Towards Autonomous SLA Management Using a Proxy-like Approach

Bastian Koller and Lutz Schubert
Höchstleistungsrechenzentrum Stuttgart, Allmandring 30, 70550 Stuttgart
koller@hlrs.de, schubert@hlrs.de

Abstract: This document presents an architecture for autonomous quality of service (QoS) management using a proxy-like approach. Service Level Agreements, as presented here, can be exploited to define certain QoS parameters that a service is to maintain during its interaction with a specific customer. As such, they provide a means of defining electronic contracts.

Within this paper, a way is presented to autonomously manage such forms of contracts. Since this requires an accordingly designed framework, the relevant objectives that have to be respected in this context have been analysed. Most of these requirements are explicitly put forward by service providers, respectively customers and cover issues that such a framework ideally should support. On this basis, an architecture for a management system has been developed. The main innovation of this framework consists in the introduction of a third-party "proxy" that allows for easy, "plug'n'play" management of SLAs.

Using the architecture, an implementation guideline has been sketched that will provide some insight into how such a system may be set up and reused.

The proxy-like approach presented here has been critically analysed and possible improvements for further development and research in this area have been identified.

Keywords: Service Level Agreement (SLA), Quality of Service, SLA Management, Service Oriented Architecture, eBusiness, Framework, Proxy

1 Introduction

In day-to-day business situations, consumer and service provider negotiate or agree on pre-formulated contracts in a written form in order to ensure that a certain quality of service is met by the provider, respectively that the customer does not assert unjust claims. Since this however is no *guarantee* for correct performance, the customer will generally want to compensate for possible losses from non-performance on service provider side by defining adequate penalties[1]. In order to prevent this, customer and service provider want to monitor the progress of service execution so as to react accordingly to possible deviations from the original plan – this however is generally associated with a high work overhead on both service provider and customer side. The concept of SLAs (Service Level Agreements) foresees the definition of electronic contracts that define specific quality of service parameters. Given the right system, SLAs would enable automatic observation of a specific service's performance and thus allow for timely reactions to critical deviations. With the respective set-up, the information thus provided can also be used to support self-management(cf. [FKS05]) functionality.

2 Towards Autonomous SLA Management

There have been various approaches towards an electronic management of such contracts (e.g. [D$^+$04]) to allow for autonomous observation whether the agreed upon terms are *maintained*. Most of these approaches, however, are either mere theoretical models without guidance towards realisation or completely dismiss important user requirements. We therefore want to present with this document a component-based architecture that could easily be implemented and is in principal independent of the underlying SLA-representation [L$^+$03], [L$^+$05].

It is hence relevant to first identify the main objectives forming the user requirements (cf. [MN]):

2.1 Main SLA Objectives:

The main purpose of autonomous service level agreements consists in automatic enactment and observation of the QoS parameters, i.e. **monitoring** the performance of a service provider, and comparing it with previously agreed terms (**evaluation**). This information may then be used for fair **billing** of the service usage.

[1] Ideally, these penalties would be high enough to fully compensate financial losses of the customer - in real business cases, however, hardly a service providers will want to risk extreme high penalties

2.2 Indirect SLA Objectives:

In addition to these obvious functionalities, SLAs may also be used as a criterion for the **identification** of services that fulfil a given set of (desired) QoS-descriptions. By publishing a set of potential SLAs that can be maintained by the service, a potential customer may identify whether the service meets his/her needs or not. Such information can also be used for an autonomous discovery process.

Since the concrete terms are generally up to **negotiation**, to e.g. agree upon the cost of service, such potential SLAs are just guidelines for the identification process. Negotiation, too, may be enacted autonomously, when the respective goals of the negotiation (e.g. "no cost higher than x") are defined.

2.3 Business Objectives:

A SLA management system may *under no circumstances* make data available that is protected by enterprise specific **privacy** issues, like e.g. the firewall settings of a specific computer within the network. Though this is generally a security issue that needs to be solved separately to the SLA management system, design of the latter should allow for the respective requirements and not make security realisation more difficult. In general, the system should not allow to access data that is not explicitly made available by the agreed SLA definition.

Similar to this, **internal policies** have to be respected by the system – besides for security issues like the ones above, this also includes statements about SLA-restricting behaviours, e.g. limitation of the performance so as not to overheat the system.

These issues are of particular relevance in connection with **self-management** of services: a service provider may want the system to automatically adjust the service behaviour (within a given limitation) in order to avoid SLA violations, thus always providing full QoS compliance[2].

Since measured performance may have an influence on billing the service usage, customer *and* service provider would furthermore like to keep the system **neutral** so as to not result in biased evaluation of the data, as this would improve, respectively reduce the profit unfairly.

