Fraunhofer
IESE

Engineering Service Level Agreements as an Integral Part of Software Systems and their Architectures

Authors:
Taslim Arif
Frank Herold
Thomas Kiesgen
Matthias Naab

IESE-Report No. 057.15/E
Version 1.0
December 2015

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr.-Ing. Peter Liggesmeyer
(Executive Director)
Prof. Dr. Dieter Rombach
(Director Business Development)
Fraunhofer-Platz 1
67663 Kaiserslautern
Germany

Engineering Service Level Agreements as an Integral Part of Software Systems and their Architectures

Taslim Arif[1], Frank Herold[1], Thomas Kiesgen[2], Matthias Naab[1]

[1]Fraunhofer IESE
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{taslim.arif, frank.herold, matthias.naab}@iese.fraunhofer.de
[2]vwd GmbH
Stiftsplatz 6/7, 67655 Kaiserslautern, Germany
tkiesgen@vwd.com

Abstract. Service level agreements (SLA) for most software systems which are offered as services, are usually written in text form only. Thus, they are often fuzzy and not integrated with the system, i.e., the development activities and architecture are independent from the SLAs. Therefore, there is typically no transparency for service consumers regarding the actual service status, while service providers have no control over the system with respect to the SLAs or are dependent on experts. In this paper, we present a process for systematically integrating SLAs with an interoperable software system and its architecture. We propose a maturity model for SLA enforcement so that organizations can clearly see where they are with respect to SLA integration. Formalizing the SLAs with a Domain-Specific Language and establishing architecture-centric monitoring and analysis can contribute a lot to the enforcement of SLAs. We share our experiences and key takeaways for SMEs learned from an industrial case study. Our approach and the takeaways will help service providers to practically guarantee high quality of SLAs and will make the service providers more trustworthy.

Keywords: Service Level Agreement, Service Oriented Architecture, Monitoring, Maturity Model, Runtime Architecture

1 Introduction

Software systems are offered as services (Software-as-a-Service – SaaS) or interoperated with other software systems quite often. In most cases, the provider and the consumer of those software systems are not from the same organization. They mostly have a common understanding of the functionality, but unfortunately quality requirements are often fuzzy. Usually, legal contracts called Service Level Agreements (SLA) are available on paper, but their wording is rather ambiguous. As a result, consumers often lack confidence regarding the quality level they are getting. And as the SLAs are not written in concrete terms, consumers do not have a clue when such an SLA is being violated. Taking legal actions against the provider is also not possible without evidence. Moreover, in case of interoperable systems, the quality of composed systems cannot be

guaranteed. On the other hand, from the provider's perspective, what is missing is the opportunity to take clear actions to achieve those SLAs and to get competitive advantages. They are heavily dependent on intuition and experts for making their systems SLA-aware.

Large organizations are more likely to manage and deliver services and integrate SLA enforcement following their own proprietary methods and tools. They can even afford various experts. However, for SMEs it is extremely hard to enhance an existing system or design a new service-oriented system with regard to SLA integration and enforcement, as there is a lack of methods and tools supporting SMEs in a lightweight and efficient manner. Available tools are either too complex or costly and are not tailored to the needs of SMEs. So, the question we address here is: How can software architects enhance Service-Oriented Systems Engineering for SMEs to an appropriate level of service level enforcement?

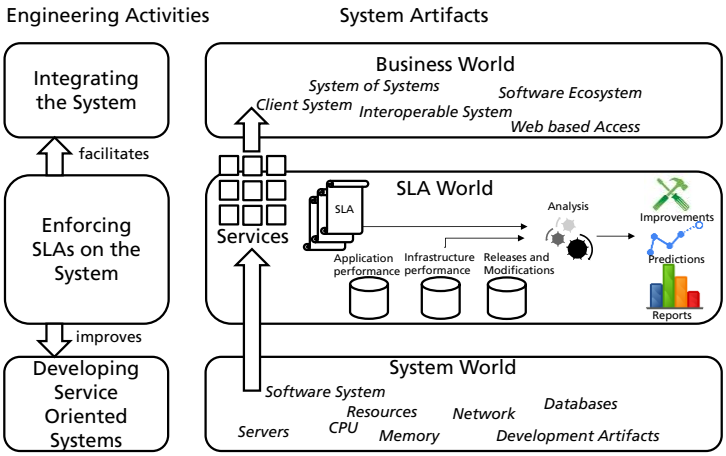

Fig. 1. Positioning "SLA Enforcement" in the Context

Fig. 1 depicts the context software architects need to be aware of in the sense of SLA enforcement from the perspectives of artifacts and engineering. We identified three conceptual layers. At the basis is the system world, which is composed of the system itself and all the resources it uses during runtime – e.g., machines, storage devices, networks, etc. On top of this, we have the SLA world, which is composed of the SLAs, monitored data, SLA status, etc. with respect to the services offered by the system world. The highest level represents the business world – here we consume the services and create new systems, e.g., ecosystems, interoperable systems – or just access services via a web client. For each of these layers, software architects need to be aware of corresponding engineering activities. For the system layer, they need to identify the various development activities, which are most often independent of SLAs. For the SLA layer, these are mainly the SLA enforcement activities aimed at integrating SLAs into the system. Software architects should try to improve development activities by providing improvement suggestions with respect to SLA status. As these activities are

mostly missing in practice, they are the focus of this paper. Moreover, in the business layer, we have the integration engineering activities. If SLA engineering has not been done to a sufficient extent, service selection or composition cannot be done appropriately and quality cannot be predicted on the integration level.

