Translation of Service Level Agreements: A Generic Problem Definition

Constantinos Kotsokalis[1] and Ulrich Winkler[2]

[1] Dortmund University of Technology, Germany
constantinos.kotsokalis@udo.edu
[2] SAP Research, Belfast, UK
ulrich.winkler@sap.com

Abstract. A Service Level Agreement (SLA) is the electronic equivalent of a real-life contract, which describes expectations from a service and governs its consumption. Ideally, a SLA provides certainty as regards customer experience and Quality of Service (QoS) received. For self-contained, isolated services this type of certainty is relatively straight-forward to achieve. However, for services that are composed by others, or that rely on others to execute, such functional dependencies imply similar non-functional ones. Therefore, SLAs offered by a service to its customers are in turn depending on other SLAs, which the same service establishes in its role as a customer of the services it relies upon. In this paper we explore this dependency between different SLAs, and formalize the problem of converting an agreement for a composed service into individual agreements for the services from which it is composed.

1 Introduction

Service composition and re-use is the cornerstone of Service-Oriented Computing. Through service composition, it becomes possible to take advantage of the expertise that each provider offers through its products. At the same time, the provisioning, execution and access of a service may presume the existence of infrastructural services, which are used explicitly or implicitly. In both cases, there are dependencies between the composite (or higher-level) services, and the composed (or infrastructural) services. In the text that follows, we will refer to the former as *dependents*, and the latter as *antecedents*. Such dependencies *may* directly affect the operation of a dependent if non-functional expectations of its antecedents are not met. For instance, a company's web server will not be accessible to the outside world if the company's network fails. If, however, such a company is connecting to the internet through two different links and one of them fails, then the web server will still be accessible to the rest of the world through the second connection. Clearly, finding the dependencies and documenting them in detail is not always straight-forward, especially in complex systems with a multitude of inter-dependent services.

Nevertheless, when a provider needs to augment its services with guarantees towards its clients, this automatically means that guarantees must also be provided by the respective antecedents. The exact way that such guarantees are

A. Dan, F. Gittler, and F. Toumani (Eds.): ICSOC/ServiceWave 2009, LNCS 6275, pp. 248–257, 2010.

related is use-case specific and cannot be universally defined for an application domain, as is also evident from the example of the previous paragraph. However, a formalization of the problem is much needed as a first step for translating the guarantees that a dependent service offers to guarantees that it requires from its antecedents. This paper provides a means to address this open scientific problem, using *Service Level Agreements* (SLAs) as an instrument. SLAs are electronic contracts, that specify what a customer should expect from a service he/she consumes. These expectations may concern both functional and non-functional service properties, which are grouped in a single reference document (the SLA) as it results from negotiation between the parties involved.

The scientific contribution of the present work is *a generic definition of the SLA translation problem*, and *an abstract description of how it needs to be addressed during negotiation time.*

The paper is organized as follows: Section 2 provides information on prior related work. Section 3 discusses the topic of service dependencies, followed by Section 4 that elaborates on the dependencies of service properties and therefore SLAs. Next, Section 5 discusses SLA translation. Finally, Section 6 concludes the paper with a summary and outlook on future work.

2 Related Work

Although not directly related to SLAs, there is prior art that concerns relationships between different services and/or resources. In [1], the authors describe a *Functional* and a *Structural* dependency model, on which they base their work. They provide a specific classification of dependencies, into a number of dimensions such as locality, time, type, dependency strength and criticality, etc. Then, building on the two models, they discuss a method for dependency analysis. In [2] the same authors append the two models with an *Operational* one, and further refine a proposed architecture for managing dependencies. The main differences with our work is that they constrain theirs in the ICT domain where they also provide an architecture to tackle the dependency discovery problem. Additionally, they do not consider SLA management.

In [3] the author discusses the problem in a generic manner, much like our paper. However, the work is tied to software services; more specifically, it proposes a scheme where services declare their dependencies on other services based on access patterns (how often are services accessed, in what sequence, etc) with dynamic service selection in mind. Additionally, it looks specifically at functional dependencies. Contrary to that, dependency management in the context of SLAs requires a generic view that handles non-functional dependencies as well.

[4] is looking at the same problem as regards network services. The authors describe a model called *Inference Graph* that represents the dependencies. Based on that, they present an algorithm for inferring probabilistically the malfunctioning components of the network, given real-world observations. The main difference with our work is that the Inference Graph describes dependencies based on service states, and probabilities that the state of one depends on the state

of another; additionally, the referenced work is explicitly addressing network services.

