Towards Service Level Management in Clouds

Philipp WIEDER[1], Peer HASSELMEYER[2], Bastian KOLLER[3]
[1]*Service Computing Group/ITMC, TU Dortmund University,*
August-Schmidt-Strasse 12, 44227 Dortmund, Germany
Tel: +49 231 7552767, Fax: +49 231 7552763, Email: philipp.wieder@udo.edu
[2]*NEC Laboratories Europe,*
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
Tel: +49 6221 43420, Fax: +49 6221 4342155, Email: peer.hasselmeyer@neclab.eu
[3] *Höchstleistungsrechenzentrum Stuttgart, Universität Stuttgart,*
Nobelstraße 19, 70771 Stuttgart, Germany
Tel: +49 711 68565891, Fax: +49 711 68565832, Email: koller@hlrs.de

Abstract: Current Service Level Management solutions for distributed e-Business infrastructures are designed to work in specific domains and are bound to particular languages and protocols. With the transition to Cloud-like environments, it became evident that migrating such solutions from the Grid domain is not a trivial endeavor. Although the various distributed systems concepts may share similar technological foundations, the business and usage models differ significantly. Everything-as-a-Service is replacing the Everything-is-a-Resource paradigm, requiring a different view on Service Level Management. This paper proposes a novel approach that suggests a dynamic solution featuring function-replacement at run-time in order to accommodate the different needs and requirements of service providers and consumers efficiently and effectively. Furthermore, we envisage that the concept of plug-ins for Service Level Management will create a new market for trading pluggable service functionality, most likely provided by SMEs operating in niche markets.

1. Introduction

A large variety of industrial and academic research efforts have been conducted to provide Service Level Management (SLM) capabilities for distributed e-Business infrastructures. Having focussed on Grids during the previous decade, we now face Cloud Computing as the stimulus of IT service business as well as distributed systems research. In general, *Clouds* introduce a completely different usage model and a higher level of technical abstraction in comparison to Grids, thus they lessen the burden users often experience with Grid infrastructures' complexity.

Regarding Service Level Management, Grid-based solutions do not provide the abstraction level and generic support necessary for different business models if transferred unchanged to Cloud infrastructures. Furthermore, most of the Grid solutions are still too static in terms of their functions and are not interoperable with other frameworks that are not based on the same technology, something which Clouds promise to change. Therefore it is essential to develop tailored concepts and methodologies for Service Level Management solutions in Clouds, and to realize solutions that reflect the paradigm change in distributed computing and that do not introduce additional complexity to the business user.

2. Objectives

This paper presents a novel approach towards a flexible and interoperable Service Level Management solution by combining a plug-in approach with the concepts of the Open Model methodology [1]. This methodology propagates the application of the development process of open source software to the context of conceptual modelling. This implies the production of open reference models that can be copied, used and refined in a collaborative public process.

The core idea of our work is the combination of plug-ins and the context-aware Service Level Agreement (SLA) model introduced by Koller in [2]. The definition of context is aligned with the one provided by Dey [3] in 2001, who defines context as "*any information that can be used to characterize the situation of an entity.*" This definition exactly describes the importance we predict context-awareness to have on the behaviour of future systems. Based upon the use cases presented in Section 3, the context of different domains is characterized and the respective requirements are derived. Mapped onto the approach we describe here (cf. Section 6) and building upon existing Service Level Management frameworks, we propose a flexible, pluggable and interoperable solution leading towards smart *Middleware-as-a-Service* (MaaS) complementing the *Everything-as-a-Service* (XaaS) landscape.

3. Requirements from a Business Perspective

To provide a good basis for validating the proposed solutions, we have chosen two concrete industrial use cases from projects we work on that provide complementary requirements.

3.1 The eHealth Use Case (Software-as-a-Service)

The "Goodwill" Hospital in Castle Rock gets a new patient with a referral from his family doctor who suspects a cerebral bleeding. First examinations with a computed tomography confirm this suspicion and lead to the decision of performing a neurosurgery within the next 12 hours.