1.1 Case Study and Research Approach

To find out the clear requirements of SLA enforcement, we collaborated with vwd GmbH (which runs a service-oriented system and offers it as a service) and elaborated the requirements in consultation with them. We then checked existing approaches and concepts (both from the literature and the tools perspective) in conjunction with the industrial settings to identify the gaps in existing approaches. Our concepts were prototypically integrated into their system to prove their feasibility. Moreover, we validated the results in their context.

1.2 Contributions

1. **Maturity model for SLA enforcement**: We built a maturity model that shows the SLA enforcement roadmap for an interoperable software system (3.1). Maturity models are available for service level management (SLM)[6]. Obviously, service level enforcement is part of the overall SLM maturity, but to date, no specific and detailed SLA enforcement maturity model has been available for software systems.
2. **Implementation of the maturity model**:
   (a) **Approach to formalizing SLAs**: SLAs need to be defined formally and unambiguously to benefit from them. We explain how a custom DSL can be created to formalize SLAs (3.2). There are quite a few DSLs [7, 9, 11] already available, but to avoid over- or under-formalization, organizations need just enough formalization. We show how that can be achieved.
   (b) **Approach to doing compliance checking**: We reveal how software architects can identify what needs to be monitored to do compliance checking. Monitoring tools usually provide several monitoring techniques and computing tools help to analyze the data, but we provide guidance on what needs to be monitored and how it needs to be analyzed (3.3).
   (c) **Approach to building runtime architecture**: To get the most out of formalized SLAs, it is necessary to understand the runtime architecture and the context of the target system and environment. We show how software architects can build a runtime architecture meta-model for the system (3.4).

2 Business Scenarios for Engineering SLAs for Systems

SLA integration brings along a number of business benefits that we considered as motivation to do research in this direction. The typical business scenarios are described below.

- **Exploiting Transparency**: From the consumers' point of view, it is important to be able to select among different services and have evidence of the provided service level qualities. If there is no choice, the consumers at least want to know what they are getting because nowadays, more and more business processes are dependent on software systems and their failures and unexpected behavior (in both functional and non-functional terms) can easily lead to serious issues at the business level. From the providers' point of view, it increases the credibility of an SME to expose the status of a software system to the outer world. It clearly shows confidence and maturity to fulfill the SLA contracts instead of just expert guessing.
- **Ensuring the quality of the system**: Integrating SLA formalization and enforcement into their software system will help SMEs to ensure system quality. Instead of formulating SLAs in a fuzzy and ambiguous way and trying to put SLA enforcement as something to be done on top of an already designed, implemented, and running system, an integration mechanism for SLA enforcement during development and operation time will have a huge impact on the quality of a system. As a result, SMEs will be able to run and monitor their systems in an SLA-aware manner. Furthermore, software architects who already consider SLA enforcement explicitly before system implementation will come up with a positive result in increased system quality.
- **Competitive advantage (providers' view)**: Being able to provide evidence and compliance with respect to specific SLAs can create a competitive advantage. Especially for SMEs, being able to report SLA compliance is crucial. It is becoming ever more important to make the non-functional aspects of a system explicit in order to compete with other service providers. Integrating SLA enforcement into the software system is therefore a key point and worth investing in.
- **Possibility of quality interoperation increases**: To expose not only the current status of a system from an external point of view, but to also provide insights into a software system to consuming services with respect to SLA enforcement contributes to an interoperability scenario. Especially in emerging interoperating scenarios such as Industry 4.0 or the Internet of Things, the importance of interoperating service is increasing significantly. A formalized and machine-readable SLA definition and status are thus an important component and a prerequisite for service composition of higher quality. Otherwise SLA enforcement in a service bundling setting is not predictable.

3 Approach for Engineering SLAs for Systems

To help software architects of SMEs integrate SLAs into their system and systems engineering, we have developed a maturity model for SLA enforcement. Before starting the integration work, the software architect needs to identify the enforcement situation and also needs to foresee the overall road map. The maturity model will help the software architects analyze their context and provide inputs in order to set goals with respect to SLA enforcement. Obviously, there is an existing service level management maturity model provided by ITIL [6] and service level enforcement is part of it. Within

the scope of this paper, we focus on the more specific aspect of service level enforcement and have detailed this aspect on several levels.

Once the context is analyzed and goals are set, the software architects need to implement some transitions of the maturity model for their organization. We show three important cases (Formalization, Compliance, and Building Meta-Model) of transitions from one level to another in the maturity model. The three cases are also examined in a case study. As organizational settings might vary drastically, the examples can be seen as first guidelines and should be tailored to the specific context of the organization interested in implementing the model.

