Service-Level Agreements for Electronic Services

James Skene, *Member*, *IEEE*, Franco Raimondi, and Wolfgang Emmerich, *Member*, *IEEE Computer Society*

Abstract—The potential of communication networks and middleware to enable the composition of services across organizational boundaries remains incompletely realized. In this paper, we argue that this is in part due to outsourcing risks and describe the possible contribution of Service-Level Agreements (SLAs) to mitigating these risks. For SLAs to be effective, it should be difficult to disregard their original provisions in the event of a dispute between the parties. Properties of understandability, precision, and monitorability ensure that the original intent of an SLA can be recovered and compared to trustworthy accounts of service behavior to resolve disputes fairly and without ambiguity. We describe the design and evaluation of a domain-specific language for SLAs that tend to exhibit these properties and discuss the impact of monitorability requirements on service-provision practices.

Index Terms—Service-level agreements, electronic services, contracts, domain-specific languages, model-driven engineering.

1 Introduction

Application-service Provision (ASP) [1], Software as a Service (SaaS) [2], utility computing [3], cloud computing [4], virtual enterprises [5], Service-Oriented Architecture (SoA) and Service-Oriented Computing (SoC) [6], Web services [7], the World-Wide Web (WWW) [8], object-oriented middleware, Remote-Procedure Calls (RPC) [9], and client-server architectures [10] are all paradigms for the implementation of distributed systems which, although they do not all benefit from unambiguous or definitive coinage, clearly share two distinctive features: They rely on the exchange of service request messages between distributed client and service software agents, sometimes resulting in an exchange of response messages delivering part or all of the value of the service being implemented, and they may be federated, i.e., the client and service agents may be the responsibilities of distinct, financially independent parties. In this paper, we refer to such systems as *electronic services*.

The benefits of these paradigms are various. However, it is commonly stated that the division of logic between the client software and one or more services enables a beneficial separation of concerns between the provision of the archetypical services and the employment of those services to some business end. This results in IT infrastructure that is organized according to the logical structure of the business or task being automated, and is therefore more reusable, maintainable, and amenable to the continuous integration efforts required by ever-changing software trends and corporate mergers [11].

This argument is reinforced by the potential for the federation of such systems: Large organizations rely on a variety of services; this presents a management challenge; moreover, it is difficult to rely on competition to improve the quality/cost ratio of internal services. Therefore, a recent management trend has been to attempt to outsource services unrelated to an organization's core competencies, to simplify management and promote a competitive market for services. Notable examples are Amazon's Elastic Compute Cloud [12], Salesforce.com [13], and SAP's Enterprise SOA [14].

The potential for outsourcing is taken to its extreme in the Web-services vision [7]. Here, it is proposed that outsourcing can be achieved at the component level, with Web-service rental representing a viable alternative to the purchase of off-the-shelf libraries or packages. Under the Web-services model, a system may be composed across multiple organizational boundaries, with outsourced services in turn outsourcing elements of their own functionality.

Although the architectural benefits of client-server computing have been extensively realized, as demonstrated by the enormous market in middleware software, the practice of outsourcing electronic services is far less ubiquitous. Certainly, the infusion of the Internet with functionality, in addition to media content, is still not a widespread reality.

We argue that financial risks associated with the outsourcing of electronic services are a major inhibiting factor in the proliferation of federated systems. Outsourcing risks arise because the client of an electronic service must first invest to integrate a service, and then surrenders control of many of the factors governing the successful transaction of their business to the service provider.

- *J. Skene is with the Department of Computer Science, The University of Auckland, Building 303, 38 Princes Street, Auckland, New Zealand. E-mail: j.skene@gmail.com.*
- *F. Raimondi is with the School of Engineering and Information Sciences, Middlesex University, The Burroughs, London NW4 4BTT, UK. E-mail: f.raimondi@mdx.ac.uk.*
- *W. Emmerich is with the Department of Computer Science, University College London, Malet Place Engineering Building, London WC1E 6BT, UK. E-mail: w.emmerich@cs.ucl.ac.uk.*

Manuscript received 29 Feb. 2008; revised 24 Sept. 2008; accepted 2 Dec. 2008; published online 17 Aug. 2009.
Recommended for acceptance by R. Schlichting.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2008-02-0091.
Digital Object Identifier no. 10.1109/TSE.2009.55.

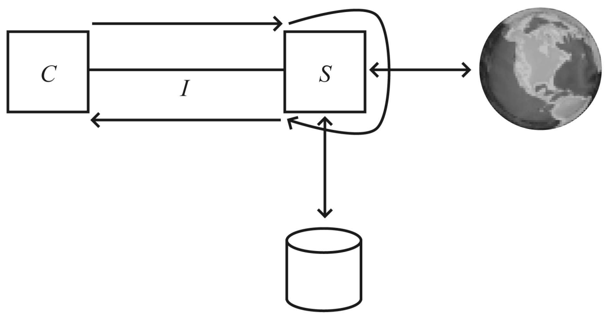


Fig. 1. A three-party electronic-service scenario.

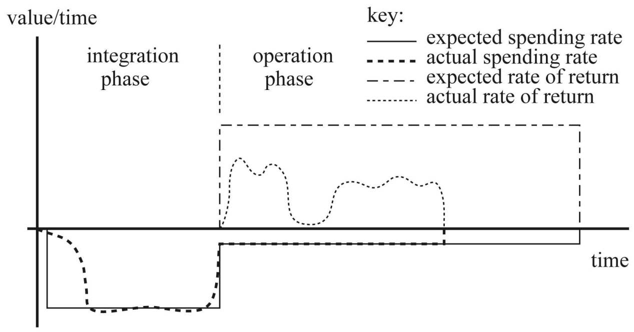


Fig. 2. Value flows in a hypothetical electronic-service relationship.

These risks may be mitigated by Service-Level Agreements (SLAs) that associate financial penalties with adverse outcomes. However, this establishes incentives for the participants to cheat, either by attempting to avoid paying penalties for which they are liable, or by forcing their peer to pay a penalty. Consequently, it is necessary for SLAs to be *precise* and *understandable* so that the parties cannot disagree over the provisions of the agreement, *monitorable* so that a party may not breach the agreement without their peer being aware of it, and *nonexploitable* so that parties cannot be forced to pay unfair penalties. In this paper, we examine how these requirements can be achieved by describing the design and practical evaluation of our language, SLAng.

We also highlight a significant result of our analysis of the monitorability of SLAs in the common scenario in which a service is provided across a network controlled by a financially independent third party. In order to achieve a system of mutually monitorable SLAs without introducing new risks for the participants the network-service provider must act as a reseller of the service. Clearly this result may have an impact on existing models of Internet service provision.