To ensure the best treatment, the surgeon wants to plan the surgery as detailed as possible in advance, and he additionally wants to be able to optimize his strategy during the surgery with the help of additional simulations. The duration of the surgery is approximately 5 hours, therefore a maximum of 7 hours is left for preparation. The hospital would like to use the HemeLB [4] application, which provides the required simulation methods. As the hospital's IT infrastructure does not include HemeLB, the hospital needs to find a provider who offers this software with the necessary specifications and who has free capacities to perform the simulation immediately. In addition, the access to the Software as a Service offer needs to be guaranteed during the surgery to allow for additional simulations whenever necessary.

3.2 Car Crash Simulations (Platform-as-a-Service)

A car manufacturer is planning to develop a new family car. As a basis, a predecessor of the new car is taken and changes are planned to address new and extended features for families.

After an initial planning phase the first prototype is modelled and the next step is to simulate the behaviour of this car in case of a crash. This is not a single simulation run, as several simulations are needed while adapting the design of different components like air-bags or the roll-over bar according to the simulation results.

This is not a trivial task, as to make different simulations comparable, the environment needs to be permanently stable, including Solaris v10 OS, Java v1.6 update 7, and LS-DYNA V971 R3.1 (R3.43919). Obviously, the car manufacturer needs the guarantee from

the provider of the IT environment that the requested simulation platform will be available unchanged during the whole duration of the agreed service offer. In addition, there are strong requirements on privacy and data protection, as the simulation inputs and outputs are highly sensitive datasets, which are of high interest to the manufacturer's competitors.

4. Methodology

Methodologically, the developments described below are based on the Service Level Agreement lifecycle as presented by the TeleManagement Forum (TMF) [5]. As shown in Figure 1, the lifecycle covers six different phases, namely Development, Negotiation (including SLA template discovery), (Service) Implementation, Execution, Assessment and Termination.

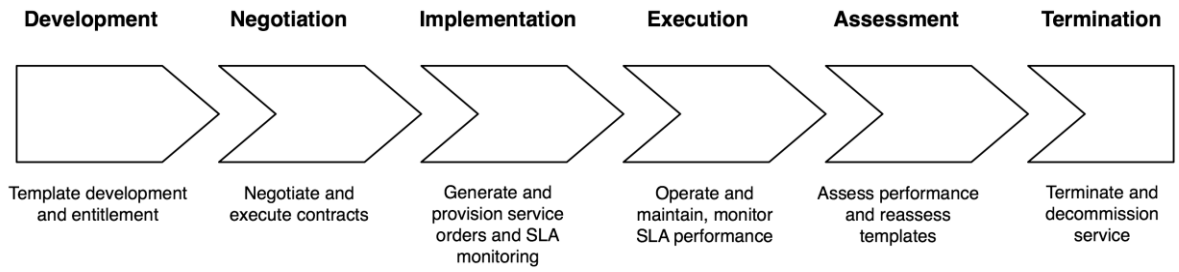

Figure 1 – The Service Level Agreement Lifecycle [5]

Based on this lifecycle, but also on the analysis of requirements on SLAs in Cloud-like environments (as, for example, reported in [6], [7]), we present an evolution of currently existing SLA management approaches. The aforementioned reports stress the need for enhanced capabilities in Service Level Management, especially in terms of SLA representation, but also regarding their processing with respect to the mapping of high-level business terms to low level infrastructure terms. According to the ETSI Cloud analysis [6], "*in order to match the technical characteristics of cloud infrastructures with an appropriate commercial environment, a flexible, lightweight and dynamic framework for management throughout the SLA lifecycle is required.*"

To address these issues, we present in this paper an extended concept of context-aware services and plug-ins (as introduced in [2]), and we apply this concept to Clouds. First, state-of-the-art frameworks implementing the SLA lifecycle are analysed to derive an abstract and generic Service Level Management model. The model is then evaluated in the light of Clouds and XaaS, leading to an enhanced model which constitutes the transition from Grids to Clouds. Although Service Level Management is the research area chosen to validate our approach, the solution presented here is applicable to most other aspects of current service frameworks and Clouds such as security, workflow management, and others.