3.1 Maturity Model

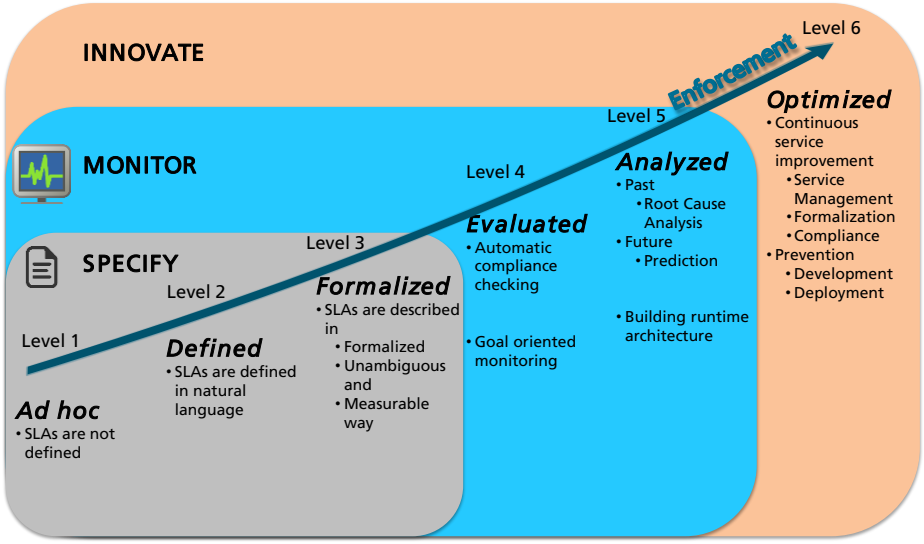

Fig. 2. Maturity Model for Service Level Enforcement

The maturity model (**Fig. 2**) describes several levels of SLA integration into software systems from the product and process points of view. The maturity model was developed by analyzing an organization that builds SLA-aware SOA systems and consolidating our experiences. It also takes into consideration the existing literature. As it might be possible theoretically to reach any specific level by bypassing a lower level, it is more likely that companies will upgrade step by step. Each level provides some benefits but also comes with some cost for the organization. Obviously, it makes sense to do this step by step to find the optimal trade-off between costs and benefits.

In the following, we describe six levels of the maturity model. Three criteria determine the level. The first criterion is related to specification – how well the SLAs have been formalized. The second criterion focuses on dedicated activities (for example monitoring etc.) for analyzing the situation with respect to SLA enforcement. The third criterion deals with improving the development and operational activities (supported

by data) with respect to SLA enforcement. The first three levels in the maturity model mostly focus on the specification and are therefore on the SPECIFY category level. The fourth and fifth levels are related to dedicated activities performed to analyze the SLA status. This is why they are on the MONITOR category levels in the model. The sixth level addresses improving the development and operational activities in a more SLA-aware manner. It is in the INNOVATE category of the model as it might require several changes in the organization and in the processes.

Level 1 – SPECIFY: Ad-hoc. SLAs are not defined even in case of necessity. No dedicated measures are undertaken to analyze the SLA enforcement situation. Measures for SLA integration from a product point of view and process point of view are ad-hoc, and in general, only a few measures are taken.

Level 2 – SPECIFY: Defined. SLAs are defined in natural language, but this leaves much room for interpretation. No dedicated measures are undertaken to analyze the SLA enforcement situation. Few measures are taken to integrate the SLAs during development and operation. Measures are taken based on manual expertise.

Level 3 – SPECIFY: Formalized. SLAs are defined formally in an unambiguous, machine-readable, and measurable way. No dedicated measures are undertaken to analyze the SLA enforcement situation. Several measures are taken to integrate the SLAs during development and operation. There is an established process in accordance with these measures. However, the measures are still taken based on experience and feedbacks from the customers.

Level 4 – MONITOR: Evaluated. SLAs are defined formally. Systems are monitored with respect to SLA fulfillment and automatic compliance checking is performed. Several measures are taken to integrate the SLAs during development and operation. The measures are not totally based on experience anymore but are also supported by compliance checking output.

Level 5 – MONITOR: Analyzed. SLAs are defined formally. Systems are monitored with respect to SLA fulfillment and SLA enforcement related analysis. In addition to automatic compliance checking, root cause analysis, prediction of future violations, etc. is done on this level. Several measures are taken to integrate the SLAs during development and operation. The measures are based on various analysis results. In this step expertise level required to select some measures is significantly less because of the availability of the analysis results.

Level 6 – INNOVATE: Optimized. SLAs are defined formally. Systems are monitored with respect to SLA fulfillment and SLA enforcement related analysis. In addition to automatic compliance checking, root cause analysis, prediction of future violation,

etc. is done on this level. All measures are taken to integrate the SLAs during development and operation. Most importantly, the development and operation activities are integrated optimally. The measures are based on various analysis results. Measures are also undertaken to continuously improve the SLA enforcement activities (formalization, analysis, etc.).

