A Framework for Multi-level SLA Management*

Marco Comuzzi[1], Constantinos Kotsokalis[2], Christoph Rathfelder[3], Wolfgang Theilmann[4], Ulrich Winkler[4], and Gabriele Zacco[5]

[1] Eindhoven University of Technology, The Netherlands
[2] Dortmund University of Technology, Germany
[3] FZI Research Center for Information Technology, Karlsruhe, Germany
[4] SAP Research, Karlsruhe, Germany
[5] Fondazione Bruno Kessler, Povo (Trento), Italy
m.comuzzi@tue.nl, wolfgang.theilmann@sap.com

Abstract. Service-Oriented Architectures (SOA) represent an architectural shift for building business applications based on loosely-coupled services. In a multi-layered SOA environment the exact conditions under which services are to be delivered can be formally specified by Service Level Agreements (SLAs). However, typical SLAs are just specified at the customer-level and do not allow service providers to manage their IT stack accordingly as they have no insight on how customer-level SLAs translate to metrics or parameters at the various layers of the IT stack. In this paper we present a technical architecture for a multi-level SLA management framework. We discuss the fundamental components and interfaces in this architecture and explain the developed integrated framework. Furthermore, we show results from a qualitative evaluation of the framework in the context of an open reference case.

Keywords: Service Level Agreement (SLA), Service-Oriented Infrastructure (SOI), e-Contracting, Adaptive Infrastructures, Manageability, Non-Functional Properties.

1 Introduction

The paradigm of *Service-Oriented Architectures* (SOA) has changed the way for building IT-based systems [2]. Initially SOA was mainly applied to restructure the IT stack within an organisation. More recently it has also evolved as a common paradigm for cross-organisational service landscapes where services are considered as tradeable goods. Consequently, services operate under a strong business context where service customers can expect services to be provided under well-defined and dependable conditions and with clearly associated costs.

Service Level Agreements (SLAs) are a common way to formally specify the exact conditions (both functional and non-functional behaviour) under which services are or shall be delivered. However, the current SLAs in practice are just specified at the customer-level interface between a service provider and a

* Presented by the authors on behalf of the SLA@SOI consortium [1].

A. Dan, F. Gittler, and F. Toumani (Eds.): ICSOC/ServiceWave 2009, LNCS 6275, pp. 187–196, 2010.

service customer. Customer-level SLAs can be used by customers and providers to monitor whether the actual service delivery complies with the agreed SLA terms. In case of SLA violations, penalties or compensations can be directly derived. Customer-level SLAs do not allow service providers to either plan their IT landscapes according to possible, planned or agreed SLAs; nor do they allow understanding why a certain SLA violation might have occurred. The reason for this is that SLA guarantee terms might not be explicitly or directly related to actual performance metrics or configuration parameters. This makes it difficult for service providers to derive proper configuration parameters from customer-level SLAs and to assess (lower-level) monitoring metrics against customer-level SLAs. Overall, the missing relation between customer-level SLAs and (lower-level) metrics and parameters is a major hurdle for managing IT stacks in terms of IT planning, prediction or adjustment processes and in accordance with possible, planned or actual SLAs.

As part of the European Research project SLA@SOI [1], we developed the vision to use the paradigm of SLAs for managing a complete IT stack in correlation with customer-level SLAs which are agreed at the business level. This complies with the current technical trend to apply the paradigm of service-orientation across the complete IT stack, i.e. infrastructure/platform/software as a service, but also with the organisational trend in IT companies to organise different departments as service departments, providing infrastructure resources, middleware, applications or composition tools as a service. SLAs will be associated with multiple elements of the stack at multiple layers, e.g. SLAs for elements of the physical/virtual infrastructure, middleware, application and process-level. Such internal SLAs describe the contract between the lower-level entites and a higher-level entities consuming the lower ones. More precisely, the SLAs specify the required or agreed performance metrics but also the related configuration parameters.

