can be motivated by 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.

In this paper, we approach the gap between existing QoS methods and Grid/Cloud services by proposing an architecture for Grid/Cloud service management with components for *meta-negotiations* and *SLA mappings* [9, 8, 7]. Meta-negotiations are defined by means of a *meta-negotiation document* where participating parties may express: the prerequisites to be satisfied for a negotiation, for example, requirement for a specific authentication method; the supported negotiation protocols and document languages for the specification of SLAs; and conditions for the establishment of an agreement, for example, a required third-party arbitrator. SLA mappings are defined by XSLT[1] documents where inconsistent parts of one document are mapped to another document e.g., from consumer's template to provider's template. Moreover, based on SLA mappings and deployed taxonomies, we eliminate semantic inconsistencies between consumer's and providers SLA template.

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 sample architecture named *VieSLAF*. Section 7 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 [21, 11]. Work presented in [24] 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 [13]. Work described in [25] 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. describe a framework for semantic matching of SLAs based on WSDL-S and OWL [23].

Ardagana et al. [3] present 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 [11] 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.

Hill et al. discuss 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 [17]. Similarly to Hill et al. work discussed in Vambenepe et al. address global service management based on principles of autonomic computing [22].

Dan et al. [12] present a framework for providing customers of Web services differentiated levels of service through the use of automated management and SLAs. Work described in [15] 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. Jurca et al. propose 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 [18].

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 deals with meta negotiations supporting matching of negotiation requirements.

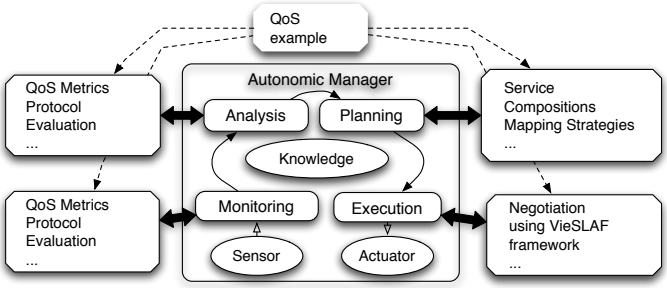

Figure 1: General Architecture of an Autonomic System Explained on a QoS Example

3. ADAPTABLE, VERSATILE, AND DYNAMIC SERVICES

In this Section we discuss how adaptable, versatile, and dynamic services can be realized.

3.1 Overview

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. We propose models and concepts for adaptive services building on the approach defined by means of autonomic computing [19, 3].

We identified the following objectives:

Negotiation bootstrapping and service mediation. The first objective is to facilitate communication between publicly available services. Usually, before service usage, service consumer and service provider have

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

to establish an electronic contract defining terms of use [6, 11]. Thus, they have to negotiate the exact terms of contract (e.g., exact execution time of the service). However, each service provides a unique negotiation protocol often expressed using different languages, representing an obstacle within the SOA architecture. We propose novel concepts for automatic bootstrapping between different protocols and contract formats increasing the number of services a consumer may negotiate with. Consequently, the full potential of public services could be exploited.

Service Enforcement. Services may fail, established contracts between services may be violated. The second objective is to develop methods for service enforcement, where failures and malfunctions are repaired on demand and where services are adapted to changing environmental and system conditions. We propose development of knowledge bases where the directives, policies, and rules for failure adjustment and repair may be specified and stored. Furthermore, adequate methods for the condition specification and condition evaluation are emerging research issues.

Service adaptivity. Service failures or violations of electronic agreements must be detected in an efficient manner. Moreover, the reaction to failures should be done in an adequate way. Thus, the third objective is the development of novel methods for modeling of intelligent logging capabilities at the level of a single service as well as composite services. Sophisticated concepts for the measurement of service execution parameters and Quality of Service (QoS) are needed as well as generic monitoring capabilities which can be customized on-demand for different services.