2.4 Additional Objectives:

Most approaches require the service provider to install or implement components on their system that allow for realising the respective functionalities – obviously, most companies

[2]Obviously, a SLA management system can not provide the means of autonomous self-management, but it can provide the relevant information to a given administrative instance so as to allow and support SLA-related management.

or business entities will be reluctant to any changes to their systems which may cause worse performance and less compatibility with existing customers and/or business partners. Accordingly, a SLA management system should have as **little impact** on the existing infrastructure as possible.

Along the same line, implementation and maintenance of the system, including adaptations to different SLAs should be easy and straightforward, thus requiring as **little administrative overhead** as possible.

3 A SLA Management Architecture

In order to fulfil the previously identified requirements (cf. section 2), we recommend an architecture according to figure 1.

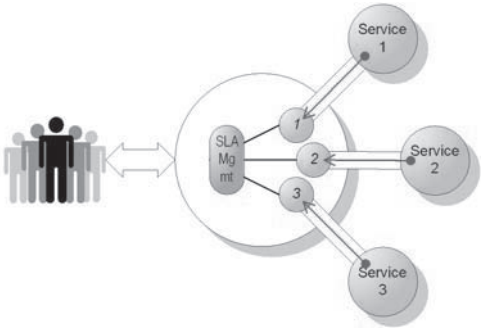

Figure 1: High level idea

Our approach foresees a clear distinction between service providers, customers and the actual SLA supporting system, by introducing an intermediary, proxy-like component that can take over most contract-related tasks. Likewise, third-party entities can exercise this role, thus allowing a) as little influence on the service providers as possible and in particular b) independent and neutral evaluation of the service's performance.

Obviously, the service providers need to support the proxy-tasks with the respective configuration of their infrastructure, yet the general idea ensures that this impact is kept to a minimum, as shall be described in more detail below.

Introduction of such a proxy could mean that the customer *may*[3] get only the address of the proxy and no more of the service itself – this would imply, that all communication between customer and service has to take place via a given proxy service.

[3]Note that this is just one possible approach towards designing the communication interaction. Alternatives consist in having the service provider redirect SLA related issues or using the SLA Management service as a stand-alone third-party service that gets is role assigned by service provider and customer. We chose the "proxy"-approach here in order to allow for further enhancements of the system by adding supporting services for security, policy control etc. This would decouple the actual service from supporting tasks as much as possible, thus keeping the service provider's implementation and configuration effort to a minimum.

3.1 Architecture Overview

A more detailed view of the architecture is provided in figure 2.

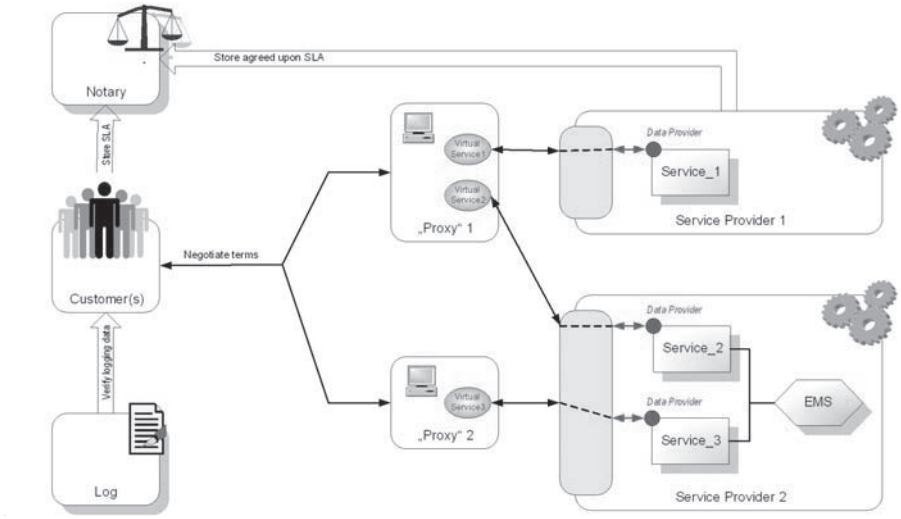

Figure 2: Architecture Overview

3.1.1 The "Proxy"

As mentioned above we see the proxy as an interface between customer and service provider. This way, a customer does not directly interact with the service, but via this interface that needs to be able to forward all messages to the service provider – this, however, is not a necessity, as already noted. Accordingly, the web service interface and the proxy could be regarded as linked together, since on the one hand the proxy serves as message forwarding instance and on the other hand has the responsibility to evaluate the service's behaviour. From the customer's point of view, the proxy is identical to the actual web service interface and thus *has to* allow the same operations[4].