This paper presents the detailed conception and implementation of a multi-level SLA management framework and it is built on a previous discussion of a purely conceptual architecture [3]. The remainder of this paper is organised as follows. Section 2 introduces the developed framework while Section 3 provides evaluation results in the context of a case study. Section 4 concludes with a brief summary and outlook. A full version of this paper including a discussion of related work can be found at [4].

2 SLA Management Framework

The design and implementation of our SLA management framework is based on the conceptualisation presented in [3]. There, we introduced the following core concepts:

- The four roles of service customer, software provider, service provider and infrastructure provider;
- The three layers of business, software, and infrastructure management;

- A service lifecycle model including design, negotiation, provisioning, operation, i.e. monitoring and adjustment, and decommissioning;
- Conception of relevant basic data store entities covering design-time and run-time data for the various layers and roles; and
- Conception of functional flows for the lifecycle phases of negotiation, provisioning, and operation.

In the following we now show a concrete technical architecture which implements the previous conceptualisation.

2.1 Technical Architecture

The technical architecture is presented in two steps. First we show how the architecture is split into different modules, each of them having a clear responsibility and interface. Second, we show how the envisioned SLA management procedures are realised in terms of scenario flows. The main challenges for such an architecture are (1) a clear separation of concerns between concepts that are highly interrelated and (2) a reasonable abstraction and flexibility that can easily support different target scenarios.

The technical architecture of the SLA management framework consists of six modules. Out of them, four modules are primarily responsible to orchestrate the core activities of the SLA negotiation, SLA provisioning, and SLA operation phase:

- The *negotiation module* is responsible for conducting multi-level SLA negotiation. It contains the design time repository (which allows for model-based performance predictions), the SLA template registry (that stores all the possible/offered templates), and it is also aware of the rules that support SLA translation between layers.
- The *provisioning module* co-ordinates the provisioning of SLA-specified services. It contains the SLA registry (storing the established SLAs), the software landscape (storing all software related artefacts for provisioning), and a scheduling component that makes sure temporal and other kinds of dependencies between services are honoured.
- The *monitoring module* is responsible for the decomposition of a SLA into monitorable rules and the analysis of a flow of incoming monitoring events against these rules. Eventual violations can be reported again on the level of SLAs.
- The *adjustment module* is responsible to collect all detected SLA violations or warnings and to trigger adjustment actions. It makes use of a set of adjustment patterns and a manageability model to achieve this.

The architecture is complemented by two modules which address the pure business / infrastructure view.

- The *eContracting module* is responsible for managing the business relationship with service customers. It contains the notion of *business offers* (products) and *policies* (how individual customers are processed in terms of pricing etc).

– The *infrastructure module* is responsible for managing all the infrastructure resources in a cloud-like manner. It contains the landscape of existing physical resources and possible virtualised counterparts.

Last, a technically motivated module realises an *event bus* to support asynchronous communication between the other modules.

Figure 1 provides an overview of the technical architecture, its modules, interfaces, and main relationships.

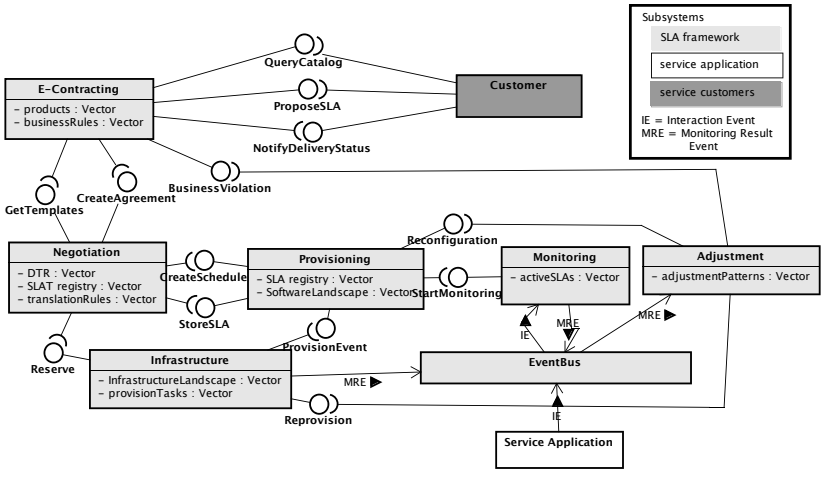

Fig. 1. Module Architecture

The dynamics of the SLA management framework are described along two scenarios. The first scenario concerns the negotiation and provisioning of the SLA, whereas the second scenario refers to runtime support, and, in particular, the monitoring, which detects SLA violations, and the adjustment, which reshapes services deployment configuration in relation to detected SLA violations.