In [5] the authors introduce an agent-based automated SLA negotiation architecture and discuss efficient search-based algorithms to determine an acceptable Service Level Agreement in a multi-agent environment. The main difference to our work is that we assume a complex and layered multi-tier services landscape that requires additional analysis steps and related data-structures for SLA negotiation. In addition, we consider SLA translation as an essential part of the SLA negotiation process. Although some of the search-based algorithms analyzed in [5] would be able to cope with multiple objectives that negotiation agents may have, it is not the primary concern of the authors' work.

In a purely IT context, there have been efforts to address similar problems for modeling, analyzing and converting metrics and properties (e.g. response time and throughput) into concrete system configuration descriptors (e.g. number of servers or system policies). Performance engineering, capacity planning and configuration management are IT areas concerned with this specific problem. Layered queuing network (LQN) models [6,7] are, for example, an important modeling and analysis concept in performance engineering. LQNs are used to model layered client-server architectures, where a server can become a client to other servers and servers are mapped onto resources, such as CPUs or storage. Analytic solvers or simulations are used to estimate resource utilization. Other approaches are probabilistic models, reinforcement learning [8] and Bayesian networks classifiers [9] to map system properties and metrics to system configurations. The main difference to our work is that we address a formal specification of SLA metric-to-metric translation problem in service oriented architectures, and we do not target capacity planning or performance engineering specifically for IT systems.

3 Service Dependencies

As already mentioned in Section 1, service dependencies may be *explicit*, or *implicit*. Explicit dependencies play a central role in Service-Oriented Computing. The assumption here is that a service's logic (the functionality it delivers) depends on some other service by means of composition. If we assume that a service implements an algorithm, and that an algorithm is represented by a series of atomic operations (*instructions*) as shown in Figure 1, then one or more of those instructions are carried out by the services on which this former service depends on. A typical example are business processes, usually modeled as workflows, that bind to external services and invoke them during execution. Complete outsourcing of one's work is also an example; in this case, a service provider who cannot satisfy the demand of its customers, acts as a customer to another service provider and delegates some of its work to this latter provider. A service modeled as part of a composition may or may not be actually used, depending on the composition's logic. It is use-case-specific details which define if a service is invoked or not. Still, dependencies exist in the form of *candidate invocations*.

Figure 2 illustrates a composed service ("*Composition*") that relies on four different services ("*A*", "*B*", "*C*" and "*D*"). The instructions that correspond to the composite service are representing the glue code for the composition, which performs invocations, data staging and similar tasks.

Fig. 1. A series of abstract instructions implementing an algorithm

Implicit dependencies are, typically, dependencies related to infrastructure services (without which a higher-layer service cannot operate at all), and services used for reasons of redundancy. For instance, any web server presumes that a working DNS system is available so that clients can access it. Any software relies on some (physical or virtual) hardware on which it can execute. When it comes to redundancy, the standby services set up for this reason are normally not used, but their existence affects the service that depends on them, in the case of a failure. Figure 2 illustrates the concept, with two infrastructure services used from the higher-level software services.

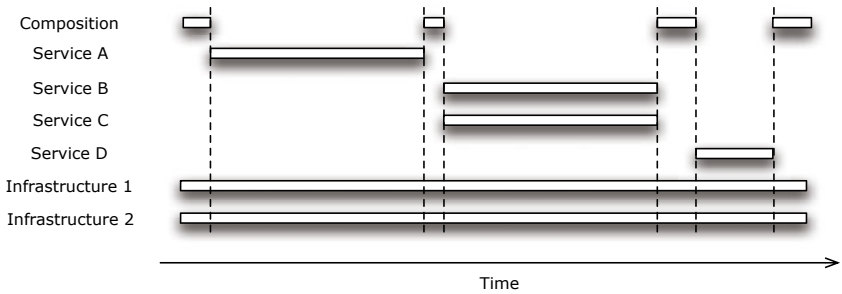

Fig. 2. Example service dependencies: Compositions and infrastructure

Although we can perform this rough classification of explicit and implicit dependencies, it is not possible to refine it further and try to identify in greater granularity the relations between dependent services. Rather, the notion of *dependency* itself is enough to construct a *dependency graph* [10,11] as a first step. This dependency graph can then be elaborated in the context of SLAs, as we will see in the upcoming sections, and function as a basis for the translation process. Figure 3 illustrates an example service dependency graph, where each vertex represents a dependency (direction is from dependent to antecedent).