The proxy-approach allows for "neutral" evaluation of the SLAs, i.e. whilst service provider and even customer may have an interest in changing the SLAs to their own profit (triggering violations to diminish costs or retaining notifications of SLA breaches etc.), the party providing the proxy has no own interest in making any adaptations.

[4]Note that a proxy can serve multiple service providers at the same time.

3.1.2 Service Provider Side SLA Management

Obviously, the proxy has no (and should not have) direct access to the actual system status of the provided service – hence the service provider *has to* extend his services so as to allow for status monitoring. Generally, the required tools are available for most users (like e.g. WMI[TC03], [Mic] which comes with the latest Windows Versions or Ganglia[MCC04] which is an open source client for Linux). Using the proxy the principle of little impact is maintained. Since SLA specifications differ from the data provided by these tools, some "parsing" functionality needs be supported (cf. below). It is implementation-dependent whether the proxy or the service provider itself should take over this task, though, generally, the service provider will be responsible for this so as to a) prevent sensible status data to be accessible without having been converted accordingly and b) to keep the effort of configuring the proxy to a minimum[5].

"Local" management furthermore can be extended to allow for "preventive" SLA management: by lowering the parameters agreed upon in the SLA, a service administrator or self-management system could be informed of an increased violation risk *in time*, thus being able to take corrective measures. Again, a service provider *may* use the proxy for this functionality by providing an additional, "preventive" SLA, this way saving the integration effort – generally, it may be assumed though, that a company would want to keep the respective process short and quick, without having to access external parties first.

3.1.3 Third Parties

A third party "notary" may store the SLA during the actual usage of the service by the customer for reference. This way it is ensured that potential differences in the two SLAs can be solved by a neutral party[6].

Likewise, third party "logs" could keep an account of all the violations that have occurred during the actual operation and serve as a reference to e.g. support billing on the basis of service performance.

3.2 Interaction Description

In the following sections, the overall interactions from service discovery up to billing shall be outlined so as to provide some insight into the processes involved. Note that we shall re-examine these processes from an implementation point of few in the successive chapter.

[5] We do propose though to design a proxy that is capable of "understanding" the most common data providing tools, namely WMI and Ganglia, so as to realise the highest flexibility and minimise the implementation- and adaptation-effort on the service provider side even further.

[6] A nowadays more common approach consists in enacting a complicated **signing** protocol that leads to an encoded SLA that may be read, yet not altered. Here the Notary may act as a "witness" or actually perform the signing process based on keys provided by the respective parties – refer to e.g. [Nex] for more information

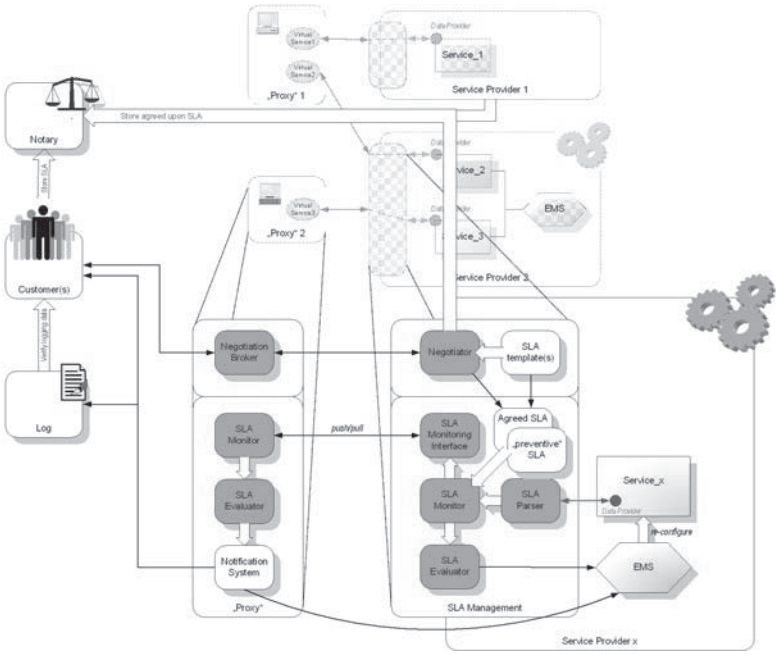

Figure 3: Components of the respective systems

3.2.1 Identification

A (potential) customer may query UDDI-like repositories with a description of the required service to identify potential service providers. We assume here that some link to the service provider's SLA template repository is retrieved with this information, so that the customer is able to request the SLA templates from the providers of interest. Such templates represent the quality that may be *principally* maintained by the service – on basis of these, a customer may identify the most suitable service for his/her purposes, i.e. the one which can fulfil the QoS-requirements best[7].