The remainder of this paper is structured as follows: In Section 2, we examine outsourcing risk and consider requirements for systems of SLAs with the principal purpose of mitigating outsourcing risk; in Section 3, we describe our theories of monitorability and approximate monitorability and consider the levels of monitorability possible for SLAs in a basic electronic-service-provisioning scenario; in Section 4, we describe the SLAng language and discuss how its formal specification lends it properties of precision and understandability; in Section 5, we describe the evaluation of SLAng using a case study; in Section 6, we discuss related work; and in Section 7, we conclude and discuss future work.

2 Systems of SLAs for Electronic Services

2.1 Outsourcing Risks

At least three parties are usually involved in the provision of an electronic service. These are the client C, the service provider S, and the network-service provider, in the context of the Internet also known as an Internet-Service Provider (ISP), denoted by I. The scenario is depicted in Fig. 1.

The client, utilizing an appropriate client software agent, submits requests to the service at its discretion or according to a loose schedule. The network, under the supervision of the ISP, conveys these requests to the service, which, under the supervision of the service provider, performs some appropriate processing, possibly performing or instigating some real-world activity as a result, and possibly storing or modifying some data held on behalf of the client. In due course, a response may be returned to the client via the network.

More than one ISP may be involved in the delivery of messages, with ISPs exchanging the messages at the boundaries between their networks. Client programs under the control of a single client organization may also be distributed in the network.

To initiate a service-provisioning relationship, the client first identifies the service that it desires, then arranges the permissions required for the service to be delivered to the point at which it wishes to access it, typically the client's own interface to the Internet. This may mean entering into one or more service-provisioning relationships, possibly governed by formalized agreements.

A client is exposed to two major risks when entering an electronic-service outsourcing relationship: First, the service may not meet some requirements necessary to deliver the value that the client expected to receive as a consequence of using the service. This will result in a cost to the client, either directly or in terms of lost revenue. Second, the client will usually have to make an initial investment to acquire or implement client software capable of using the service, or more generally to integrate the service into its IT infrastructure. If the service ceases to work altogether within the expected period of service provisioning, degrades to the extent that it is no longer cost effective for the client to rely on the service, or if for any reason the service provider prematurely withdraws permission for the client to access the service, then the client will have lost some opportunity to recuperate those costs.

These risks are illustrated in Fig. 2. The graph depicts four flows of cash or value over time, related to a hypothetical service: the client's expected spend, the client's actual spend, the client's expected return, and its actual return. The relationship between the client and the provider can be seen to be divided into two phases, the integration phase and the operation phase. In the former, the client spends to integrate the service, and receives no value from the service. In the latter, the client incurs operating costs as a result of using the service, but has the opportunity to receive value in return.

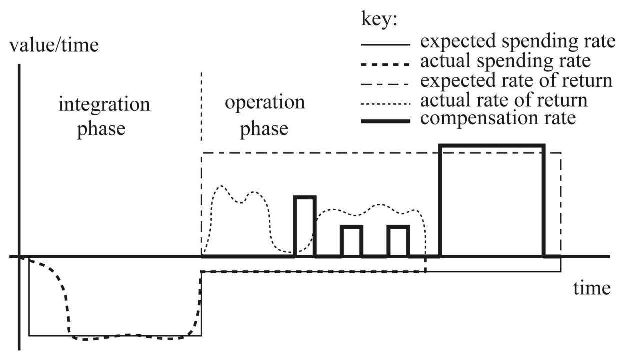


Fig. 3. Value flows in a hypothetical electronic-service relationship with compensation payments governed by an SLA.

The graph depicts a relationship in which the operating period is shorter than expected for some reason, and the client also receives less value than expected during the operating period due to poor service performance. Overall, the service has been rendered unprofitable: The small income achieved is more than offset by the integration costs and operating costs paid. We call these causes of lost income *inefficiency* and *termination* costs.

We argue that outsourcing risks are a significant discouragement for parties wishing to make use of outsourced services, and one reason that service-oriented technology has thus far found its principal applications in structuring the activity within large organizations, such as banks and retailers, where these risks can be mitigated administratively, in trivial services offered at low cost without warranties, and in very high-value relationships where costly risk-mitigation techniques such as due-diligence inspections are feasible.

2.2 SLAs for Electronic Services

In this section, we discuss the requirements for SLAs that can mitigate outsourcing risks, highlighting the most important general requirements, and the conditions needed in the electronic-service scenario. A fuller description of requirements for electronic-service SLAs is given in [15].

The term "Service-Level Agreement" is applied to a range of document types of varying content and import. SLAs in which commitments to deliver various kinds of compensation are related to the behaviors of one or more electronic services and parties may be used to mitigate risk in electronic-service scenarios.

Fig. 3 reprises the example service-provisioning relationship described in the previous section. Now an additional cash flow is depicted representing compensation payments paid by the provider to the client according to the terms of an SLA. Note that compensation is paid in response to poor performance of the service, and in the event of early termination of the relationship, and goes some way to balancing the inefficiency and termination cost incurred by the client.[1]

1. The force of an SLA may be delivered by legal protection, in which case the SLA may form all or part of a contract for outsourcing. However, no part of this paper should be regarded as legal advice. The authors are not legally qualified, and readers should seek the advice of a lawyer before entering any legally binding SLA.

An SLA is a mechanism whereby a party may become obliged to make a payment to another party. The parties to an SLA will therefore have incentives to cheat, either to avoid having to pay a penalty, or to force their peer to pay them a penalty. For SLAs to be effective at mitigating risks, they should be written so as to minimize the opportunities for cheating. If the client cannot rely on compensation in the event of poor service or early termination because the provider may cheat, then outsourcing remains a risky proposition. If the SLA allows the client to extort penalty payments from the provider, then the provider should not enter the agreement, rendering it ineffective for mitigating risk.

To resolve a disagreement concerning an SLA, the parties must first agree an account of all events pertinent to the agreement (for example, concerning the behavior of the service). Then, the implications of the SLA in respect of this account must be recovered from a concrete record of the agreement, and these implications acted upon.

Consequently, we consider the most important requirements for an SLA that applies financial penalties to be: that the SLA should be *monitorable* so that the parties may obtain trustworthy information concerning the pertinent events, *understandable* so that the original intentions of the parties concerning the agreement can be recovered, and *precise* so that these intentions cannot be misinterpreted.

We assume that a client of a service should reasonably only be concerned with the behavior of a service insofar as it affects the client. The internal behavior of the service should be the responsibility and concern of the service provider alone.

Referring to Fig. 1, it is clear that two kinds of behavior of the service may affect the client in the electronic-service scenario. First, the client may receive information from electronic services via the network; and second, the service may affect the outside world in such a way that the consequences are eventually visible to the client.

Communications originating from a service have two main attributes with which the client may be concerned: what is returned and when it arrives. Conditions related to the interval between a service request and the time of arrival of a correlated response are variously referred to as performance, latency, or *timeliness* conditions.