In the following sections, we will describe how transitions can be made from one level to another. **Fig. 3** shows the scope of this paper. We detail the steps for formalization, compliance checking, and building meta-models for various types of analyses. To date, we only briefly visited the other transitions and will continue to work on them as part of our future research on this topic.

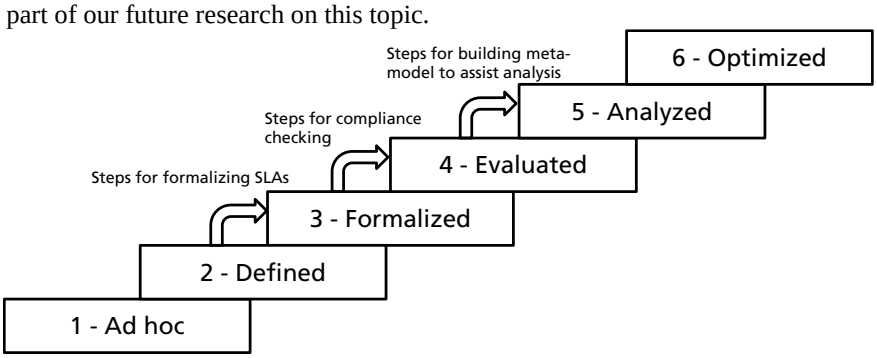

Fig. 3. Scope of this paper (3 transitions)

3.2 Level 3 - Formalization

SLAs need to be defined formally. This is mostly required to improve the concreteness and support the automation of SLA realization mechanisms. An SLA specification has to be able to express the quality requirements of the interoperation unambiguously. SLAs should define the quality metrics and the acceptance criteria. The formalization is not an end in itself; rather, formalization is a dynamic task that should be done in consideration of the future goals. If we include more SLA-relevant aspects, the formalization needs to be enhanced with respect to those goals. Below we describe the steps that would help architects to formalize SLAs in their specific organizational context.

- Identifying quality attributes: The first point is to identify the quality requirements from the customer's point of view. Becha et al. [2] presented a set of quality requirements for SOA systems. As the importance of one quality attribute varies from system to system, architects need to identify the quality attributes their organization would like to put into the SLAs, taking into consideration their customers' demands.
- Characterization of quality attributes: Selecting the quality requirements is not enough. Software architects need to characterize the quality attributes using architectural scenarios. This detailing is required to identify metrics that are measurable on the system boundary level as scenarios usually include triggers of some environmental context of the system and the corresponding responses and response measures.

- Identifying SLA context: Once the quality attributes are characterized, software architects need to identify the service level objectives (SLOs). Service level objectives indicate when some SLA is considered to be violated. This might be dependent on some context factors as well (for example time interval). Moreover, penalties are also part of the context.
- Select from existing DSL or create a new one: Once architects have defined their quality requirements, the metrics that represent those qualities, and the SLOs, they can check the existing DSLs available to represent these. If the existing DSLs are suitable, one of them can be selected. Otherwise, a new one should be created based on the needs. Sometimes existing DSLs can be very heavyweight to start with (even though it is possible to express everything). On the other hand, existing DSLs might have tool support that could be used for several SLA management activities.

In the above steps, we just described what has to be formalized with respect to the customer at the system boundary. For an analysis of the status of SLA, more insight views about the system are required. To achieve such insights, the metrics that are considered on the system boundary level need to be mapped to internal metrics or vice versa. Obviously this depends on the level of enforcement the organization wants to achieve and on the analysis or the tasks that need to be supported.

3.3 Level 4 - Evaluation: Compliance Checking

What is written in the SLA contract and how the system is really performing needs to be checked periodically according to the contract. Mostly this is done on a monthly basis, but it should be possible to do so for any time period. In some cases, a real-time compliance status is also valuable.

- Analyzing the SLAs: The first step in compliance checking is to analyze the SLAs. From there, we identify what has to be reported to the customer.
- How to monitor the selected metrics: Once we have identified what has to be monitored, we need to determine the monitoring strategy - how to monitor, how frequently to monitor, which tool to use, and so on. Monitoring should be appropriate to conclude about the SLAs. Therefore, a mapping between the monitored data and external metrics (metrics in the SLOs) needs to be established. Moreover, in some cases, human input might even be necessary.
- Evaluation: Once appropriate monitoring is in place, we need to evaluate or at least be able to evaluate the compliance of the SLAs. What has to be reported and to whom needs to be identified. Depending on the complexity of the evaluation, software architects need to provide guidance for their organization regarding the tool – whether to create their own tool or buy an existing one.

3.4 Level 5 - Analysis: Building a Meta-Model as a Foundation for Analysis

If SLA violations take place, the root cause for the violation needs to be identified. This could be done, for example, with the help of historical data. An even more advanced

step would be to predict whether certain SLAs will be violated in the near future. In that case, it might even be possible to take countermeasures in order to prevent such SLA violations. All these analysis activities require a runtime view on the system. This view needs to include not only the runtime architecture of the system, but all the environmental and associated aspects as well. A meta-model is crucial as a foundation for all those analyses (see example in **Fig. 7**). Below we briefly describe the process of creating such a meta-model.