5. The State of the Art in SLA Management

Service Level Management and especially Service Level Agreements have been the focus of many research and development activities, either industry-driven (e.g. from the telecom and service industry) or as part of national or international research co-operations (for example European projects).

Despite these efforts, SLAs are currently still mostly handled manually. Automation of the complete SLA lifecycle is key to the commercial viability and success of SLAs in Cloud environments. This automation is still a hot research topic, and so far no significant uptake in the industrial and academic worlds has happened due to still existing showstoppers.

For representing SLAs electronically, which is a required basis for automation, a number of languages have been developed such as WSLA [8] and the Open Grid Forum's WS-Agreement [9]. Although they both have similar characteristics, they address different viewpoints of SLAs and can be seen as being complementary. The BREIN [11] or TrustCoM [12] projects have successfully shown their integration.

The abovementioned representation languages define only templates or high-level languages for describing SLAs. The actual domain-specific content inside the SLAs, i.e. the specification of guarantee terms, is left open. This is desired and good practice, as it allows for the integration of arbitrary SLA terms and term languages and thereby ensures the applicability of the templates to arbitrary domains. The templates can be seen as skeletons that allow terms that are specific to a particular usage to be plugged in. This easy way of adaptation has so far only been realised in the SLA representation. This paper shows how the plug-in approach can be extended to other parts of SLA management, namely the SLA creation phase, and how this allows for the simple migration of Grid applications to the Cloud.

The focus of this paper is on the negotiation of SLAs. SLAs can be established in various ways and it has been shown that the best way to negotiate an SLA depends on the context of service use [2]. Potential protocols are bi-partite negotiations and auctions. Although these protocols result in the same output (a contract for service access), their workflows are completely different and have so far only been implemented in separate systems. A first attempt to standardize negotiation protocols was done within the WS-Agreement specification, addressing the Discrete-Offer Protocol, which covers simple negotiation in two phases. Within this protocol, the customer can request an offer, which is then created and sent back by the service provider on which the customer can react with an agreement or a disagreement. Currently the negotiation part is addressed in a separate specification document called WS-Agreement Negotiation [13] which also covers multi-phase negotiation. This paper proposes an approach to integrate and unify the different methods of negotiation.

6. Developments

Looking at the current state-of-the-art in SLA management, it is quite obvious that several solutions exist, but they are limited in scope, in their capabilities, and especially in terms of flexibility.

Experience has shown that as soon as a new or adapted behaviour of an SLA management framework is required, large amounts of efforts are spent on rewriting major parts of the code or even re-implementing the system from scratch. The inflexibility of the currently existing systems is seen as one of the main obstacles which prevent general adoption of SLA management systems in electronic services markets, and in particular in the Cloud domain.

6.1 The Plug-In Approach

So far, all approaches towards the delivery of SLA management frameworks for electronic commerce ended up with static solutions. This leads to the unwanted situation that in case *Capability A* is needed, *Framework X* is the best to use, but when *Capability B* is needed, it is better to use *Framework Y*. In the case where a customer or a service provider wants to make business using A and B, he has to install and maintain X and Y, increasing the needed effort and costs for the infrastructure. On the other hand, a customer/service provider only providing B is loosing competitiveness in the market by not being able to support customers using A.

We propose a new approach which may in the long-term lead to a change of the methods and paradigms of the development of Grid and Cloud frameworks – adaptation of the behaviour of components on-the-fly during runtime of services. The main idea behind this approach is the separation of particular implementation logic and algorithms from the more holistic flow of abstract processing steps. If a generically usable "workflow" can be found for a problem and the interfaces to the particular function implementations can be defined, such particular functions can easily be replaced by implementations that fit particular needs and circumstances. We call this concept the POL-approach (POL = Plug-in Of Logic), which foresees exactly this aforementioned split in

- Service Base and
- Logic Plug-ins.