The first scenario (see Figure 2) starts with the customer proposing an SLA offer to the SLA Framework, by invoking the eContracting module interface. The scenario implies an initial phase where the customer queries the eContracting module for retrieving the available product templates, which is not represented in Figure 2. The SLA offer proposed by the customer is accepted if it can be provisioned by the SLA framework according to the current status of software and infrastructure resources. The SLA offer is, therefore, forwarded to the Negotiation module, which creates the SLA hierarchy and issues requests for provisioning the software services and reserving the virtual machines required for execution of the services. If both requests are successful, the SLA is stored and then sent back to the customer. Before being sent back to the customer, the SLA is completed with pricing information. Note that we are using WS-Agreement for expressing SLAs and the usage of WS-Agreement has a counter intuitive side-effect: Since

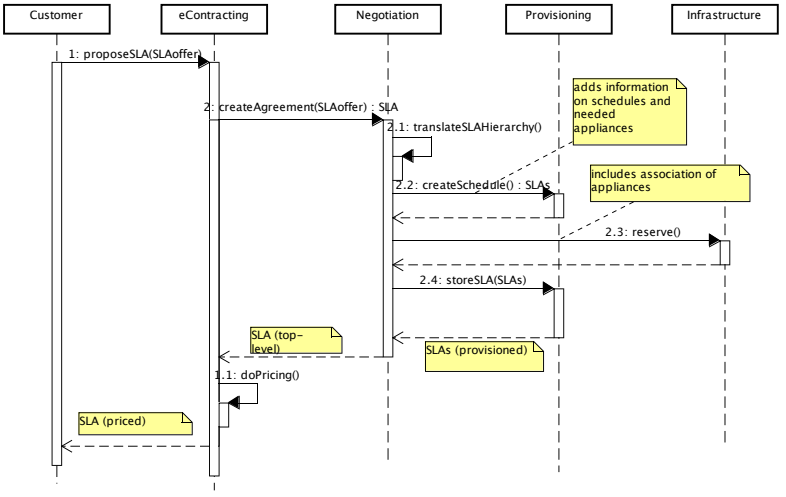

Fig. 2. The negotiation and provisioning scenario

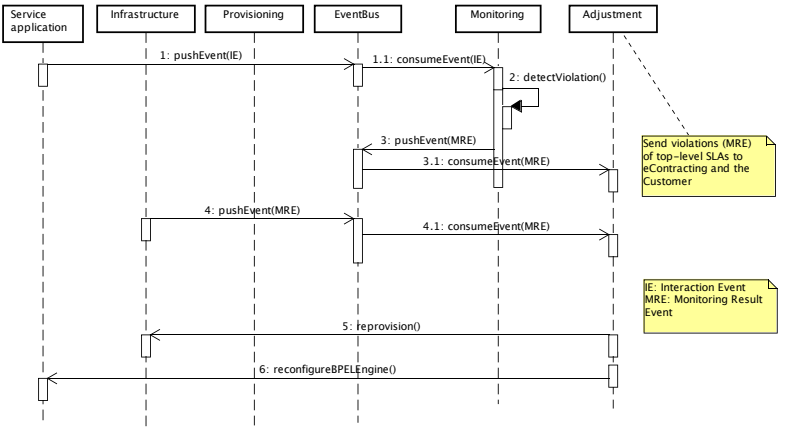

Fig. 3. The monitoring and adjustment scenario

there are no counter-offers foreseen in the specification, it is not possible for the initiator (here: the customer) to reject the proposed price. The upcoming WS-Agreement-Negotiation specification will resolve this issue.