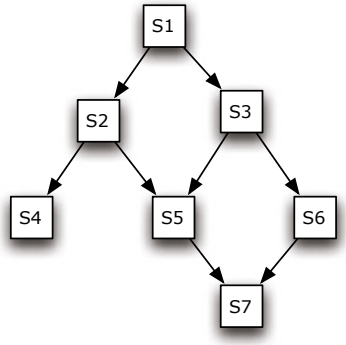

Fig. 3. Example service dependency graph

4 Service Properties Dependencies

We define a SLA as a set of guarantees over the consumption of a service. Often, these guarantees may be some facts universal to the SLA, or may refer to properties of the service that the SLA concerns. For instance, if we assume a software service, its availability is one of its properties. A threshold for this availability in the form of a guarantee towards customers, may constitute part of a SLA.

Dependency of a service upon others, naturally means that its properties are depending on properties of these other services. Assuming the availability example for a depending service, it will be affected by the availability of its antecedents. However, there might be a case where a dependent service's property may be related to a completely different property of its antecedents. For instance, the cost of a service is typically affected by the quality of the services that it uses for completing its tasks. Additionally, it is certainly possible to have a SLA which does not refer to all properties of a service, and therefore we only need to take into account a limited number of dependencies: only those which affect the properties mentioned in the SLA.

It is becoming clear that there cannot be a universal classification of service properties, and their dependencies. Certainly, facts that set the context of a SLA cannot apply to all SLAs either, but we should rather assume distinct facts for distinct SLAs. We can only define the problem generically based on the abstraction of conditions that need to hold true, when aggregated according to the SLA structure. Use-case-specific knowledge needs to be applied by domain experts to instantiate these conditions, and associate them in the context of different, dependent services.

Starting from the service dependency graph, we can make a next step towards a service *Properties Dependency Graph* (PDG). Let us assume a service S_i, with properties $P^{S_i} = (p_1^{S_i}, p_2^{S_i}, ..., p_m^{S_i})$, and a service S_j, with properties $P^{S_j} = (p_1^{S_j}, p_2^{S_j}, ..., p_n^{S_j})$. If S_i depends on S_j, then we can formulate a dependency of S_i's r-th property as a function f_r of properties of S_j:

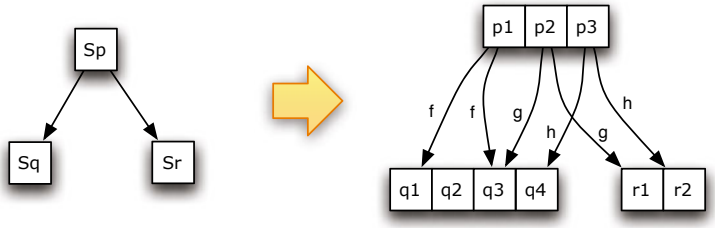

Fig. 4. Example service Properties Dependency Graph

$$p_r^{S_i} = f_r(p_1^{S_j}, p_2^{S_j}, ..., p_q^{S_j}) \tag{1}$$

where $1 \geq r \geq m$, and $1 \geq q \geq n$.

This definition can be extended for a service S that depends on more than one services. Let us assume that S has properties p_1^S, p_2^S, ..., p_N^S. Let us also call the antecedents S_1, S_2, ..., S_m, with property sets $P^{S_i} = (p_1^{S_i}, p_2^{S_i}, ..., p_{N_i}^{S_i}), 1 \leq i \leq m$. Then, each property p_r^S of service S can be generically expressed as follows:

$$p_r^S = f_r(p_1^{S_1}, ..., p_{N_1}^{S_1}, p_1^{S_2}, ..., p_{N_2}^{S_2}, ..., p_1^{S_m}, ..., p_{N_m}^{S_m}) \tag{2}$$

Figure 4 illustrates such an example. Here, service *Sp* depends on services *Sq* and *Sr*. Domain experts can then create rules about the dependencies of *Sp*'s properties *p1-p3* on those of *Sq* (*q1-q4*) and *Sr* (*r1-r2*), based on modeling techniques, simulation, or real-world observation. These rules are represented by functions f, g and h that implement a mapping of each property on the properties it depends.

5 SLA Translation

Operating under the abstractions from Sections 3 and 4, we can discuss about the concept of *SLA translation*. In essence, SLA translation is the process of analyzing a SLA in relation to the PDG, applying heuristics and pre-existing knowledge, and coming up with one or more subsequent SLAs for antecedents. These subsequent SLAs provide reasonable (but not complete) certainty that the top-level SLA will not be violated, unless some of them are violated too. We assume all SLAs to be negotiated using some well-specified language and protocol, such as WS-Agreement [12] – probably with an extension that supports counter-offers.