Figure 2 shows this split for the example of plugging in SLA negotiation logic. From the set of available negotiation logic implementation, the system chooses the one which best fits the needs of the customer or the service provider (e.g. a one-phase commit or an auction protocol).

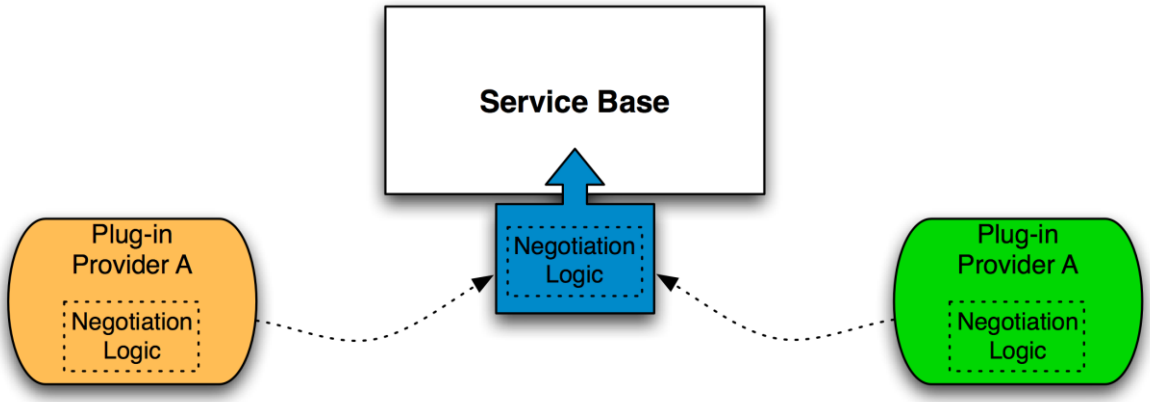

Figure 2 – Service Base and the Logic Plug-in

Figure 2 also introduces the role of the *Plug-in Provider*. This provider opens up a new business model for selling/renting/delivering the logical extensions for the system to be able to operate in the most efficient way. The plug-in provider offers pieces of functionality (the plug-ins) that can be used by service providers and users to augment their software with certain capabilities. In the example these capabilities are particular negotiation protocols. Plug-ins and their providers are found via service discovery and are plugged in a "functionality socket" which defines the needed process on an abstract level (see below). Plug-ins must match that socket and can then be used at the appropriate time by the process realized by the service base.

6.2 Describing the Capabilities – An initial sketch

The plug-in approach requires paying attention to the interface between service base and plug-in. Two aspects are of main importance: the definition of the interface provided by the service base and to be satisfied by plug-ins, and the description of the plug-ins themselves. Both aspects need to be described in an agreed and standardised way in order to ensure interoperability. Compared to protocol definitions, integration is thereby moved to a higher level of abstraction.

The interface description is provided by the designers of the service base. Ideally, these interfaces are defined by a standards body in an open way in order to encourage integration by developing plug-ins for different execution environments and contexts. The interface description consists of syntactical as well as semantic descriptions of the available