Because the client has no access to the implementation of the service, its expectations concerning the behavior of the service will depend on a description of the service given to them by, or negotiated with, the service provider. If the service subsequently behaves in a manner other than that described, the client is likely to suffer. Hence, the client will wish to ensure that the service either behaves as described to a high degree or the client will be entitled to receive compensation. Such conditions are normally called *reliability* conditions.

Communications via electronic services have no other attributes, so we conclude that the client will be primarily concerned with timeliness and reliability conditions relating to these services, and with conditions relating to the real-world behavior of the service as a whole.

For example, consider the purchase of goods by a merchant from a wholesaler. Let us assume the merchant interacts with the wholesaler's purchase order system via a

Web service, browsing its stock lists and, in due course, submitting an order. This interaction will consist of a sequence of responses and requests transacted entirely through the medium of the network. The submission of orders is no doubt a matter of urgency for the merchant—the ordering process contributes to the time taken to replenish their stock, and the merchant has better things to be doing than interacting with a slow computer system. Therefore, an SLA could be established to constrain the timeliness of responses from the service. Reliability will clearly also be a concern, so that the merchant can be confident that their orders are being processed. Interactions will also result in activity on the wholesaler's part to fulfill orders. Stock will be retrieved from a warehouse, or ordered from a third party, and will be packaged and dispatched via a hauling service. This latter type of behavior does not require interaction with the client over a network, but still ultimately affects the client when the stock is delivered, or fails to arrive when expected.

Reliability and timeliness conditions applied to a service provider in an electronic-service SLA force the provider to provide a service, or else pay a penalty. The provider will therefore require the inclusion of a charging scheme in the SLA, to compensate it for the cost of providing the service. This is one example of a condition implied by the requirement for nonexploitability.

The provider also assumes a risk due to the finite capacity of such services. Characteristically, the timeliness of an electronic-service will decrease drastically once some critical resource, such as a processor or database, approaches 100 percent utilization [16]. Since, in an electronic service, the ability of the provider to improve the capacity of the critical resource at runtime will be limited, the only way to control the degree of resource contention and hence the overall time spent waiting for resources is to limit the rate of requests. However, the rate of requests is controlled by the client. Therefore, it is possible for a client to attempt to exploit an electronic-service SLA by increasing the rate of requests. This will result in a reduction in timeliness. Also, due to the necessarily finite capacity of queues in the implementations of electronic services, if the volume of incoming requests remains high, it will eventually become necessary to begin ignoring requests, impacting service reliability.

This risk to the provider can be mitigated in an SLA by applying a condition to the client that requires a limit on the rate of service requests; a *throughput* condition.

As discussed in Section 2.1, termination risks are a major risk for clients of outsourced services. Equally, service providers may want to safeguard their revenue streams, so either party may wish to include penalties for early termination in an electronic-service SLA.

In this section, we have described requirements for monitorability, understandability, and precision, and argued that electronic-service SLAs need conditions relating to timeliness, reliability, throughput, payment, and termination. Two considerations complicate the task of designing language support for SLAs starting from these requirements. First, electronic services may be delivered to the client over one or more networks controlled by ISPs. These providers may be independent of the electronic-service provider, but it is clear that the behavior of the networks has the capability to introduce delays and faults into the communications between the client and the electronic services constituting the electronic service. This is precisely the risk that the client is seeking to mitigate through the use of SLAs. Therefore, we must also consider the relationship of electronic-service SLAs to guarantees that may be offered concerning the behavior of the network.

Here, we note that any given electronic-service scenario may require not merely one SLA, but a system of SLAs, in order to mitigate outsourcing risk without introducing unacceptable new risks. We consider how this can be achieved in the next section.

The second issue is that, although the types of conditions that we expect to find in an electronic-service SLA are limited in number, the ways in which these conditions will be formulated are extremely various. The conditions may need to vary in response to external conditions relating to the business being conducted, and this must be captured in the SLA. Conditions related to the real-world impact of services may also need to be expressed. These requirements for general expressivity conflict with a desire for a precise, fully defined language with a finite number of constructs. We discuss our approach to this problem in Section 4.

3 The Monitorability of SLAs

3.1 Overview

Three parties participate in the basic electronic-service scenario introduced in Section 2.1: the client, the service provider, and the network-service provider. Discounting the behavior of the service outside the network, and assuming that the interface to the service is a simple, synchronous electronic service, let us consider what could go wrong for the client in this situation.

One possibility is that, having submitted a request, no response is received by the client within some reasonable interval of time. The client complains to the service provider that a timely response was not received. The provider claims that no request was received, produces a log of requests as evidence supporting this claim, and directs the client to complain to the ISP who was responsible for conveying the request to the service. The ISP insists that the request reached the service provider and produces a log supporting this claim. Who can the client trust? Both the ISP and electronic-service provider have delivered easily fabricated evidence concerning an event, the delivery of the request at the service-provider's interface, that the client was incapable of independently monitoring.

Let us assume that, for its own reasons, the client chooses to mistrust the service provider, and requests that it enter into an SLA. In this agreement, the client seeks to reduce the costs that it expects to incur when the service fails to perform as expected by receiving a penalty from the provider, also giving the provider a disincentive to poor performance. The client perceives that the problem with the service is a lack of availability due to an erratic maintenance regime on the part of the provider. The provider duly commits to provide 95 percent availability over the lifetime of the contract of which the SLA forms a part.

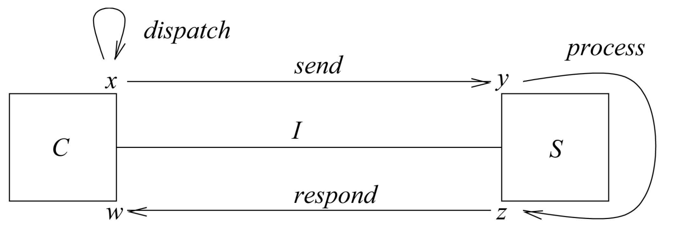


Fig. 4. An interaction model for electronic-service provision showing actions and their associated events.

Over the period of the contract, the client uses the service frequently, and frequently responses are not generated following requests. At the termination of the agreement, the client seeks compensation from the provider, which refuses to pay. The provider argues that, although the service was unavailable when requests were made for which responses were not received, at all other times the service was available. Accumulating the fleeting intervals during which the failed requests were being delivered and the service was admittedly unavailable does not amount to 5 percent of the lifetime of the contract, and hence the provider need not pay.

These examples highlight monitorability as an important requirement for SLAs. In both cases, the client became concerned with an event that it fundamentally could not observe: In the first example, the delivery of the request to the service; in the second, the transition of the service between availability states. In both cases, the concern arose because another event that they could observe, the delivery of the response to the client's interface, failed to occur when expected. Had the client complained about this latter event, they would have had a stronger argument because no party could convince it of a falsehood concerning that event.

In the event of a dispute between the parties to an SLA, it will be necessary for the parties to collect and present convincing and pertinent evidence in order to determine how the intent of the agreement should be applied. Depending on how an SLA is written, such evidence may be easier or harder to obtain. We refer to the property of an SLA that determines how easily relevant and trustworthy evidence may be obtained as its *monitorability*.