In most cases it will be necessary to negotiate the actual SLA parameters, as the service provider will allow for some flexibility with respect to the quality and the obligations that the services may fulfil (e.g. lower price for less memory usage etc.) In addition to this, a template is no *guarantee* for a provider being able to maintain the described quality at the time of demand – accordingly, the negotiation process also serves the purposes of verifying availability. Since the customer is only able to communicate with the proxy, the negotiation broker of the proxy needs to provide a linkage between the customer and the negotiator installed on the service provider.

[7]It may occur, however, that *no* service meets all the requirements, in which case an adaptation of the constraints needs to take place.

Once an agreement is reached, the negotiated SLA will be stored in Notary and/or signed so as to prevent further modifications. Theoretically the service is now ready to be used by the customer, though generally this is not a requirement. In some cases, it might be that the customer wants to use the service at a later point in time (e.g. a month, a day) which will obviously influence the negotiation process and entails that the service needs to be reserved. In the given case however we will assume the service to be used immediately.

3.2.2 Configuration

In order to fulfil the agreed upon quality of service, the service provider configures his service accordingly – this is common praxis and shall not be explained in detail here.

As mentioned before the customer wants to receive status reports in a certain time-interval. To this end, performance data is gathered regularly from the service provider and compared to the agreed upon quality terms. Violations of these terms will lead to a notification being sent to all involved parties (customer and service provider) who may take according actions (cf. below).

Since the service provider will want to prevent such violations, he/she may make use of the SLA management components for violation prevention – we may thus differentiate between "preventive" and "business" monitoring:

In figure 3 the monitoring and evaluation components on the service provider side may take over responsibility for **preventive monitoring**. Therefore it takes a SLA (the so-called "preventive SLA") with SLA Parameters modified in a way that will indicate a potential violation, yet not lead to a contract breach in the original SLA – generally, this is achieved by slightly lowering the violation-thresholds.

Accordingly, these internal components may take over the task of **business monitoring** too, where breaches of the original SLA are to be identified. However, the customer may not *trust* the service provider to evaluate the data correctly ("neutrality"), whilst the latter will not want to expose (sensible) system data to the customer ("privacy").

To circumvent these (and related) problems (cf. chapter 2), we introduce our "Proxy" to the SLA Management structure: it will query the monitoring interface of the service provider regularly to update status information and compare it to the agreed upon SLA.

3.2.3 Enactment

Status information, as provided by any data provisioning unit, is generally not identical to the parameters as defined in the SLA document, some form of conversion is required to make the data understandable for the evaluator. Since this conversion is directly dependent on the data provider and since the service provider may want to "neutralise" sensitive information, it is logical to assume that the respective parsing capability is supported at service provider's side. Otherwise, a SLA parser capable of understanding every system parameter provided by the most common data providers would have to be realised on the proxy's side. Accordingly, the "preventive" monitoring may act upon these converted data, too, i.e. use the same document-structure as for the original SLA.

This status information will be validated by the SLA Evaluator components to identify violations.

3.2.4 Violation handling

Depending on "preventive" vs. "business" monitoring, the reactions to such violations vary:

In order to prevent the actual contract breach, the service provider will want to re-configure his system so as to lower the respective risks. We hereby assume that some form of intelligent "Execution Management Service" may achieve this, by e.g. moving the process to a different host or freeing resources.

In the case of actual business violations, different reactions are thinkable and need be agreed upon during SLA negotiation – however, the following are generally considered the most appropriate ones:

If the cause for violation is only temporarily (i.e. not leading to permanent contract breaches, like e.g. when the service's system is down) and/or may be remedied by the provider, the customer may be satisfied with applying a (financial) penalty as defined in the SLA.

However, if a permanent (or constantly recurring) decrease in the service's quality is to be expected, the customer may want to either lower the respective parameters, most likely reducing costs, too. Or replace the service entirely by one offered from another provider who may maintain the quality better. This decision generally depends on how much the service has already proceeded with its progress and how much impact this loss in quality is likely to have, as identification and negotiation will result in a delay of the process – in particular, if the process provided would have to start again from scratch, i.e. if the progress so far may not be moved to the new provider.

Sometimes the service provider may request verification of a violation. For example, if the evaluator detects an violation but the service provider itself has not observed one. In such a case, the above mentioned Notary may serve as a "neutral" party to solve the issue.

Generally, all violations will be stored in a log for later reference (e.g. for billing purposes where a violation will lead to a reduction in pricing).