functions, their parameters, and their behaviour, as well as the allowed sequences of invocation. So far, we have only addressed the syntactical part and left the other parts open to further investigation and later selection of the formalisms to use. An example of our current description template is shown in Figure 3. It contains the formal part of a WSDL syntactical interface description and the informal description of the other three aspects as human readable text. We refer to this template as the "interface and logic description language – IALDL".

```
<?xml version="1.0" encoding="UTF-8"?>
<ialdl:Socket
   xmlns:ialdl="http://schemas.ialdl.org/ialdl/2010/05/ialdl"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

   <identifier>http://www.example.org/haemo</identifier>
   <wsdl>
      <wsdl:definitions>...</wsdl:definitions>
   </wsdl>
   <logic name="Haemodynamic simulation">
      <description>
         A haemodynamic CFD simulation.
         Function simulate starts the simulation and takes as
         input a blood vessel geometry and simulation parameters,
         including the blood pressure function over time,
         simulation time, and simulation granularity.
         ...
         The allowed sequence of function invocations is:
         - sim = simulate()
         - status = check_status(sim); until status equals finished
         - get_results(sim)
      </description>
   </logic>
</ialdl:Socket>
```

Figure 3 – Structure of an Interface and Logic Description Language Document

This language is a conceptual representation of the needs and is intended to act as basis for further developments. Its main parts are as follows:

- The *identifier* part: A unique identifier, usually a URL, which allows finding additional information about the document, e.g. the author of this file.
- The *wsdl* part: This part specifies the interfaces and the input/output parameters according to the Web Services Description Language (WSDL [10]).
- The *logic* part: This part is intended to contain the logic behind the interfaces, which is needed to provide all necessary functionalities of a component. As a simplification, this logic tag has a name (which supports the definitions of different logics within one description) as well as either a link to the logic (maybe provided within a different file) or the logic itself. This is currently expressed as human readable text.

The second part to be formalised are the descriptions of the plug-ins. The information needed here is

- the identification of the (WSDL) interface the plug-in realises,
- a description of the capabilities this plug-in provides (e.g. the use of an auction protocol, or the possibility to negotiate privacy parameters).

We have so far not realised a framework for expressing such descriptions. In all our experiments we have used manual selection of plug-ins. For the automated selection of plug-ins, the aforementioned information is needed in a way that is processable and

understandable by machines. In addition, the specification of requirements and preferences of the customer is a must.

7. Results

Our approach has been applied to modelling the two previously described use cases. Both use cases differ in the terms of the SLAs as well as the negotiation logic that is best suited for establishing SLAs. We used our framework to replace the negotiation logic based on the requirements and preferences of the service customer.

For the eHealth use case, the main requirement on negotiation was to get access to a simulation service quickly and to ensure that results are available well in time for the surgical intervention. As there is no time for a potentially time-consuming auction, the hospital opted to use a bi-partite negotiation scheme and to start negotiating with a set of known, pre-selected providers.

In the crash simulation use case, the time needed for negotiating a contract with a provider is not of particular impact. The main requirement here is the guarantee of a stable environment and time limitations are only given with respect to the start of the simulation. Therefore an auction scheme, potentially providing an economically better result, was chosen.

For both use cases an optimal negotiation process was selected and used. Although in one instance an auction protocol and in the other a bi-partite negotiation was used, both use cases were realized using the same middleware.

Both use cases have been extended with capabilities to use Cloud service providers (PaaS and SaaS). In the SaaS case (i.e. the eHealth use case), the service description and the associated guarantee terms had to be changed in order to accommodate the much higher level of abstraction of the SaaS service when compared to the more infrastructure-related terms in Grid SLAs. The PaaS terms remained basically the same, although their representation was different (specifying the need for some installed software (Java) and its stability in the case of a Grid SLA versus specifying a Java service in the Cloud case).

Adapting the use cases to the Cloud required the implementation of negotiation plug-ins that are able to deal with the protocols of Cloud services. As current Cloud offerings do not support SLA negotiation, we realised our own provider negotiation components. These components realised slightly different protocols in order to show that our system can adapt to different behaviour easily.

8. Business Benefits

Both service provider and customer benefit from our approach. In addition, the introduction of the Plug-in Provider role gives rise to an additional plug-in market that third party software providers can tap. We foresee that such a market will be initially most attractive to providers of domain-specific business-logic who develop value-added solutions and whose customers move their applications to run in XaaS environments. Those customers need the domain-specific services to now run in Clouds and the software providers can port existing solutions with predictable effort following our plug-in approach.