Starting from Equation 2, one can first define a complete set of such equations, one for each property that the customer of the dependent service requires in the negotiated SLA. Properties that the customer does not require guarantees for, can be excluded. Then, this system must be solved, taking into account existing constraints. For that, we presume that services expose *SLA templates*, which are used exactly to guide the negotiation process with initial values, and indicate

the constraints of negotiable parameters. For instance, a provider may be unable (or unwilling, for business reasons) to accept a SLA request about a service with the "*availability*" property set to 100%, although availability level is up for negotiation. In this case, the provider would publish a template, where the upper limit would be clearly stated, e.g. 99.5%. The limits as regards what can be negotiated help customers to reduce the search space for their offers. This search space may otherwise be so large that the problem becomes practically infeasible from a computational point of view.

Optimization of subsequent SLAs is one more requirement for the translation process. As a matter of fact, optimization is part of translation, as it affects the final output. The system mentioned above will typically accept plenty of different solutions, and it is up to optimization to select the best of all those candidates. As we deal with many properties simultaneously, this becomes a multi-criteria optimization problem [13,14].

Using the example from Figure 4, we have the following relations:

$$p1 = f(q1, q3)$$
$$p2 = g(q3, r1)$$
$$p3 = h(q4, r2))$$

If an incoming SLA for service *Sp* refers to all three properties, a SLA negotiation mechanism which includes translation functions should first find out the dependencies of these three properties based on the PDG. We assume, as also mentioned earlier, that these dependencies are known as domain-specific expertise encoded into our system. For instance, if the property is "*availability*", it will typically depend on the availability of all antecedents; this is fairly straightforward to assume. However, if we examine cost, it is not necessary that the cost of invoking service *Sp* is related specifically to cost properties of services *Sq* and *Sr*. On the contrary, it may be the case that there are no cost properties for these two services, but rather that their providers only apply flat-rate pricing. In this case, the cost of *Sp* may rely on properties such as Quality of Service characteristics of *Sq* and *Sr*.

The next action should be to find acceptable value spaces for all of $(q1, q3, q4, r1, r2)$. By "acceptable", we refer to values which remain within constraints set inside the templates of the two lower-level services, and on the same time satisfy the requested values in the offer for the higher-level SLA.

The last step during the translation process, would be to come up with an optimal solution to the problem, according to a multi-criteria optimization algorithm, which perhaps takes into account business objectives such as profit maximization. It may well be the case that there exists no single optimal solution. In this case, one of the solutions on the *Pareto front* should be chosen. The Pareto front is a set of all those solutions that are considered to be optimal in multi-criteria optimization. More formally, they are solutions where none of the included criteria can be improved (accept a better value), without some other criteria in the same solution receiving a worse value.

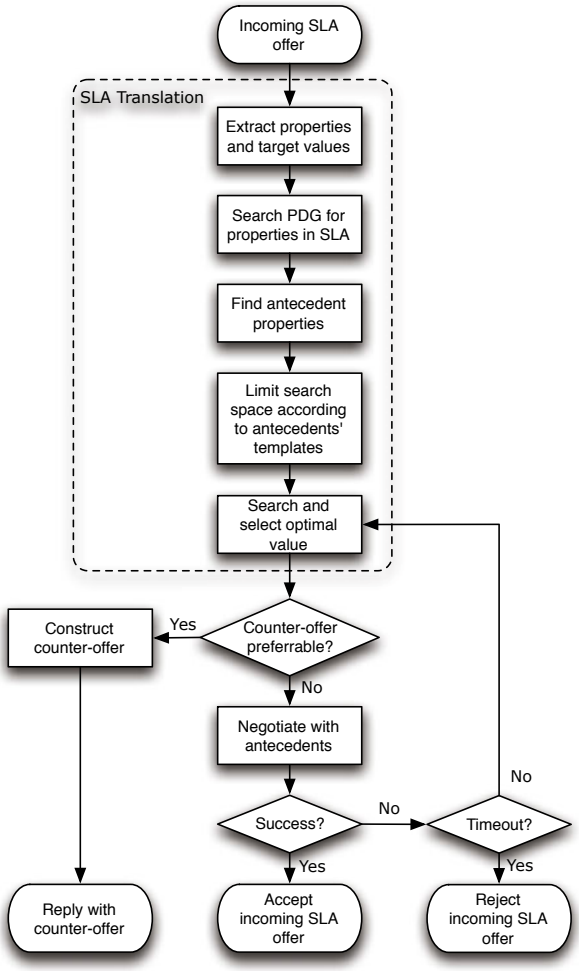

Fig. 5. Generic flowchart for translation/negotiation of SLAs

Figure 5 shows how this translation process can be implemented according to the previous discussion. We are assuming that counter-offers are possible, as part of the negotiation protocol being used. The reason that we are interleaving translation and negotiation in this flowchart, is exactly that they take place in parallel and depend on each other. Negotiation feeds translation with confirmations whether the translation's results are acceptable; and translation provides negotiation with input according to existing domain-specific knowledge, and any further intelligence implemented within the system. It should be clarified, at this point, that we are separating between the negotiation process and all kinds of decision making; that is to say, we assume negotiation concerns only the exchange of messages in such a way that implements correctly a specific protocol, while all