4 Tackling a Simple Implementation

Within this chapter we want to sketch a potential implementation of above described architecture - this bases partially on results from IST projects (e.g. [Nex], [Tru]), as well as own experiments and implementation efforts, and partially on theoretical reflections. Note that this is just one approach towards realisation of the proxy architecture and hence may not be considered ideal for all environments.

For demonstration reasons, we will present an implementation that takes the proxy-structure

to an extreme where the service provider disobeys all confidentiality issues and fully "trusts" the proxy provider to maintain all privacy, security and related policies.

4.1 Technology Recommendation

Our implementation efforts so far have been basing on a Windows .NET platform, as - at the time of writing this - more tool-support can be found for this platform than for the Java/Linux domain[8]. We furthermore make use of the WSE 2.0 by Microsoft (http://msdn.microsoft.com/webservices/building/wse/default.aspx) and the WSRF.NET library by the University of Virginia (http://www.cs.virginia.edu/ gsw2c/wsrf.net.html).

Accordingly, we use standard SOAP messaging with WS-Addressing and WSRF extension. As said, this does not represent requirements, but a mere recommendation: by using WS Resources rather than ASP.NET state management, we allow for more freedom with respect to the linkage between proxy and service provider, WS-Addressing enables usage of Endpoint References to identify the resource etc.

Beyond this, we recommend the usage of the WSLA language [L+03] for defining SLA documents, as it currently provides more functionalities than WS-Agreement - we will provide a sample document below. Since the implementation bases on the Microsoft Windows operating system, we assume the service to be monitored being deployed in a Windows environment, too. This allows us for exploiting the Windows Management Framework (WMI [Mic]) as a data provider, though we would generally recommend the Ganglia toolkit [MCC04] for the typical HPC environments.

Unfortunately, we can not provide a full tutorial on implementing a SLA Management proxy service here and thus will have to assume that the reader is familiar with the Web Services concept and most of the popular WS-* standards, in particular WSRF [GKM+05]. The description will have to be limited to a set of essential operations, were we can only give an indication of how to realise all further functionalities as described in the text above.

4.2 Detailed Structure

According to our set-up, each subsystem of the SLA Management and the Proxy component is realised as a Web Service entity. Though this approach implicates a performance loss due to SOAP message interaction being slow compared to direct procedure calls, we gain flexibility with respect to set-up: as interaction may thus take place over systems' borders, the administrator of the respective hosting environment may deploy the services at any convenient location.

In figure 4 we provide an overview over the classes involved in the simple set-up. Note that we skipped details on a notification layer here to keep the picture simpler. Accordingly, it has to be assumed that all communications between SLA Management and Proxy are

[8] We may mention the Globus Toolkit (http://www.globus.org/toolkit) here for the sake of completeness.

supported by a notification-/ messaging-layer that allows for all relevant functionalities, like message encoding, authentication, topic association etc.

For simplicity's sake, we furthermore assume that all classes are derived from the WSRF.NET base class (not included in the figure) and thus provide the means to maintain a permanent state. Accordingly, the SLA documents relevant for the respective subsystem are stored in a back-end database and may be accessed by the classical WSRF means.

4.2.1 SLA Management Structure

Since we want to present an extreme "plug'n'play" scenario, we hereby assume that the SLA Management structure acts mainly as a means of exposing the system status in a specific form (to some known entity). Accordingly, our SLA Management will not maintain a SLA on its own and not provide "real" monitoring functionalities - nonetheless, all four main components of the SLA Management structure are provided: the SLA Monitor in combination with a SLA Parser, the Monitoring Interface and the Negotiator. Optionally, we may provide local Evaluation functionalities to allow for "preventive" SLA management, which we will omit here though, as this may be handled via the proxy, too, by providing an additional SLA document with only the service provider subscribing to the violation events.

- SLA Monitoring Interface: Theoretically, the SLA Monitoring Interface may be realised as a simple WSDL interface to the SLA Monitor component, yet some service providers may want to realise additional functionalities with respect to manageability of their services, which functionally does not belong to monitoring (like e.g. manipulating the state). One of the functionalities assumed here consists in "neutralising" sensitive data, which could be achieved by turning concrete values into a relative measurement, like e.g. instead of the amount of available storage just the percentage of how much of the agreed upon size actually is available at the moment.

  In our scenario here, however, the only functionality of the interface consists in exposing the status related operations of the SLA Monitor.