- Identifying the software elements and their relationships: To build a runtime view of the system, a software architect needs to identify the software elements and their relationships. This includes software components, interfaces, dependencies, deployment artifacts, and mapping of the software components to the deployment artifacts.
- Identifying the runtime context of the software system: The runtime context of the system is also necessary for evaluation. It includes, for example, the load, the current SLA status, and environmental aspects.
- Identifying the development artifacts: The development artifacts – for example release dates, modification dates, code, configuration, etc. – need to be identified, including their relationships.
- Mapping software elements to development artifacts: To relate the system to its development artifact or process, software architects need to provide a mapping of the software elements to the development artifacts. This is a prerequisite to allow identifying what caused problems during development and operation and what has to be improved.

4 Case Study and Experiences

In order to validate our ideas, we performed a case study with vwd GmbH. The vwd GmbH mainly functions as a data provider to banks and other financial organizations. We took one of their SOA systems that provides financial market data to external systems (owned by their customers). The internal architecture of the system is shown in **Fig. 4**.The system has a message broker architecture and uses several services in the service layer that collect and provide specific types of data. These services can be consumed asynchronously by sending messages to the Message Broker (in the communication middleware layer). In the application layer, requests from external systems arrive at the Molecule Composition component. Depending on the request, it creates/instantiates several atom controllers responsible for one specific type of data retrieved from the services of the service layer. The Display Configurator is used to format the output based on the configuration of the requesting user once all atom controllers have provided data to the Molecule composition component. Obviously, the service component and all other components use resources during runtime and are often replicated to achieve the desired quality.

We used the maturity model to evaluate the organization's level and to set goals for SLA enforcement. The maturity model did indeed help to identify that. The organization already had SLAs, but not formalized, along with a standard monitoring system in place and could therefore be considered about level 2 of the maturity model. They plan

to achieve customer-specific SLA compliance status and to make root cause analysis faster and less dependent on experts.

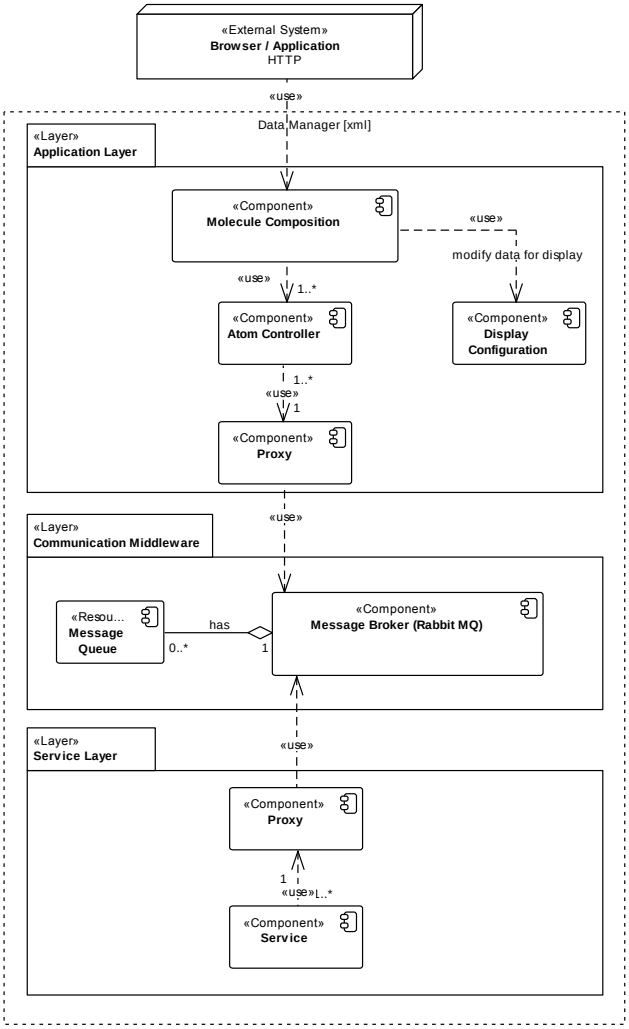

Fig. 4. Architecture of the Case Study System

4.1 Level 3 – Formalization

Following the steps in 3.2, we came up with the SLA formalization shown in **Fig. 5** as output of our case study. As a pilot, we considered simpler SLAs for latency and availability. As the formalization is a gradual process, it will be different when we include other quality attributes.