planning functionality lives elsewhere and is, in this case, part of the translation process. Regarding planning itself, we anticipate that coping with multiple service level attributes and n-dimensional Pareto fronts might become difficult. We assume that –in order to be able to implement an efficient and automated SLA translation and negotiation framework– use-case dependent heuristic functions will be required to narrow the search space down to a feasible size.

One last thing to mention here, is that in this case we are tackling the problem of producing lower-level SLAs starting from a higher-level one, in a top-down approach. A different scenario is that of a bottom-up approach, where the lower-level SLAs are already established and used to decide whether an incoming higher-level offer can be satisfied. Although this is a valid and realistic scenario in the business world, in this paper we have set to define the process for a completely dynamic setup of SLA hierarchies, which may help to avoid under- and over-provisioning. Nevertheless, the essentials of our previous discussion on producing and using PDGs for SLA translation can be applied just as well on such a bottom-up process, as they are modeling service relationships without being tied to a specific usage scenario.

6 Conclusions

In this paper we discussed the topic of service dependencies, and how they reflect onto dependencies of respective SLAs. More specifically, we described service dependencies in terms of Dependency Graphs, and extended this data structure into Properties Dependency Graphs. Based on the latter, we defined what SLA translation means, and illustrated its relation to SLA negotiation.

The process of translating SLAs for dependent services into SLAs for antecedents requires domain-specific and use-case specific expertise, for producing the conversion rules and for fine-tuning them. In some occasions, historical data may provide insight, while in others this will be a modeling task.

As part of our continuing work within the SLA@SOI EU project, we are researching structures and algorithms that can express these concepts and ease the computational task of producing optimal SLA offers towards antecedents, during negotiation. An architecture that takes into account these requirements has already been produced, and is being currently implemented.

Acknowledgements

The research leading to these results is supported by the European Community's Seventh Framework Programme (FP7/2007-2013) and the SLA@SOI Integrated Project under grant agreement no.216556.

References

1. Keller, A., Blumenthal, U., Kar, G.: Classification and computation of dependencies for distributed management. In: IEEE Symposium on Computers and Communications, p. 78 (2000)

2. Keller, A., Kar, G.: Determining service dependencies in distributed systems. In: IEEE International Conference on Communications (ICC 2001), vol. 7, pp. 2084–2088 (2001)
3. Hasselmeyer, P.: Managing dynamic service dependencies. In: 12th International Workshop on Distributed Systems: Operations & Management (DSOM 2001), pp. 141–150 (2001)
4. Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D.A., Zhang, M.: Towards highly reliable enterprise network services via inference of multi-level dependencies. In: SIGCOMM 2007: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 13–24 (2007)
5. Di Nitto, E., Di Penta, M., Gambi, A., Ripa, G., Villani, M.: Negotiation of service level agreements: An architecture and a search-based approach. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, p. 295. Springer, Heidelberg (2007)
6. Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S.: The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. IEEE Trans. Computers 44, 20–34 (1995)
7. Petriu, D., Woodside, M.: Analysing Software Requirements Specifications for Performance. In: Third Int. Workshop on Software and Performance, pp. 1–9 (2002)
8. Tesauro, G., Jong, N.K., Das, R., Bennani, M.N.: On the use of hybrid reinforcement learning for autonomic resource allocation. Cluster Computing 10(3), 287–299 (2007)
9. Kumar, V., Schwann, K., Iyer, S., Chen, Y., Sahai, A.: A State Space Approach to SLA based Management. In: IEEE/IFIP NOMS (2008)
10. Katzela, I., Schwartz, M.: Schemes for fault identification in communication networks. IEEE/ACM Transactions on Networking 3(6), 753–764 (1995)
11. Gruschke, B.: Integrated event management: Event correlation using dependency graphs. In: Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 1998), pp. 130–141 (1998)
12. Open Grid Forum: Web Services Agreement Specification, WS-Agreement (2007)
13. Sawaragi, Y., Nakayama, H., Tanino, T.: Theory of Multiobjective Optimization. Mathematics in Science and Engineering, vol. 176. Academic Press, London (1985)
14. Ehrgott, M.: Multicriteria Optimization. Springer, New York (2005)