The main benefit for service providers is their ability to easily enhance their capabilities and market penetration by simply adding plug-ins to their offerings. No large up-front investments are needed which would be required when installing an additional framework.

Similar benefits can be reaped by service consumers as they can increase the number of candidate service providers by adding plug-ins to their service acquisition system. Again, no large up-front investments are required.

The potentially large space of plug-ins has the potential to create a plug-in market that specialised Plug-in Providers can tap. The providers can focus on developing, maintaining,

and offering plug-ins for various functional aspects and protocols. Providers will compete on price, functionality, reliability, and other aspects, creating a market in its own rights with the associated benefits of efficiency for service providers and customers.

9. Conclusions

The framework presented in this paper demonstrates how Grid-like approaches can be enhanced to provide more flexibility and reliability and by that increasing the attractiveness of the Cloud and the competitiveness of Cloud service providers. This is achieved by integrating approaches from other research fields, such as the Open Model methodology, with current implementation technologies. Furthermore, the attractiveness of new business models is increased for the providers and the users of Middleware as a Service deployed within a Cloud infrastructure.

All this is achieved with keeping the generality of the solution in mind: the approach has to be applicable in domains beyond Service Level Management, for example for business process execution. Whether this goal can be achieved needs further research, but we are confident that the solution is amenable to easy adaptation to other SLA lifecycle phases. Another benefit of the approach is the re-use of existing European results realised by the previous wave of distributed systems research, leading to harmonized Service Level Management solutions for Grids and Clouds.

References

[1] Koch, S., Strecker, S., and Frank, U. Conceptual Modelling as a New Entry in the Bazaar: The Open Model Approach, In: Open Source Systems, Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M., and Succi, G. (eds.), Springer, Heidelberg, Volume 203, IFIP International Federation for Information Processing, 2006, pp.

[2] Koller, B., Towards Optimal Creation of Service Level Agreements, In: Proceedings of the eChallenges 2009 Conference (eChallenges 2009), Istanbul, Turkey, October 2009, IIMC International Information Management Corporation, 2009, ISBN: 978-1-905824-13-7.

[3] Dey, A. K., Understanding and using context, Personal and Ubiquitous Computing, Vol. 5, pp. 4–7, 2001.

[4] Mazzeo, M. D. and Coveney, P. V., HemeLB: A high performance parallel Lattice Boltzmann code for large scale fluid flow in complex geometries, Computer Physics Communications, Vol. 178, No. 12, pp. 894–914, 2008.

[5] The TeleManagement Forum, SLA Management Handbook, Volume 2, Concepts and Principles, Release 2.5. The TeleManagement Forum, Morristown, New Jersey, United States, 2005.

[6] ETSI Technical Committee CLOUD, Initial analysis of standardization requirements for Cloud services, ETSI Technical Report, TR 102 997 V1.1.1, 2010.

[7] Jeffery, K., et al., The Future of Cloud Computing – Opportunities for European Cloud Computing beyond 2010, 2010.

[8] Keller, A., and Ludwig, H., The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, Journal of Network and Systems Management, vol. V11, pp. 57–81, March 2003.

[9] Andrieux, A., et al., Web Services Agreement Specification (WS-Agreement), Technical Report, Open Grid Forum, Grid Forum Document GFD.167, 2007.

[10] R. Chinnici et al.,"Web Services Description Language (WSDL) Version 2.0", A W3C Technical Report (2007)

[11] Laria, et al., D4.1.3 – Final BREIN Architecture, Public Deliverable of the BREIN project, 2009.

[12] Wilson, M., Arenas, A., and Schubert, L., D63 – TrustCoM Framework V4, Technical Report, The TrustCoM Project, 2007

[13] A. Andrieux, K. Czajkowski, *et al.*, "Web Services Agreement Negotiation Specification," tech. rep., Open Grid Forum - OGF, 2007.