In this section, we summarize a technique for analyzing systems of SLAs to determine the degree of monitorability possible, first presented in [17]. We also discuss a significant result of this analysis applied to the electronic-service scenario, and the handling of measurement error in SLAs.

3.2 Modeling the Three-Party Scenario

Fig. 4 depicts the interaction model for the electronic-service scenario. Interactions with the real world and any database have been elided, and a simple synchronous model of communications has been assumed. Four events are indicated, $E = \{x, y, z, w\}$, each corresponding to the completion of an action occurring during a service request, respectively, $dispatch$, $send$, $process$, and $respond$. In [17], we used this model as a starting point for a formalization of SLAs in the service-provision scenario in order to consider their monitorability.

We considered the client's requirement for timely delivery of results. The client is concerned with the amount of time that elapses between a request being dispatched into the network and a response being completely received. This can be expressed as $(C, w - x < t)$. Either of the other parties in the scenario could offer C an SLA insuring this requirement: the service provider could offer $(S, (C, w - x < t))$ or the network-service provider could offer $(I, (C, w - x < t))$. However, this requirement could also potentially be met by constraining the relative times of events occurring between w and x, the intuition being that the overall time taken to complete a request is acceptable if the times taken to complete each action required to service the request are also acceptable. For example, $w - x < t$ will hold if $y - x < t_1$, $z - y < t_2$, and $w - z < t_3$, in all cases where $t_1 + t_2 + t_3 < t$.

Even supposing that the delay between events x and y exceeds the arbitrary bound t_1, this does not imply that the client's overall requirement will be violated. $w - x < t$ will be met if $z - x < t_1 + t_2$ and $w - z < t_3$, and other combinations of constraints are also possible. The total set of observations with which we are concerned is hence

$$O = \{y - x < t_1, z - y < t_2, w - z < t_3, z - x < t_1 + t_2, \\ w - y < t_2 + t_3, w - x < t\},$$

where $t_1 + t_2 + t_3 < t$ and t, t_1, t_2, and t_3 are all positive and nonzero.

Given that SLAs may be made insuring any of these observations, we considered what systems of SLAs are possible in the scenario. In principle, any party may insure any observation for another party. The set of parties is $P = \{C, S, I\}$. The maximum number of possible SLAs in a given scenario is $|O| \times |P| \times (|P| - 1)$. The number of combinations of SLAs is, therefore, $2^{|O|\times|P|\times(|P|-1)}$. In this case, $2^{3\times2\times6} = 2^{36} \sim 6.9 \times 10^{10}$ distinct systems of SLAs are possible.

Based on our model, we formalized criteria for good systems of SLAs. The various events in the scenario are only intrinsically visible to certain participants. The event x, for example, is visible to C and I. However, S could find out about x if it trusted C or I and that party reported to S. We made the conservative rule that a party can only report an event if they were not involved in an SLA related to the event; otherwise, they would have an incentive to cheat. A party can only *monitor* an event if it is visible to that party, or a party they trust can report it. Clearly, an SLA is *monitorable* to a party if it can monitor all events relevant to the SLA. An SLA is *mutually monitorable* if it is monitorable to both client and provider. An SLA is *arbitratable* if it is monitorable by a third party trusted by both client and server and which is trusted to report.

Some additional conditions are required to find useful systems of SLAs. Some systems do not insure the client's requirement. We call systems that do *adequate*.

The system consisting of a single SLA $\{(S, (C, w - x < t))\}$ is adequate, but it is not *safe*. The service provider insures the performance of the service at the client's interface to the network. If an error occurs in the network, the service provider must pay the client, but the fault will be the network-service provider's. The service provider can only intrinsically *guarantee* the observation $z - y < t_2$. The network-service provider can guarantee $y - x < t_1$ and $w - z < t_3$. An SLA is safe if the provider of the SLA either guarantees the requirement being insured, or is the client of

TABLE 1
Results of a Monitorability Analysis for the Electronic-Service Scenario

Satisfactory	Safe	Non-redundant	Non-reciprocal	Non-client	Mutually monitorable	Arbitratable	Solutions
–	–	–	–	–	–	–	$\sim 6.9 \times 10^{10}$
✓	–	–	–	–	–	–	$\sim 6.6 \times 10^{10}$
–	–	–	✓	–	–	–	$\sim 3.9 \times 10^{8}$
–	–	–	–	✓	–	–	$\sim 1.7 \times 10^{7}$
✓	✓	✓	–	–	–	–	281
✓	✓	✓	✓	–	–	–	122
✓	✓	✓	✓	–	✓	–	1
✓	✓	✓	✓	✓	–	–	34
✓	✓	✓	✓	✓	✓	–	1
✓	✓	✓	✓	✓	✓	✓	0

SLAs insuring any dependent observations required to make up the requirement. For example, S could safely offer $(S, (I, z - x < t_1 + t_2))$ if it received $(C, (S, y - x))$. Of course, this second SLA is not necessarily safe for C, which cannot intrinsically guarantee any observation.

Certain other systems of SLAs should be ruled out. Systems in which SLAs for the same requirement are offered by parties reciprocally can produce a false illusion of safety. Systems in which the client unnecessarily receives SLAs for their requirement and a subrequirement, e.g., $w - x < t$ and $y - x < t_1$ are fine in principle, but are unlikely to occur in the real world. We may also wish to rule out the possibility of the client offering SLAs.

These rules are formalized in [17], which also describes a depth-first algorithm for discovering systems of SLAs with particular characteristics, such as safety and monitorability, and analytical approaches to determining the number of satisfactory, nonreciprocal, and nonclient systems. A summary of the analytical results and the results of the search algorithm for the scenario is shown in Table 1.

The most significant result of this analysis is that in only one possible system are all SLAs mutually monitorable. This is true whether or not we permit the client to offer SLAs.

In this scenario, for a system of SLAs to be safe and satisfactory, both S and I must issue SLAs supported by guarantees contributing to C's requirements. Hence, all parties will be financially involved in every contractual situation, and no party can be trusted to report any events that occur remotely from another party. Hence, only SLAs between adjacent parties can be mutually monitored, namely, contracts between C and I, and I and S. Only one scenario meets this requirement. It consists of the contracts $(I, (C, w - x < t))$ and $(S, (I, z - y < t_2))$. The ISP vouches that the service will perform correctly across its interface with the client. It is capable of guaranteeing that the request reaches the server in a timely fashion, and that any response makes it back in time. To fully guarantee the round-trip time of the service, the ISP must only obtain a guarantee from the service provider that the service will complete in good time.

That no arrangement can be arbitrated is obvious without applying the search algorithm. Because all parties in the scenario must be involved in contracts to satisfy C's requirement, no financially independent third party can be present to observe any interaction.