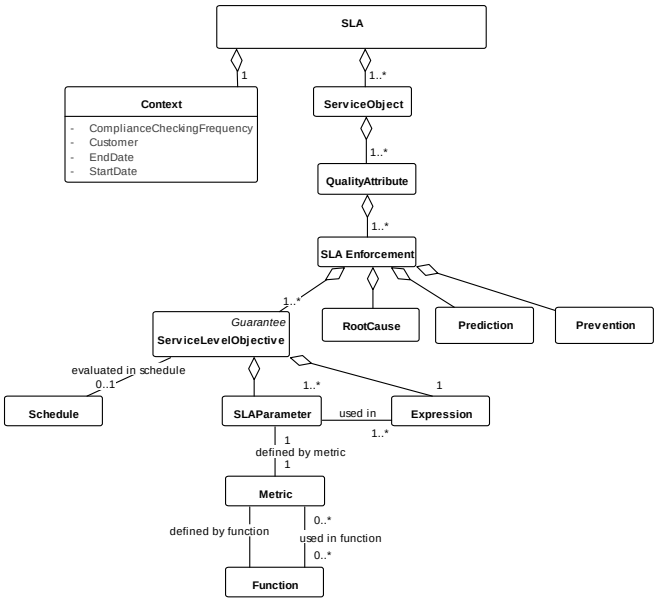

Fig. 5. Formalization of SLAs

Experiences

- *Was the approach applicable in the scenario?* A prototypical implementation for formalizing the SLAs worked and validated the approach. The prototype was realized using xsd/xml techniques to define and instantiate the DSL. The formalized SLAs could also be used to generate text sections to use for the technical/measurable part of the SLA definition contract paper.
- *What were difficult/ easy parts?* It was easy to implement a first xml-based definition of (simple) metrics (as in our prototype), which are interpreted in Java. Existing SLAs (which were not formalized or evaluated automatically) helped to define a first version. So it is easier if the company already has experience with SLA reporting in general.
- *How much effort was required?* It is easy enough to start with formalization and compliance checking when standard monitoring is in place. As the effort for creating a new DSL does not seem to be too high, it is better to start with something than to search the market for the silver bullet.

4.2 Level 4 - Evaluation: Compliance Checking

Fig. 6 shows the compliance checking system structure. In the case study, we implemented the key part of it. Therefore, the necessary system components were implemented in a prototypical sense using Java. Using the steps described in 3.3, we identified what has to be monitored and how.

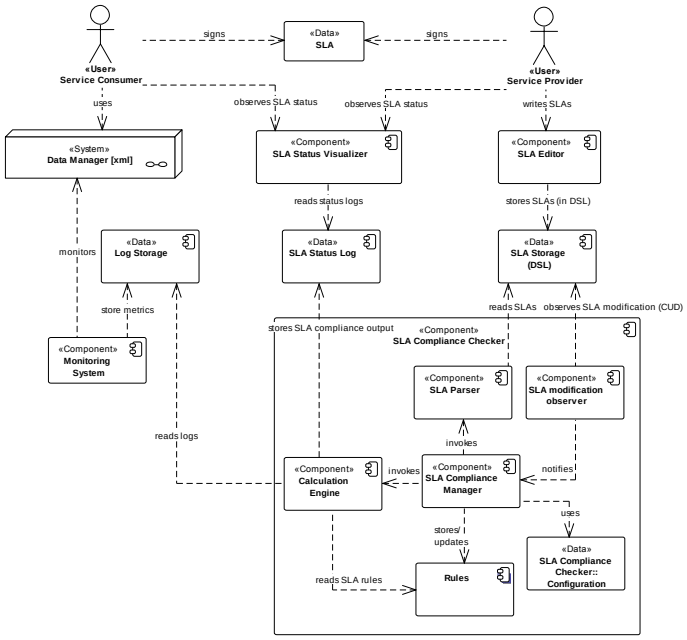

Fig. 6. SLA Compliance Checking System

Experiences

- *Was the approach applicable in the scenario?* Yes, the approach worked. Especially the part of treating the system from a customer perspective helped to get a sharpened impression of SLA compliance. As our considered (current) SLAs were not too complicated, it was not too difficult to identify the metrics and the monitoring requirements and define the mapping to SLOs. For most of the metrics, it was not a big problem to monitor as standard http log files and a (simple) self-made checking tool for the customer perspective could be used.
- *What were difficult/easy parts?* With a general monitoring system in place, it is not too difficult to further formalize and automate more SLA-relevant aspects. But it was difficult and has not been solved yet to synchronize external measurements with internal data (e.g., the external system says “not available”, but internal measurement says “ok”).
- *How much effort was required?* First steps towards compliance checking are more than feasible for an SME.

4.3 Level 5 - Analysis: Building a Meta-Model as a Foundation for Analysis

Fig. 7 shows the meta-model that connects the software elements at runtime, the resources, the development artifacts, and the runtime context. A meta-model is crucial for a software architect to get a clear picture of the connection and relations between

the runtime, devtime, and operational aspects of a software system. We developed the meta-model primarily to narrow down our searches for root cause analysis.

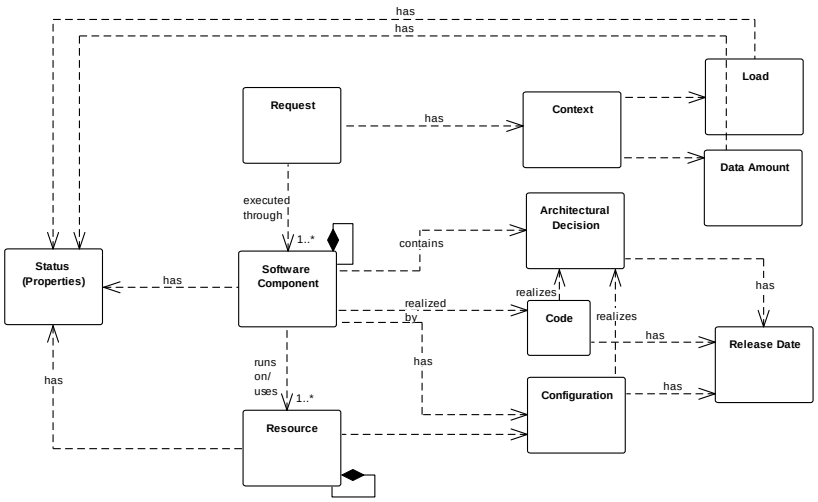

Fig. 7. Meta-Model for Connecting Runtime, Devtime, and Operational Aspects

Experience.