Service Governance. Policies and rules for service enforcement should not be defined in a static way. Moreover, the rules should evolve over time. The fourth objective is the development of the governing guidelines for rule definition and rule-evolution. This includes the development of adequate languages for rule specification and rule evolution as well as novel reasoning techniques.

In order to achieve aforementioned goals we utilize the principles of *autonomic computing*. Autonomic computing research methodology can be exemplified using Quality of Service (QoS) as shown in Figure 1. The management is done through the following steps: (i) *Monitoring*: QoS managed element is monitored using adequate software sensors; (ii) *Analysis*: The monitored and measured metrics (e.g., execution time, reliability, availability, etc.) are analyzed using knowledge base (condition definition, condition evaluation, etc.); (iii) *Planning*: Based on the evaluated rules and the results of the analysis, the planning component delivers necessary changes on the current setup e.g., renegotiation of services which do not satisfy the established QoS guarantees. (iv) *Execution*: Finally, the planned changes are executed using software actuators and other tools (e.g., *VieSLAF* framework [9]), which query for new services.

3.2 Negotiation Bootstrapping and Service Mediation

Autonomic computing can be applied for other managed elements e.g., service negotiation. In the following we explain the first steps in achieving aforementioned architecture: *meta-negotiations* and *SLA mappings*.

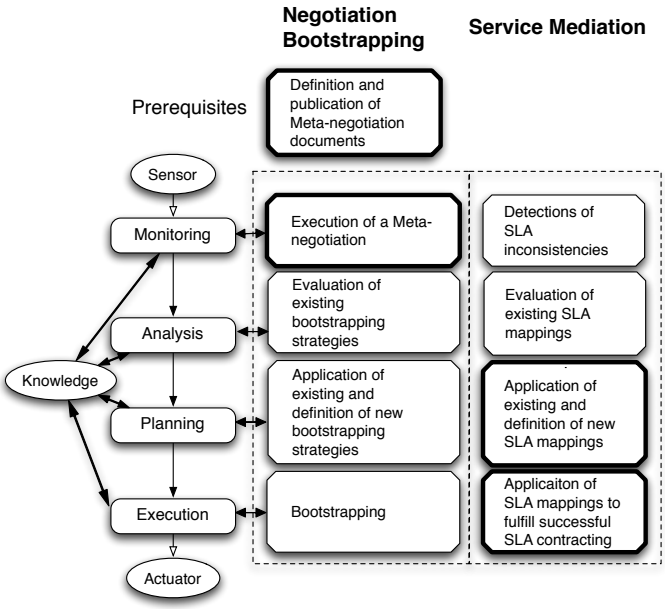

Figure 2: Negotiation Bootstrapping and Service Mediation as Part of the Autonomic Process

Figure 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 selected where negotiation bootstrapping is required. 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 2, in this paper we present solutions for the definition and accomplishment of meta-negotiations (Section 4) and for the specification and applications of SLA mappings (Section 5). In the following section we explain the principles of meta-negotiations.

4. GRID 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.

```
1. <meta-negotiation ...>
2.   ...
3.  <pre-requisite>
4.   <role name="consumer"/>
5.   <security>
6.    <authentication value="GSI" location="uri"/>
7.   </security>
8.   <negotiation-terms>
9.    <negotiation-term name="beginTime"/>
10.   <negotiation-term name="endTime"/>
11.    ...
12.  </negotiation-terms>
13. </pre-requisite>
14. <negotiation>
15.  <document name="WSLA" value="uri"
16.    version="1.0"/>
17.  <protocol name="alternateOffers"
18.    schema="uri" version="1.0" location="uri"/>
19. </negotiation>
20. <agreement>
21.  <confirmation
22.   name="arbitrationService" value="uri"/>
23. </agreement>
24.</meta-negotiation>
```

Figure 3: Example Meta-negotiation Document

4.1 Meta-Negotiation Scenario

The meta-negotiation infrastructure can be employed in the following manner: (i) *Publishing*: A service provider publishes descriptions and conditions of supported negotiation protocols into the registry; (ii) *Lookup*: Service consumers perform lookup on the registry database by submitting their own documents describing the negotiations that they are looking for. (iii) *Matching*: 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; (iv) *Negotiation*: 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.