That the scenario is only mutually monitorable by all parties in one system of SLAs is a highly significant result with two important consequences: First, service constraints will be required at both the interface to the client and the interface to the service, but these guarantees will be of the same form, related to the exchange of meaningful messages, rather than the movement of data in the network. Therefore, to achieve mutually monitorable end-to-end QoS guarantees for electronic services, only one type of SLA language need be used. There is no need for a different vocabulary to describe network QoS.

The second consequence is of perhaps greater practical importance. This system of SLAs requires the ISP to offer guarantees on the received quality of an electronic service at the interface to the client, effectively forcing the ISP to act as a reseller of electronic services, a business model seldom adhered to in practice today.

This result may be generalized to scenarios including multiple ISPs and clients distributed in the network. Clearly a sequence of SLAs can be established between each client and the service, such that each SLA is made between two adjacent parties, one serving as the client and the other as the service provider. This establishes that, when multiple ISPs are involved in delivering the service, then systems of SLAs exist that are at least mutually monitorable.

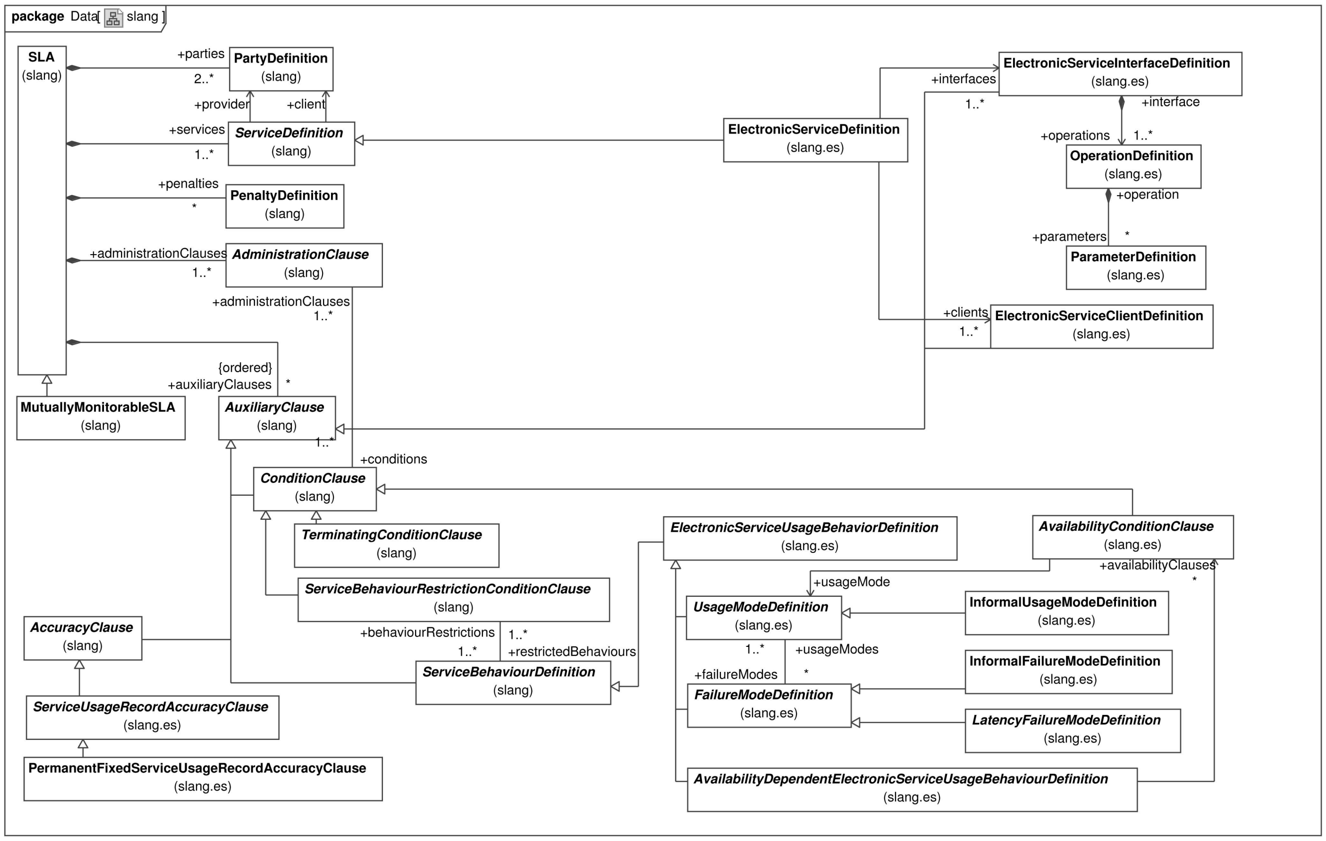
3.3 Approximate Monitorability

The calculation of penalties in relation to an SLA must be performed using an account of service behavior which has been collected by measuring the service, and will therefore be subject to measurement error. Error processes are intrinsically unmonitorable—otherwise, the errors could be corrected. During the administration of an SLA, a malicious party could present falsified evidence and claim that it had been honestly gathered, inaccuracies being due to error. What is required in a good SLA is a constraint on the accuracy of information used during administration, conformance to which may be checked by other parties (with access to their own, trustworthy accounts of system behavior) using statistical techniques, in order to gain a bounded confidence that a party is acting honestly. We refer to such a condition as *approximately monitorable*, and showed in [17] how this can be achieved. Approximately monitorable accuracy constraints are included in our language, SLAng, which is described in the next section.

4 Precise Domain-Specific SLA Languages

4.1 Overview

In [18], we describe how a combination of metamodeling languages, standardized under the Object-Management Group's (OMG) Model Driven Architecture (MDA) initiative [19], can be used, in combination with a modeling



package Data[slang]
SLA (slang)
PartyDefinition (slang)
ServiceDefinition (slang)
PenaltyDefinition (slang)
AdministrationClause (slang)
MutuallyMonitorableSLA (slang)
AuxiliaryClause (slang)
ConditionClause (slang)
TerminatingConditionClause (slang)
ServiceBehaviourRestrictionConditionClause (slang)
ServiceBehaviourDefinition (slang)
AccuracyClause (slang)
ServiceUsageRecordAccuracyClause (slang.es)
PermanentFixedServiceUsageRecordAccuracyClause (slang.es)
ElectronicServiceDefinition (slang.es)
ElectronicServiceInterfaceDefinition (slang.es)
OperationDefinition (slang.es)
ParameterDefinition (slang.es)
ElectronicServiceClientDefinition (slang.es)
ElectronicServiceUsageBehaviorDefinition (slang.es)
AvailabilityConditionClause (slang.es)
UsageModeDefinition (slang.es)
InformalUsageModeDefinition (slang.es)
InformalFailureModeDefinition (slang.es)
FailureModeDefinition (slang.es)
LatencyFailureModeDefinition (slang.es)
AvailabilityDependentElectronicServiceUsageBehaviourDefinition (slang.es)
+parties 2..*
+provider +client
+services 1..*
+penalties *
+administrationClauses 1..*
+administrationClauses 1..*
{ordered} +auxiliaryClauses *
1..*
+conditions
+behaviourRestrictions 1..*
1..* +restrictedBehaviours
+interfaces 1..*
+interface
+operations 1..*
+operation
+parameters *
+clients 1..*
+usageMode
+availabilityClauses *
1..* +usageModes
+failureModes *

Fig. 5. Principal classes and relationships in the abstract syntax of SLAng.

idiom that is now commonly called “model-denotational semantics” [20], to define an SLA language, SLAng, that is both precise and understandable. Subsequent work has refined the definition of SLAng [21], [15]. We now describe the advantages of our approach in light of the most recent version of the language.