- *Was the approach applicable in the scenario?* The approach was especially useful as we do have a distributed system that, at least in its runtime configuration, changes occasionally. It was also helpful to relate software code changes to parts of the system with a monitored problem to detect the source of a problem faster (and thus to find the solution faster). It should be doable once there is a runtime architecture with specified relationships to the code used by the respective part of the system.
- *What were difficult/ easy parts?* It was easy to attribute the executed Java code and the communication infrastructure (which is based on messaging) to automatically derive information for building the runtime architecture (it is at least possible in modern platforms like Java (as in our case) or .Net). We have built the meta-model but we do not have an implementation to build the runtime architecture. So at this point it is not possible to judge the difficulty.
- *How much effort was required?* Full implementation of the meta-model building approach will result in quite some effort.

5 Takeaways for SMEs

- **No existing DSL is a silver bullet.** There seems to exist no silver bullet DSL, and overhead for creating your own is not high. Creating a new DSL could serve your needs exactly and is probably simpler than adapting to and working around problems with any existing DSL. So create a new one for the prototype and, with a clearer understanding of the needs, re-evaluate the market afterwards.

- **Do not spend too much time on creating a perfect DSL for formalizing SLAs.** Try to use a dynamic language like groovy to allow for simple definition of evaluation expressions. You may start with an xml definition for quality metrics and conditions and groovy-based evaluation expressions that operate on the raw data provided by the monitoring system based on the quality metrics used.
- **Formalization is key, especially for automation (and automation is a must)**. It is relevant for an in-depth look at the SLA status, even in real time, for customer access and for deriving analysis tools. It is input to monitoring - especially monitor in more detail internally what you are measuring against externally (for the SLAs).
- **Automated/formalized definition of SLA criteria is needed to offer customer-individual SLAs.** The idea of offering customers a tailored SLA with a connection to what is negotiated in the contract and to the real software system is great (as for an SME, this is a good opportunity to be better than a larger market competitor). This can be done using a formalized definition and automated compliance report.
- **Just by working on formalization and making it more explicit, there is an improvement in the internal qualities of the system.**
- **Plan for manual interaction.** It requires very little effort but is still relevant. For example, negotiated availability with customer should overrule what is monitored. Sometimes measurements are not correct and need overruling (e.g., for problems with the measurement itself).
- **Real-time view on SLA status can be valuable for internal planning.** For example, do not plan a major update (which increases the risk of problems because of system changes) at the end of SLA reporting phases when the quality metrics are in danger. It is better to postpone the update to the start of the next reporting period.
- **Centralized logging is helpful for distributed systems.** Open source components are available, like redis, logstash, ElasticSearch-based systems, etc.
- **Unique request IDs are key to building a runtime meta-model.** All loggings by the software components and the resources should include that ID.
- **Use aspect-oriented techniques to automatically collect data (number of requests, latency, throughput, etc.).** It will keep the actual code clean.
- **Establish "Analyzability" as a relevant non-functional requirement of development.** Software architects should establish and propagate this relevance, for example by integrating a unique request ID into each log statement for each distributed service involved in request evaluation.

6 State of the Art and State of the Practice

6.1 Research on SLA

Research has been done on SLA enforcement from various perspectives. ITIL [13], CMMI [14], Oracle [1], and others have presented overall service level management, governance, and best practices. ITIL also has a service level management (SLM) maturity model [6]. But the focus of those practices and maturity models is on any service in general; they are not tailored to a software system. Moreover, they discuss overall

service level management and service level enforcement is just mentioned as a small part. In our paper, we detailed the enforcement aspects of SLM. SEI's technical report [3] presents the main concept of SLA for SOA systems and also points out which metrics are measurable. Recently, Becha [2] did research on the qualities that are relevant from the consumer's point of view. A number of DSLs have been proposed for SLA specifications [7–9, 11]. Tang et al. [16][15] describe SLA-aware service computing techniques, and Pereira et al. [12] and Motta et al. [10] describe SLA enforcement for large organizations [10]. But the overall holistic approach for SLA enforcement for an SME's software system is not covered in the literature. Grzech et al. [5] describe the translation of SLAs into complex structures [5] and Emeakaroha et al. [4] describe how low-level metrics could be mapped to high-level metrics [4]. But we definitely need mapping from high-level metrics to low-level metrics as well. This has not been addressed thoroughly in our research yet either, but will be part of our future research.