4.2 Meta-Negotiation Document (MND)

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 3 and consists of the following main sections.

Each document is enclosed within the `<meta-negotiation>` ... `</meta-negotiation>` tags. Each meta-negotiation (MN) comprises three distinguishing parts, namely *pre-requisites*, *negotiation* and *agreement* as described in the following paragraphs.

Pre-requisites.

The conditions to be satisfied before a negotiation are defined within the `<pre-requisite>` element (see Figure 3, lines 3–13). Pre-requisites define the *role* a participating party takes in a negotiation, the *security credentials* and the *negotiation terms*. The `<security>` element specifies the authentication and authorization mechanisms that the party wants to apply before starting the negotiation process. 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 3, the negotiation terms of the consumer are *beginTime* and *endTime*, and *price* (lines 9–10).

Negotiation.

Details about the negotiation process are defined within the `<negotiation>` element. Each document language is specified within the `<document>` element. In Figure 3, *WSLA* is specified as the supported document language. Additional attributes specify the URI to the API or WSDL for the documents and their versions supported by the consumer. In Figure 3, *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 (lines 17–18).

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 as shown in lines 21–22. The meta-negotiation architecture described here was experimentally evaluated and the results were presented in a previous publication [8].

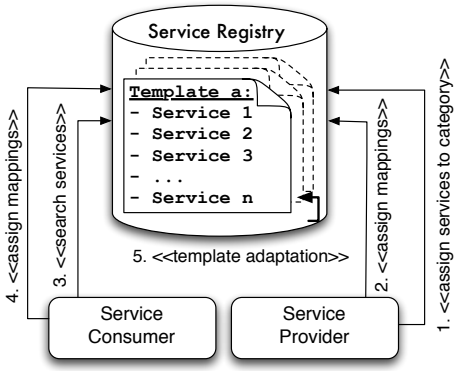

Figure 4: Management of SLA mappings

5. 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.

5.1 Management of SLA mappings

Figure 4 depicts the architecture for the management of SLA mappings and participating parties. The registry comprises different *SLA templates* whereby each of them represent a specific application domain, e.g., SLA templates for 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) and afterwards assign SLA mappings if necessary (see step 2). Each template a may have n services assigned.

Service consumer may search for the services using metadata and search terms (step 3). After finding appropriate services each service consumer may define mappings to the appropriate template the selected service is assigned to (step 4). Thereafter, the negotiation between service consumer

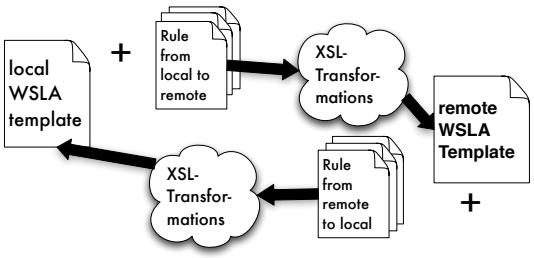

Figure 5: Scenario for XSL Transformations

and service provider may start as described in the next section. As already mentioned templates are not defined in a static way. Based on the assigned SLA mappings and the predefined rules for the adaptation, SLAs are updated frequently trying to reflect the actual SLAs used by service provides and consumers (step 5).

Currently, SLA mappings are defined on an XML level, where users define XSL transformations. However, a UML based GUI for the management of SLA mappings is subject of ongoing work [7].

5.2 Scenario for SLA mappings

Figure 5 depicts a scenario for defining XSL transformations. For the definition of SLA agreements we use Web Service Level Agreement (WSLA) [26]. WSLA templates are publicly available and published in a searchable registry. Each participant may download previously published WSLA templates and compare it with the local template. This can be done in an automatic way by using appropriate tools. We are currently developing a GUI that can help consumers to find suitable service categories. If there are any inconsistencies discovered, 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. Thereafter, the rules are stored in the database and can be applied during the runtime to the remote WSLA template. During the negotiation process, the transformations are performed from the remote WSLA template to the local WSLA template and vice versa.