The second scenario (see Figure 3) starts when an SLA becomes active and it is therefore submitted to the Monitoring module. Monitoring at the software layer is performed by checking the rules derived from the SLA against the *Interaction Events* (IE) produced by the services' execution environment, e.g. timestamped service operation calls and responses. When an SLA violation is detected, e.g. the average completion time of an operation exceeds the threshold reported in the

SLA, a *Monitoring Result Event* (MRE) is pushed on the Event Bus. Note that violations of terms at the infrastructure layer, e.g. abnormal CPU or memory usage, are directly detected by the Infrastructure module and pushed as MREs on the Event Bus. Violations (MREs) are read by the eContracting module, in order to be shown to the Customer, and by the Adjustment module. Adjustment, in particular, may decide to apply some corrective actions to overcome SLA violations (e.g. by reconfiguring software components or resizing hardware capacity).

2.2 Integration Foundations

A solid foundation for building a common approach for all modules is represented by the adoption of WS-Agreement as SLA modelling solution. WS-Agreement defines a signalling protocol for establishing SLAs. As such, it is domain-agnostic and provides generic data types to describe agreement templates and offers. WS-Agreement defines *Agreement Templates* to be roughly containers of terms, and constraints on those terms. A template consists of 4 top-level elements: *name*, *context*, *terms* and *creation constraints*, where terms are subdivided into *service description* terms and *guarantee* terms. Agreement documents themselves have the same structure, without containing any constraints section though.

For the terms, which form the content of the agreement depending on the domain at hand, the project developed a comprehensive core meta-model to be used as a basis by all modules. This core meta-model defines a number of essential constructs (e.g. temporal units, quantitative resource descriptions, etc) and binary operators for them. Using those constructs, it is possible to define the agreements' higher level terms, that can be understood in the same way by all components of our system.

The second foundation for a common approach of the modules is the adoption of a framework that eases the assembly of the architecture modules into the final platform. From a technological point of view, this is achieved using Spring [5], an open source application framework that provides core features services as well as advanced solutions for building complex applications.

3 Prototype Implementation and Case Study

The implemented framework has been evaluated against a reference application, the so-called Open Reference Case (ORC). This section briefly sketches the structure of this application, explains the related SLA hierarchy that we established, and provides results from a qualitative evaluation.

3.1 Open Reference Case

The ORC is a service-oriented software system supporting a retail chain scenario. The ORC extends the Common Component Modelling Example (CoCoME) [6], which represents a component-based trading system dealing with the various

aspects of handling sales at a supermarket. This includes the interaction at the cash desk with the customer, including product scanning and payment, as well as accounting the sale at the inventory.

The ORC is about a *Software as a Service* (SaaS) scenario where a service providers offers a solution to a number of different customers. Thus the service provider negotiates an SLA with each customer. The service provider in turn relies on a Software Provider (delivering the ORC application), a cloud-like Infrastructure Provider and External Service Providers which offer complenting functionality.

The ORC itself consists of a bundle of atomic and composite software services (for inventory, ordering and payment) which in turn may depend on other external software services (such as credit card validation). Technically, the ORC is offered with 2 deployment options (addressing small and large customers): *all in one*, with the service containers, BPEL engine, and database running on the same virtual machine; *separated database*, with the database running on a different virtual machine; and *separated database and BPEL engine*.

3.2 SLA Hierarchy

Generally, a hierarchy of SLAs is a set of SLAs that are associated in a way that captures some kind of dependency of one on another. This kind of association / dependency is not always straightforward. It may be the case that the reduced capacity of a provider forces it to rent capacity from another provider, to serve its clients as per the standing agreements. In a different scenario, the dependency might be due to a request for fail-over redundancy, in which case the failure of an agreement of the provider may not affect at all the agreements of its customer. Dependencies may just as well be related to functionality that a provider cannot offer by default, and that therefore must be outsourced.