6.2 Industrial Tools

There are plenty of monitoring tools and calculation/reporting engines available. Monitoring tools usually monitor various resources. Nagios, Shinken, etc. are examples of such kinds of IT infrastructure monitoring tools. Usually these tools are independent of what is stated in the SLAs, but they can alert to or report infrastructure level anomalies.

We also identified several SLA monitoring tools from the market – for example, AmberPoint's SOA, BMC Software's AppSight, CA's Willy SOA, Fushion Point's BSI, HP's SOA Manager, IBM's Tivoli, Progress Software's Actional, Tidal Software's Interperse, TIBCO's SOA, WSO2's Governance Registry, SLA Diator, and so on. We interviewed two of them. What we discovered is that they are mostly designed for IT-helpdesk-like SLAs in the sense of tracking and managing SLAs on the support level. The tools usually include monitoring, calculating, and reporting capabilities. But they are not tailored for software system SLA management, integration, and enforcement. These tools cannot identify what has to be monitored on the resource level or on lower levels based on the SLAs, and they are not smart enough to map external metrics to internal metrics. Therefore, currently available tools lacks the power to come up with a good mechanism for prediction or root cause analysis as they cannot create the runtime architecture of the system to discover improvement potential.

7 Conclusion

SLAs and software systems offered as services are not integrated well. The service provider is supposed to establish measures to enforce the agreements systematically, but often this is not handled with priority as it is not the main functionality of the system. This technical debt makes the provider dependent on highly experienced people.

Therefore, we developed the maturity model to analyze the SLA enforcement situation and goals. This is a first step towards supporting architects in this direction. This model needs to be validated in the context of other organizations and needs to be enriched and detailed in the future to enable further adoption in industry.

We described three transitions of the maturity model in this paper: formalization, evaluation, and analysis. Using a case study with an SME, we showed the applicability of the approaches for latency- and availability-related SLAs. More quality attributes need to be integrated in the future. There are still a lot of open questions left – for example: how to efficiently map the external metrics to internal metrics; how to do predictions of the future; how to narrow down the search for root cause analysis in case of violations; how to make the development and operation activities SLA-aware. In the future, our main focus will be on building a runtime architecture of the system that captures the runtime situation and helps to comply with SLAs as a consequence.

Acknowledgment: We would like to thank the German Ministry of Education and Research for funding parts of this work under grant number 01IC12S01F.

References

1. Afshar, M. et al.: SOA Governance : Framework and Best Practices SOA Governance : Framework and Best Practices. Gov. An Int. J. Policy Adm. 6, 3, 1–17 (2007).
2. Becha, H., Amyot, D.: Non-functional properties in service oriented architecture - A consumer's perspective. J. Softw. 7, 3, 575–587 (2012).
3. Bianco, P., Lewis, G.A.: Service Level Agreements in Service-Oriented Architecture Environments. SEI Tech. Note. 2008-TN-02, September 2008, (2008).
4. Emeakaroha, V.C. et al.: Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. Proc. 2010 Int. Conf. High Perform. Comput. Simulation, HPCS 2010. 48–54 (2010).
5. Grzech, A., Rygielski, P.: Translations of service level agreement in systems based on service oriented architecture. Knowledge-Based Intell. Inf. Eng. Syst. 6277, 523–532 (2010).
6. Hoc, A.: ITSM Maturity Model, http://www.tarrani.com/pix/ITSMMaturityModel_v3.PDF.
7. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. J. Netw. Syst. Manag. 11, 1, 57–81 (2003).
8. Kübert, R. et al.: A RESTful implementation of the WS-agreement specification. Proc. Second Int. Work. RESTful Des. (WS-REST '11). 67–72 (2011).
9. Lamanna, D.D. et al.: SLAng: a language for defining service level agreements. Ninth IEEE Work. Futur. Trends Distrib. Comput. Syst. 2003. FTDCS 2003. Proceedings. 34069, (2003).
10. Motta, G. et al.: IT Service Level Management: Practices in Large Organizations. Commun. IBIMA. 2011, 1–9 (2011).
11. Paschke, A. et al.: ContractLog: An approach to rule based monitoring and execution of service level agreements. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 3791 LNCS, 209–217 (2005).
12. Pereira, P.R.: Service Level Agreement Enforcement for Differentiated Services. Proc. 2nd Int. Work. EURO-NGI Netw. Excell. 3883, (2005).
13. Pereira, R., Da Silva, M.M.: A maturity model for implementing ITIL V3 in practice. Proc. - IEEE Int. Enterp. Distrib. Object Comput. Work. EDOC. 259–268 (2011).
14. Services, C.: CMMI® for Services, Version 1.3 CMMI-SVC, V1.3. (2010).
15. Tang, L. et al.: SLA - Aware Enterprise Service Computing - Part II, (2013).
16. Tang, L.: SLA-Aware Enterprise Service Computing - Part I, (2013).

Document Information

Title:	Engineering Service Level Agreements as an Integral Part of Software Systems and their Architectures
Date:	December 2015
Report:	IESE-057.15/E
Status:	Final
Distribution:	Public Unlimited