SLAng has the following key features:

- The SLAng language specification is written in the Essential Meta-Object Facility (EMOF) model language [22], a class-based formalism similar to Unified Modeling Language (UML) class diagrams [23], and also the Object-Constraint Language (OCL) [24], a declarative language offering constructs similar to predicate logic that can be used to define class invariants.
- The specification consists of class definitions, and may be divided into an “abstract syntax” describing the structure of SLAs, and a domain model, describing the domain entities and events occurring in an electronic-service scenario.
- The abstract syntax and domain model are related using associations. Class invariants written in OCL establish the semantics of syntactical elements in terms of the restrictions they imply over the domain elements with which they are associated. Associating an SLA with a service scenario obliges the participants in the scenario to respect the agreement by avoiding behaviors that would violate implied constraints.
- Many classes in the abstract syntax of SLAng are themselves abstract. In order to specify an SLA, it is usually necessary to first extend the language. This reflects the fact that most SLAs must capture business-related rules which cannot be anticipated in the domain of the language. However, the abstract syntax provides guidance and support for extenders in the form of both abstract side-effect-free OCL operations which must be overridden to make certain conditions concrete, and concrete side-effect-free operations which can be reused in constraints. In this respect the language definition is similar to an object-oriented programming framework.

Fig. 5 shows the main classes in the abstract syntax of SLAng. An SLA consists of a number of definitions and clauses. Definitions identify the parties, services, and penalties that are relevant to the service-provision scenario. Clauses establish constraints over these domain entities, concerning, primarily, how the SLA is to be *administered* (the process whereby the parties calculate violations and then subsequently pay penalties), and subordinately, what conditions regarding service behavior will be considered during administration. SLAng currently includes specializations of these syntactic types for electronic services. These include syntactic types for electronic-service interfaces and service behaviors relating to patterns of service usages (exchanges of requests and responses), which may include types of failures, overdue responses, or the delivery of requests. The occurrence of undesirable behaviors may be

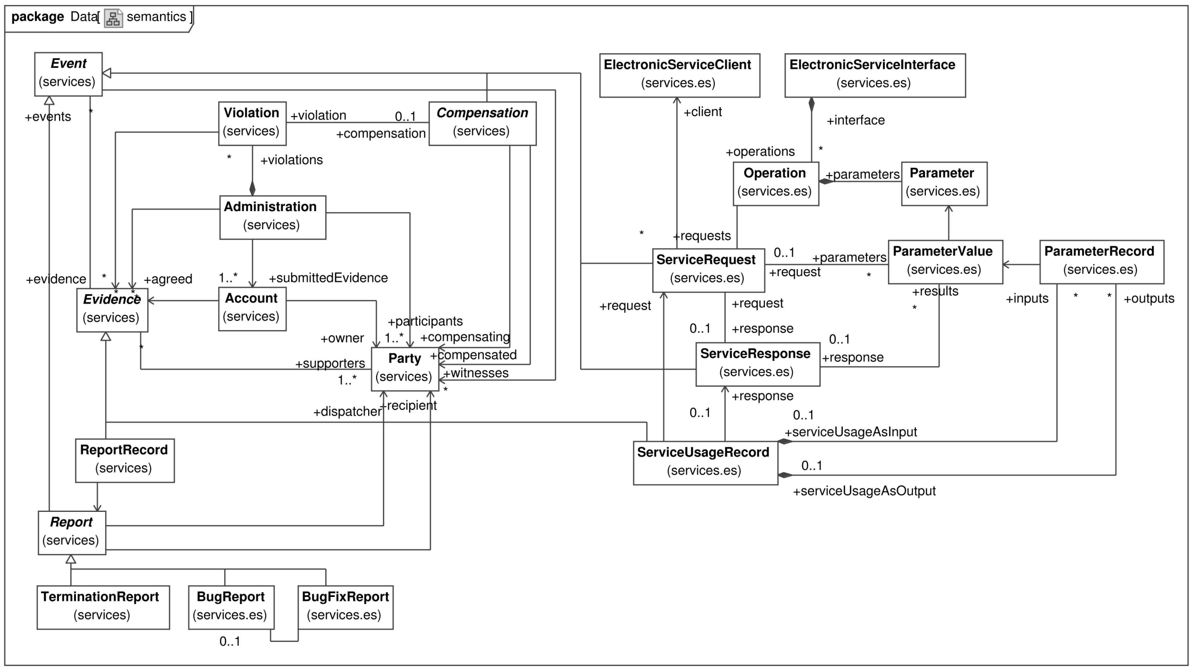


Fig. 6. Semantic model of service provision for SLAng.

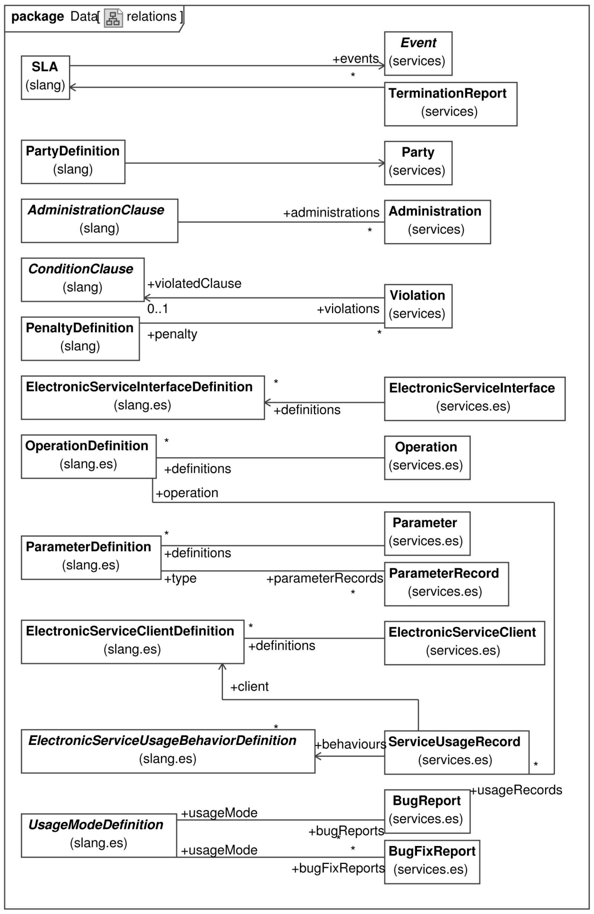
associated with an obligation to pay a penalty using service-behavior restriction condition clauses, resulting in reliability, latency, and throughput conditions. We also include availability constraints, which are defined in relation to exchanges of bug and bug-fix reports. These events are mutually monitorable, in contrast to changes in the underlying state of the actual service, which are certainly unmonitorable by the client.