SLA dependencies may also be internal to a single provider. This is the case where a department of a provider relies on another department of the same provider, in order to accomplish its tasks. Very often this has to do with a business department relying on the IT department, for back-office or other functions.

Our evaluation follows the scenario of a single provider with 3 internal departments (business department, software IT department and infrastructure department) relying on each other via internal SLAs.

Figure 4 illustrates the realized SLA hierarchy. It shows the 3 departmental layers where the top-level business SLA describes the complete offered Retail-as-a-Service solution, including software but also relevant business aspects (legal conditions, support agreements, etc.). The software service bundle relies on a hierarchy of lower-level software services. The hierarchy here allows the service provider to precisely understand and monitor his software landscape in relation to top-level business SLAs. Last, the collection of software services relies on the infrastructure where the SLA specifies the nature and conditions of the infrastructure resources (here virtual machines) needed to host the software.

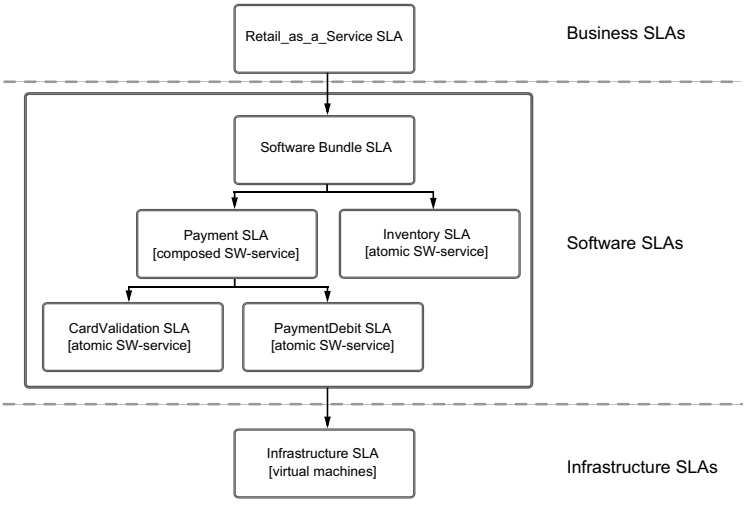

Fig. 4. Implemented SLA hierarchy

3.3 Evaluation

Our experiments with the SLA Framework cover the two main scenarios already described in Section 2.1.

For the negotiation and provisioning scenario, we experimented with different SLA templates, deployment options and infrastructure options. We decided to differentiate services starting from the highest service layer, the business services layer, and we adopted the widely used metaphor of *gold*, *silver* and *bronze* class services. Hence, we run our experiments with three distinguished business level SLA templates. Although the three SLA templates provide default values, the customer in our scenarios should be able to request individual non-functional properties, i.e. completion times and arrival rates (as *qualifying condition*) for each service operation accessible by the customer. Note, that the *qualifying condition* is understood as "customer obligation" within an SLA, meaning that if the arrival rate of service requests is higher than agreed, the service provider is no more bound to sustain the agreed completion time. In this case it is simply the customer who violated the SLA. At the software level we used individual templates for each service in order to support flexible service composition and provisioning, and to gain fine grained SLA hierarchies as depicted in Figure 4. The infrastructure template allows resource configuration (comprising multiple virtual machines) and enables the service provider to request guarantees from the infrastructure provider on various virtual resources, such as CPU or memory.

The core of the actual negotiation procedure is now the translation of terms along the SLA hierarchy, the evaluation of different composition/deployment/ infrastructure options and the selection of the best suited ones. The basic element to this procedure is a prediction service which, based on a PCM model [7] of the

ORC, can simulate the non-functional behaviour of all these options. Taking the results of this prediction, the SLA hierarchies of possible solutions are associated with terms on their non-functional behaviour and costs. Last, the negotiation component selects the cheapest option that still satisfies the requested top-level SLA. Experiments show the feasability of this approach and the selection of well-suited system setups which realize the requested customer SLAs in a cost efficient manner.