- SLA Monitor:

  Normally, we would maintain a copy of the SLA locally and base all interactions on a reference to the respective SLA - however, in order to demonstrate the full functionality of the proxy support, we assume here that no local SLA is maintained. Accordingly, the SLA Monitor may only provide the status of specific parameters and not of the whole SLA document. Note that these are not assigned as ResourceProperties in order to keep the service structure more flexible. This implies that the proxy will have to "know" the names of accessible SLA Parameters – as these are defined via the SLA document, however, this should pose no problems.

- SLA Parser: Since SLA parameters generally do not map one-to-one to the parameters provided by the respective Data Providing units and accordingly, some form of transformation will be required, too. One must assume that each host system

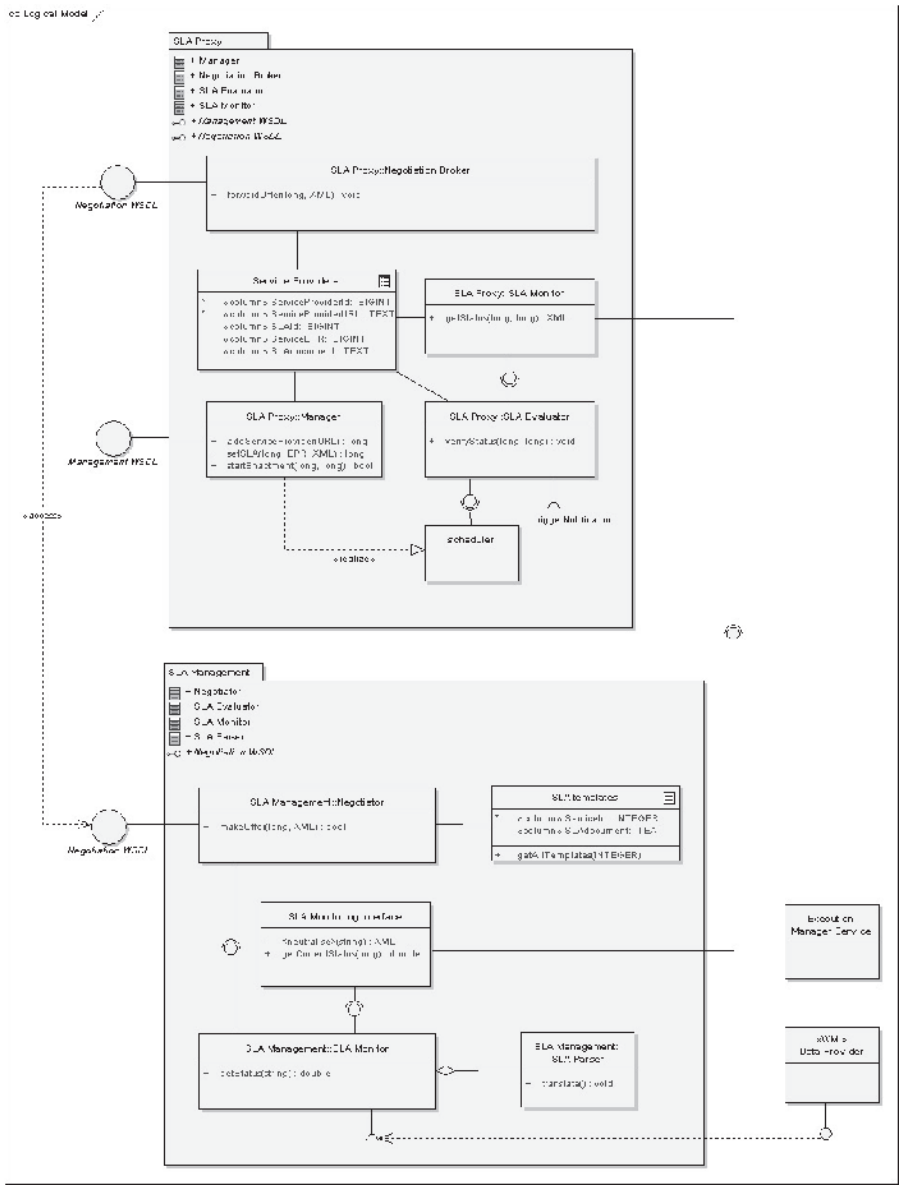

Figure 4: Class diagram for an implementation scenario

provides its own Data Providing unit and accordingly will expose different System Parameters that need according transformation functionalities – likewise, each connection to a host system would require its own parsing capability.

Without loss of generality, we will assume here that only one hosting system exists and hence exactly one data providing unit is in place. Accordingly, we shall integrate all parsing capabilities into the SLA Monitor.

- Negotiator: As described above, Service Level Agreements need to be negotiated before they may be enacted. In the case here, we will assume a simple one-phase-negotiation, where an offer is simply declined or neglected (cf. above).