Fig. 6 shows the domain model for SLAng. This is divided into two main parts: a generic model of SLA administration, and a model of electronic-service provisioning. The latter includes classes representing infrastructure elements, such as electronic-service interfaces, the operations that make up these interfaces, and the client software capable of accessing the service. It also includes events relating to service provisioning, such as service requests and responses, corresponding to the exchange of messages across some mutually monitorable interface. The model of SLA administration models the way that penalties are calculated based on accounts of service behavior submitted during administration. The semantic model hence contains a number of classes representing evidence concerning service behavior, such as report records, which document communications between the parties outside the normal operation of the service (e.g., bug reports, or notices to quit the agreement), and also records of service usage. Accuracy clauses in SLAs associated with these administration events constrain the precision with which the corresponding events must be recorded in this evidence, using the approximately monitorable accuracy constraint described in [17].

Fig. 7 shows the associations between syntactic and semantic elements in the language specification. Multiplicity constraints over these associations establish denotation relationships. For example, the association between the classes PartyDefinition and Party establishes that in a valid scenario there is at least one party corresponding to each definition. More sophisticated constraints are conveyed using invariants expressed using OCL, and these bear much of the burden of establishing the semantics of the language. In the next section, we give examples.

The principal benefits of following a model-denotational approach to defining a DSL are precision, understandability, and the ability to validate the specification using testing. When the domain of the language has a different logical structure to the syntax of the language (as is the case with SLAs, where the domain concerns service infrastructure and events, whereas the syntax expresses conditions relating to these events), the domain can be documented concisely in terms of its natural structure. The relationship of the language to the domain is then defined formally. This is superior to defining the meaning of the language directly using natural-language statements about the domain attached to syntactic elements, when the relationship between the syntactic elements and the domain is complex and difficult to describe. We showed in [25] how a model-denotational language specification may be compiled into a metadata repository in order to test, using examples and counterexamples, whether the relationship between the syntax and domain has been correctly captured.

Understandability, at least for the technical community familiar with object-oriented formalisms, is delivered through the use of the standard languages EMOF and OCL. The understandability of a particular SLA still relies on maintaining traceability from the concrete document of the agreement, through the specification of the concrete syntax in which the document is written, to the formal specification of the abstract syntax, to the related semantic elements, and finally, the natural language descriptions of these elements. In [21], we describe enhancements to the underlying standards used to describe SLAng to preserve this traceability. These recommendations are implemented in our own open-source project, the UCL UML tools, as extensions to implementations of the EMOF, XMI, and Human Usable Textual Notation (HUTN) standards. One consequence of



package Data[relations]
SLA (slang)
Event (services)
+events *
TerminationReport (services)
PartyDefinition (slang)
Party (services)
AdministrationClause (slang)
+administrations *
Administration (services)
ConditionClause (slang)
+violatedClause 0..1
+violations *
Violation (services)
PenaltyDefinition (slang)
+penalty
ElectronicServiceInterfaceDefinition (slang.es)
+definitions *
ElectronicServiceInterface (services.es)
OperationDefinition (slang.es)
+definitions *
Operation (services.es)
+operation
ParameterDefinition (slang.es)
+definitions *
Parameter (services.es)
+type
+parameterRecords *
ParameterRecord (services.es)
ElectronicServiceClientDefinition (slang.es)
+definitions *
ElectronicServiceClient (services.es)
+client
ElectronicServiceUsageBehaviorDefinition (slang.es)
+behaviours *
ServiceUsageRecord (services.es)
+usageRecords *
UsageModeDefinition (slang.es)
+usageMode
+bugReports *
BugReport (services.es)
+usageMode
+bugFixReports *
BugFixReport (services.es)

Fig. 7. Relationships between the semantic and syntactic models.

these recommendations is that all definitive natural language statements concerning the meaning of the domain model are compiled into any formal representation of the language specification. It is, therefore, possible to generate a document of the language in several formats. A complete description of the version of SLAng discussed in this paper is available online in HTML format [26], and appears as a LaTeX formatted document in an appendix of [15].

4.2 SLAng Examples

In this section, we give some examples from the SLAng specification. These are taken from the formal definition of SLAng, which uses a text-based representation of an EMOF model into which OCL expressions are embedded. We also give some examples of concrete SLAng statements, expressed using the HUTN, a generic concrete syntax standard. Both types of statement have been parsed and type checked by the UCL MDA tools [21].

The following invariant is arguably the most important in SLAng, as it establishes the responsibility of the parties to calculate violations according to the conditions included in the SLA. It is defined on the AdministrationClause class.

```
invariant {
  administrations->forall(a : ::services::Administration |
    conditions->forall(violationsCalculated(a)))
}
```

The operation violationsCalculated() is abstract, and defined on the type ConditionClause. It is overridden by specific types of condition clause. For example, in an extension required by the case study described in Section 5, we defined a number of types extending from PermanentFixedWindowFixedOccurrencesMaximalServiceBehaviourRestrictionConditionClause. These kinds of clauses associate a penalty with a behavior that occurs too frequently in a sliding window. The definitions of two properties and three operations of this class, including an overridden violations Calculated(), are given below:

```
maxOccurrences : ::types::Integer

window : ::types::Duration

violationsCalculated(administration :
  ::services::Administration) :
  ::types::Boolean {
  let first = firstMaximalViolation(administration.agreed,
    administration)
  in
    first->size() = 0
    or
    (violationExistsFor(first, administration)
      and
      allLaterViolationsCalculated(first, administration))
}

violationExistsFor(maximal : ::services::Evidence[0, *]
  unique,
  administration : ::services::Administration) :
  ::types::Boolean {
  administration.violations->exists(v : ::services::Violation |
    v.evidence = maximal and
    v.violator = service().client and
    v.penalty = getPenaltyForMaximalViolation(maximal)
  )
}
```

This definition of violationsCalculated() depends on an operation firstMaximalViolation(), the definition of which is too extensive to be reproduced here, but which calculates a subsequence of the events administration.agreed—the account of service behavior that the parties to SLA have agreed will be the basis for penalty calculations. The operation firstMaximalViolation() depends on the values specified for the properties maxOccurrences and window defined in this class. The operation violationExistsFor() checks that a violation has been calculated for a particular sequence of evidence corresponding to a violation. It depends on the operation calculatePenaltyForMaximalViolation(), which is abstract. In a subsequent concrete class PermanentFixedWindowFixedOccurrencesSteppedPenaltyMaximalServiceBehaviourRestrictionConditionClause this operation is overridden to calculate a penalty which varies according to the length of the violation.