Figure 5 depicts a service consumer generating a WSLA. The locally generated WSLA plus the rules defining transformation 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 complaint to the local WSLA. Thus, in this manner, the negotiation may be done using non-matching WSLAs.

Even the service provider can define rules for XSL transformations 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.

5.3 SLA mappings Document (SMD)

In this section, we present and discuss a sample SLA mapping document. Generally, SLA mappings can be defined using XSLT and XPath expressions.

Figure 6 shows a sample rule for XSL transformations where price defined in Euro is transformed to an equivalent

```
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. ...
```

Figure 6: Example XSL Transformation

price in US Dollars. Please note that for the case of simplicity we use a relatively simple example. Using XSLT even more complicated mappings can be defined, which explanation is out of scope of this paper.

As shown in Figure 6, the Euro metrics 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 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 behind. Thus, even slightly different SLA templates can be translated into each other.

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 *Vienna Service Level Agreement Framework - 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 [9]. 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-

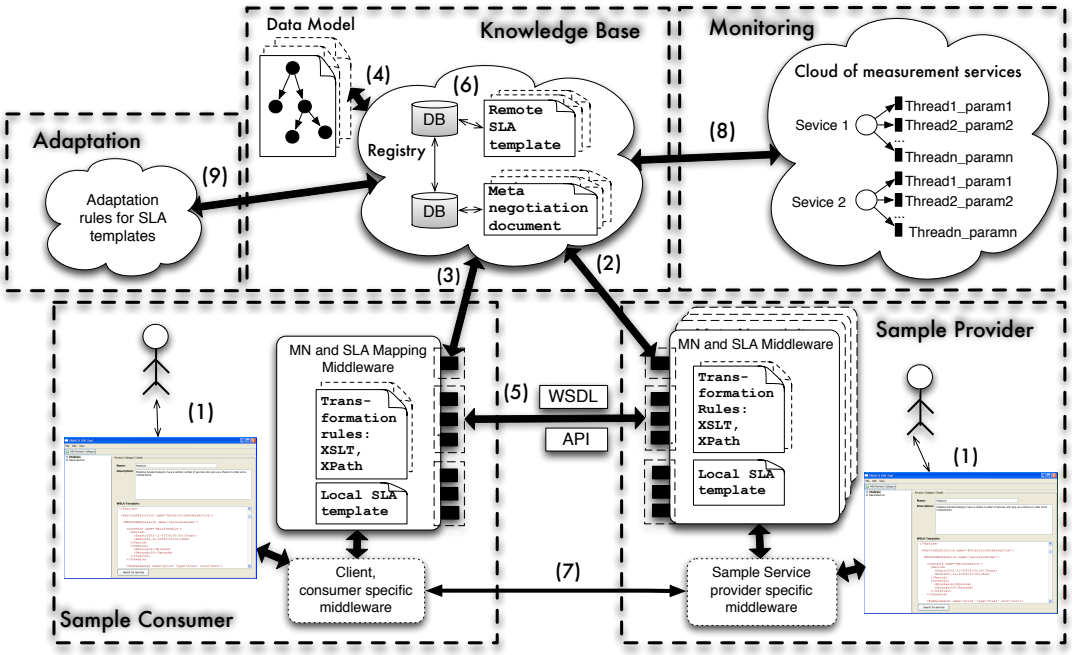

Figure 7: Extended VieSLAF Architecture with Monitoring and Taxonomies

parameters in SLAs. Furthermore, we discuss basic ideas about the adaptation of SLA templates.

We describe the *VieSLAF* components based on Figure 7. As shown in step (1) in Figure 7 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 and submit it to the registry. MN and SLA mapping middleware on both sides (provider's and consumer's) facilitates 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).