- SLA Evaluator: Our local Evaluator validates the status according to a reference SLA, i.e. with every status update we compare the new status to some "preventive" values. This implicitly assumes that "preventive" SLA and enacted SLA observe the same parameters, which is not necessarily the case: in principal, the local Evaluator can be used as part of a rather independent validation circle.

4.2.2 Proxy Structure

According to our extreme approach, the Proxy provides all relevant SLA management functionalities, i.e. it maintains the agreed upon SLA, monitors the parameters by pulling them from the service provider and validates the status. Optionally, a Negotiation Broker may act as an intermediary during the negotiation process between customer and service provider.

- SLA Monitor: Due to set-up the proxy SLA Monitor will take over responsibility for aggregating and transforming the data according to the metrics as defined in the SLA document. Accordingly, it will request values for the "low-level" parameters (i.e. the ones that are not functions of other parameters) from the service provider's monitor and apply the required mathematical functions. Such functions also cover the building of average values over time, which implies that the results are stored at the proxy-side.

- SLA Evaluator: As described above, the proxy validates the monitored parameters according to the initially agreed SLA and – in the simplest case – triggers violation notifications to customer and service provider, in case of a contract breach.

- Negotiation Broker: In our scenario, we assume that the service provider wants to be completely hidden from the customer, so that the actual location will be unknown. Accordingly, the customer would have to interact with an intermediary broker during the negotiation process. The task of the broker thus consists in nothing else but forwarding the messages.

  In more elaborate versions, this will also serve the purpose of translating between different technologies, like e.g. between WS-Agreement and WSLA.

4.3 Potential Set-up

Assuming that there exist a service provider with the above described SLA Management system deployed, as well as a proxy provider offering components as sketched, the set-up of the whole system would proceed as follows:

4.3.1 Identification and initial binding of the proxy:

Initially, the service provider would have to bind a proxy fulfilling its needs, so as to offer SLA management along with its service. We assume here that such a "proxy provider" may e.g. be discovered via a simple UDDI repository. Once identified, the service provider requests to be added to the list of customers by providing his/her contact details – from now on, the service provider may publish his service description, along with a set of SLA templates and the proxy as a contact point.

Note that we realise the data set of the Proxy as a "global" back-end resource, so that it can be accessed independently of a specific EPR, just like a typical registry.

4.3.2 Negotiation via a proxy:

SLA templates, in their simplest form, are identical to SLA documents with the assigned values not as obligations, but as optional configuration-settings. Likewise, a service provider may publish two SLAs, one stating a `CPUclockspeed` of 3 GHz and a `price` of 2 cents per minute, and another one with 6 GHz and 5 cents per minute usage.

In such a case, any customer just has the choice between the two options and no further influence on the quality of service to be provided – an according negotiation process would be simple and straight-forward: the customer sends the template of his choice to the Negotiation Broker, who forwards it to the SLA Negotiator. The latter will query its Monitor to see whether the host may currently fulfil the stated QoS and answer accordingly with an accept or a decline, which is forwarded from Broker to customer[9].

Since a provider may offer more than one service, a unique id will denote the service in question and needs to be passed as context information of the WS-Addressing header along during the negotiation phase.

4.3.3 Final binding of the proxy:

Once the SLA has been agreed upon by both sides, the Service Provider instantiates the respective service and provides the final SLA along with the EPR of the service to the

[9] We refer to this simple process as a "one-phase-negotiation". More complex negotiation processes will allow the customer more influence on the QoS, e.g. by providing *ranges* of optional values for a specific parameter or by leaving the field completely empty. Accordingly, the service provider may suggest alternative configurations, if the current is not suitable (can not be fulfilled), basing on the initial "offer" by the customer. Obviously, this may lead to a n-phase negotiation process, were differently filled templates are passed back and forth until either one of the parties agrees, declines or a time-out is reached.

proxy. This way, all further interactions between Proxy and Service Provider will refer to the specific service. Accordingly, as each Provider may use the proxy for more than one business set-up, i.e. for more than one SLA, a unique id will identify the SLA in question when enactment is triggered (cf. below).

Note that we may instantiate Evaluator and Monitor on proxy side with the respective set-up as resources, however, in this simplified approach, we will have them refer to the back-end database for the SLA information.

4.3.4 Enactment:

Once the service in question is triggered for execution, the SLA Proxy has to start monitoring and validating the performance. With triggering the enactment on proxy side, a "scheduler" process will be initialised that calls for validating (and implicitly updating) the service's status on schedule, as defined in the SLA document.