For what concerns the monitoring and adjustment scenario, a software layer MRE, e.g. a too high average response time of an operation call, is identified by the name of the guarantee term and the unique id of the SLA to which the guarantee term belongs. The violation, for instance, may refer to the guarantee term on the average completion time of the operation `bookSale` of the service `InventoryService` in the ORC. This information is sufficient for the Adjustment module to retrieve the violated SLA and the related SLA hierarchy and to decide eventual control actions. Infrastructure violations report the id of the resource on which the violation has occurred and information on the parameter, e.g. CPU or memory utilisation, which has been violated.

Adjustment actions have been realised at various layers relying on the analysis done within the adjustment module. SLA violations may result from faulty customer behaviour, e.g. where a customer exceeds the SLA-agreed maximum workload. In this case a message is sent via the eContracting module to the customer, to inform him/her about this.

Concerning the infrastructure, Adjustment may trigger the re-provisioning of the virtual machines on which services are executed. In particular, if the Adjustment receives a violation of the software-level guarantee term on a service operation completion time and, at the same time, a violation of the CPU utilisation on the virtual machine on which the service is executed, then it commands the Infrastructure to re-provision the virtual machine on which the service is executing with a higher CPU share.

Concerning the software layer, the corrective action implemented at the current stage is the reconfiguration of the BPEL engine in which composite services are executed. The reconfiguration is triggered when a software-level guarantee term on completion time of an operation of a composite service is violated, and when it is not possible to find a corresponding infrastructure violation of the virtual machine where the service is executed. The Adjustment detects, in this case, that the problem belongs to the BPEL engine in which the service is executing; the reconfiguration of the engine involves the increase of thread pool of the composite service for which the violation has been detected.

Last and if no other adjustment action can be detected, violations (MRE) are reported as fault of the service provider and the customer is informed accordingly, including the acceptance of possibly agreed penalties.

4 Conclusions

This paper presents a technical architecture and implementation for a multi-level SLA management framework. We discussed the fundamental components

and interfaces in the architecture. The framework dynamics were described via two fundamental scenarios, which cover the core SLA management lifecycle including negotiation, provisioning, monitoring and adjustment. Furthermore, the main technical and integration aspects of the developed framework have been described. The framework has been successfully applied within the context of a reference application. The qualitative evaluation includes a description of the actually realised SLA hierarchy and the details about the concrete scenario steps. A full version of this paper including a discussion of related work can be found at [4].

Although the experiments have been successful as a feasibility study, they also revealed some areas that require further improvement. The most important among these are a clearer separation of concerns between the horizontal aspect of system layers and the vertical aspect of SLA management. Second, the framework must support more flexible integration techniques for different target scenarios. Besides improving the architecture along the aforementioned lines, we will also start a thorough evaluation based on five industrial use cases, which represent a broad set of relevant but also distinct scenarios.

Acknowledgements

The research leading to these results is partially supported by the European Community's Seventh Framework Programme (FP7/2001-2013) under grant agreement no.216556.

References

1. SLA@SOI project: IST- 216556; Empowering the Service Economy with SLA-aware Infrastructures, http://www.sla-at-soi.eu/
2. Papazoglou, M., van den Heuvel, W.J.: Service oriented architectures: approaches, technologies and research issues. The VLDB Journal 16(3), 389–415 (2007)
3. Theilmann, W., Yahyapour, R., Butler, J.: Multi-level sla management for service-oriented infrastructures. Towards a Service-Based Internet, 324–335 (2008)
4. Comuzzi, M., Kotsokalis, C., Rathfelder, C., Theilmann, W., Winkler, U., Zacco, G.: A framework for multi-level sla management. Technical Report 2010-1, SLA@SOI project (April 2010)
5. Spring framework: Eliminating Enterprise Java Complexity, http://www.springsource.com/
6. Rausch, A., Reussner, R., Mirandola, R. (eds.): The Common Component Modeling Example. LNCS, vol. 5153. Springer, Heidelberg (2008)
7. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1), 3–22 (2009)