In the next section we discuss the architectural components and a sample case study, whereas first experimental results on *VieSLAF* are presented in [8, 9, 7].

6.2 VieSLAF components

As shown in Figure 7 the major *VieSLAF* components are knowledge base, components for monitoring and adaptation as well as MN and SLA middleware used by service provider and consumer. 5components are explained in more detail.

6.2.1 *Knowledge Base*

As shown in Figure 7 *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 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.

6.2.2 *Sample Provider and Sample Consumer*

In the lower part of Figure 7 a sample provider and a sample consumer are shown. Basically, a service consumer / provider consists of a client/service based middleware, MN and SLA mappings middleware facilitating the access to registries, and a GUI used for browsing remote templates.

Meta-negotiation Middleware.

The *meta-negotiation middleware* facilitates the publishing of the meta-negotiation documents into the registry and

the integration of the meta-negotiation framework into the existing clients (e.g. workflow tools) and/or service infrastructure, including, for example, negotiation or security clients. 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[2]. 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 in the context of 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). 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 into 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. The middleware can be easily plugged into existing middleware as we demonstrated in [7] with Aneka and Gridbus broker and Amadeus workflow framework.

SLA Mapping Middleware.

As already mentioned in Section 6.2.1 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 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 *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].

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

[3]Language Integrated Query

6.2.3 Monitoring Service

As depicted in Figure 7, 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 an 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.

6.2.4 Adaptation

Remote SLA templates should not be defined in a static style, they should reflect the provider's and consumer's needs. To make this possible, we implemented a first prototype of an internal registry's adaptation service, which can be used by consumers and providers as shown in Figure 7. They can specify *SLAParameters* which should be added into *SLA-Template* or choose some *SLAParameters* which they do not need in templates 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 learning capability property for each template category. The property defines how many requests for a specific *ParameterWish* has to be defined in order to add/delete *ParameterWish* to/from a *SLA-Template*. Whenever a new *ParameterWish* is accepted a new revision category of a SLA template is generated. All services and consumers are automatically re-published to this new revision. Also all SLA mappings will be automatically assigned to the new template revision. Old SLA mappings of the consumers and services will be 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 for each service new background monitoring threads are created and started.

7. CONCLUSION AND FUTURE WORK

In this paper we have 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 Grid/Cloud services bridging the gap between current QoS models and Grid/Cloud middleware and representing important prerequisites for the establishment of autonomic Grid/Cloud services. We discussed the approaches for meta-negotiation and SLA mapping representing partial implementation of negotiation bootstrapping and service mediation approaches. Furthermore, we presented the *VieSLAF* framework used for the management of meta-negotiations and SLA mappings. Using *VieSLAF* Grid service users can even monitor SLA parameters during the execution of the service calls. Finally, we discussed how SLA templates can be adapted based on the submitted SLA mappings.

As the next step of the FoSII project we plan to implement bootstrapping strategies where even consumer and provider, which understand different negotiation protocols and document languages can 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).

8. 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] A. P. Barros, M. Dumas. The Rise of Web Service Ecosystems. IT Professional 8(5): 31 – 37 , Sept.-Oct. 2006.

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

[6] I. Brandic, S. Pllana, S. Benkner. *Specification, Planning, and Execution of QoS-aware Grid Workflows within the Amadeus Environment.* Concurrency and Computation: Practice and Experience, 20(4): 331–345 John Wiley & Sons, Inc., New Jersey, March 2008.

[7] 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.

[8] 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.

[9] I. Brandic, D. Music, P. Leitner, S. Dustdar. *VieSLAF Framework: Increasing the Versatility of Grid QoS Models by Applying Semi-automatic SLA-Mappings.* Vienna University of Technology, Technical Report, TUV-184-2009-02.pdf, 2008.

[10] R. Buyya, C. S. Yeo, S. Venugopal, J. 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.

[11] 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.

[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] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

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

[20] 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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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)

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