When the SLA is violated, a notification event is triggered at the underlying notification layer (not in the diagram – in our case provided by WSRF.NET) which is forwarded to all subscribers – normally the customer and the Service Provider, who may then react to that notification as necessary. In case we set up an additional "preventive" SLA, such a reaction will generally consist in reconfiguring the service so as to prevent the contract breach (cf. "Interaction Description" above).

5 Summary and Outlook

The approach presented in this paper provides a SLA Management System that allows for easy modular usage. As opposed to many other designs, the proxy-like architecture will guarantee a clear distinction between service provider, customer and the actual SLA Management functionality, thus allowing for easy "plug'n'play" and hence observes the "little impact" objective. Since common system monitoring tools ("data providers") like WMI and Ganglia are easily available, configuration of the management system may be realised by simple deployment of the SLA document, which keeps maintenance overhead low.

By introducing SLAs with lower thresholds and additional information that reflect critical state-changes, the evaluation results can be used for introducing self-management of the service host. Since SLAs may be used to cover any issue as long the related value can be measured, the respective feedback-loops in interaction with any self-management instance (in the simplest case the system administrator) can be exploited for autonomous enactment of internal policies.

Due to the set-up, third parties can take over the proxy-service and thus enable "neutral" monitoring and evaluation of the service performance.

However, the disadvantage of any SLA management system can not be completely overcome: even in the presented case, the service provider needs to configure his/her system

so as to allow monitoring from the outside. Though e.g. WMI allows for remote access to system status information, no company would want to allow such access. Accordingly, an interface needs to be supplied that provides the accessible information and hides all other data, thus enacting internal policies and privacy issues. Such an interface, of course, allows for manipulation of the data again and hence poses a similar threat with respect to confidentiality of the data.

Nonetheless, as this interface needs to be accessed solely by the (well-known) proxy-instance, it is easier to secure the access in order to minimize this thread, as when data-access is provided to all transaction partners publicly. Obviously, it must be presumed here, that the service provider can "trust" the proxy-service to not manipulate the data itself.

Future work will have to concentrate on implementing the respective architecture and especially on providing an exhaustive specification of widely acceptable SLA parameters to reach something like a global understanding of SLA descriptions. Semantic web technologies (ontologies, dynamic logical derivations etc.) will be exploited in the long run to provide more flexibility in the definition of terms.

Other issues to solve include the calculation of dependencies between different SLAs, like e.g. when a company offers services that themselves are composed of services provided by other entities. Also, the possible "richness" of SLA templates required for identifying services, respectively the enhancement thereof will have to be examined carefully so as to allow for more information to be provided.

Current IST research projects are looking into these issues, trying to come up with solutions that cover additional criteria, like security-related factors, scalability etc. - issues we could not delve into here further for lack of space.

Autonomous SLA management certainly has some further obstacles to overcome, before it may replace a human actor: most of these related to the intelligence of the system with respect to generating and validating SLAs on its own, respectively to allow for autonomous adaptation of the actual service to meet the QoS terms and to maintain these. Future research projects will shed more light on these issues.

References

[D+04] Asit Dan et al. Web Services on demand: WSLA-driven automated management. *IBM Systems Journal*, 43(1), 2004.

[FKS05] Ian Foster, Hiro Kishimoto, and Andreas Savva. Open Grid Service Architecture Version 1. Technical report, GGF, 2005.

[GKM+05] Steve Graham, Anish Karmarkar, Jeff Mischkinsky, Ian Robinson, and Igor Sedukhin. Web Services Resource 1.2. Technical report, OASIS, 2005.

[L+03] Heiko Ludwig et al. Web Service Level Agreements (WSLA) Language Specification. Technical report, 2003.

[L+05] Heiko Ludwig et al. Web Services Agreement Specification. Technical report, 2005.

[MCC04] Matthew Massie, Brent Chun, and David Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7), July 2004.

[Mic] Microsoft. Windows Management Instrumentation. `http://msdn.microsoft.com/library/en-us/wmisdk/wmi/wmi_start_page.asp`.

[MN] Anbazhagan Mani and Arun Nagarajan. Understanding quality of service for Web services. `http://www-128.ibm.com/developerworks/webservices/library/ws-quality.html`.

[Nex] IST-2002-2.3.2.8 NextGrid. The Next Generation Grid. `http://www.nextgrid.org`.

[TC03] Craig Tunstall and Gwyn Cole. *Developing WMI Solutions: A Guide to Windows Management Instrumentation*. Addison-Wesley, 2003.

[Tru] IST-2002-2.3.1.9 TrustCoM. A Trust and Contract Management framework enabling secure collaborative business processing in on-demand created, self-managed, scalable, and highly dynamic Virtual Organisations. `http://www.eu-trustcom.com`.