We used the permanent, fixed-window, fixed-occurrence, stepped-penalty, maximal, service-behavior restriction condition clause type to implement a latency condition on a service provider in SLA 1, an SLA developed as part of the case study described in Section 5. The clause is stated using the HUTN as follows:

```
::sla1::slang::PermanentFixedWindowFixedOccurrences
  SteppedPenaltyMaximalServiceBehaviourRestriction—
  ConditionClause[nuisance]() {

  maxOccurrences = 10;

  window = ::types::Duration(10, min)

  restrictedBehaviours = using ::sla1::slang::es
    { FixedLatencyAvailabilityDependentViolation
    DependentFailureModeDefinition[nuisanceDelay] }

  penalties = using ::sla1::slang {

    SteppedPenalty() {
      threshold = ::types::Duration(1, day)
      penalty = ::combined::slang::FixedDeadline
      FixedPoundsSterlingPaymentPenaltyDefinition[day1]
    },

    SteppedPenalty() {
      threshold = ::types::Duration(2, day)
      penalty = ::combined::slang::FixedDeadlineFixed
      PoundsSterlingPaymentPenaltyDefinition[day2]
    }
   }
}
```

The behavior associated with this constraint is defined as follows:

```
::sla1::slang::es::FixedLatencyAvailabilityDependent
  ViolationDependentFailureModeDefinition
  [nuisanceDelay]
("An annoying delay accessing a page") {

  maxDuration = ::types::Duration(10, S)

  operations = {
    es::OperationDefinition[static1],
    ...
    es::OperationDefinition[results8]
  }

  availabilityClauses =

    { ::sla1::slang::es::PermanentSteppedPenaltyFixed
      DeadlineAvailabilityConditionClause[general] }

  satisfyingConditions =
    { ::combined::slang::PermanentFixedWindowFixed
      OccurrencesNoPenaltyMaximalServiceBehaviour
      RestrictionConditionClause[throughput] }

  usageModes = { es::InformalUsageModeDefinition
  [anyUsage] }
}
```

These two SLA clauses define a latency condition that applies throughout the duration of the agreement, for any usage of any operation, except when either throughput or availability conditions (defined elsewhere) are violated, and hence the latency condition becomes irrelevant. A request is considered to be overdue if it takes longer than 10 seconds to return. Penalties apply in any interval where more than 10 overdue requests are observed in a window of 10 minutes. If this behavior is observed to occur for more than one day, a higher penalty applies. This is an example of the kind of complex condition that can be formalized, and then stated declaratively, following our approach to defining SLAng.

5 An Evaluation of the SLAng Language for Electronic-Service SLAs

In [15], we describe a thorough evaluation of the current version of the SLAng language. We wanted to discover whether it was practical to use the language to define SLAs in relation to a real service, whether the SLAs produced met our requirements, and whether the language as a whole appeared to be useful, such that using it offered a clear advantage over other approaches, such as either natural language or a language proposed in previous research. We addressed these questions using several approaches: First, we applied the language to defining SLAs in a case study concerning a real service; second, we evaluated the resulting SLAs against our requirements; third, we employed a set of metrics to measure the relative sizes of the parts of the language specification relied on by the SLAs, the extensions required, and the SLAs themselves, to demonstrate that the SLA language was contributing a large part of the meaning of the SLAs; finally, we compared SLAng to related work in terms of the criteria defined by our requirements. This latter exercise is summarized in the next section. Here, we describe the case study.

The eMaterials project, now complete, funded a collaboration between UCL grid-computing researchers in the Department of Computer Science (CS) and the UCL Chemistry Department to investigate the computational prediction of organic crystal structures from chemical diagrams. This problem is relevant to the discovery of new drugs, but is computationally demanding in general. A large number of molecular packings must be considered for each compound, the thermodynamic likelihood of each being indicated by a calculation of the lattice energy. Physical properties of likely crystals must then be estimated. Computation involves search in a large space, coupled with sophisticated analysis of the candidates.

Prior to the eMaterials project, the chemists would execute this search using two Fortran programs, `MOLPAK` and `DMAREL` and a combination of manual and batch control, on a 4 CPU Silicon Graphics server. A typical search would take between one and four months to complete [27].

Within UCL, the Information Services (IS) division is a support group that manages computational and network resources for the administrative departments and the student population. It also administers the interdepartmental network. Individual departments may also have independent groups fulfilling the same role for the academic staff, and this is the case for the Chemistry and Computer Science Departments.

The eMaterials project funded the creation and administration of a computational grid, controlled by researchers in the CS, but consisting of nodes maintained by IS. The aim was to support the analysis activities of the chemists while providing the opportunity to research grid engineering for computer science.

This scenario represents a potentially interesting use of SLAs. The project is now complete and, without a centralized source of funding, the various participants must consider how their costs are to be covered if the simulation infrastructure is to continue to be used. For the purpose of this case study, we adopted the assumption that the parties remain financially independent and that the principle benefit of the infrastructure is to the chemists. They must, therefore, pay for it from funding into minerals research. The various service providers must recuperate their costs by charging for their services. In return, their clients may expect them to provide quality-of-service guarantees.

The case study proceeded as follows:

1. An understanding of the state of development of the scenarios was obtained. This included an understanding of who the stakeholders in the scenario are and what requirements the service-provisioning scenario is intended to meet for them.
2. On the basis of the understanding of the requirements developed in the first step, requirements specifically relevant to SLAs were considered; this involved an analysis of the risks to which the parties were exposed in the (unmodified) scenario—we achieved this objective by examining the system use cases and considering, for each step, what adverse outcomes could occur for each party.
3. A plan for the introduction of SLAs was made, aiming to avoid modifying the existing scenario significantly, which would imply redevelopment costs; this involved developing an initial system of SLAs for the scenario.
4. The SLAs developed in stage 3 were evaluated in terms of the additional advantages they provided with respect to the scenario requirements and any associated costs or disadvantages that introducing the new technology might imply. Consideration was then given as to whether a significant benefit was delivered compared to the cost of introducing SLAs, and whether this benefit could be increased in a cost-effective way by reengineering the original service to better meet derived requirements.

Fig. 8 shows the deployment of the eMaterials service. Dashed rectangles in the figure indicate networks. Arcs between the various components represent communications that pass through these networks. CS serves a number of webpages, collectively known as the `Polymorph Search Webclient`, allowing the chemists to configure and initiate computational simulations. This traffic passes between Chemistry and CS nodes via networks administered by Chemistry, IS, and CS. Following the initiation of a simulation, a BPEL workflow is created within a workflow engine on a CS node [28]. This coordinates the submission of

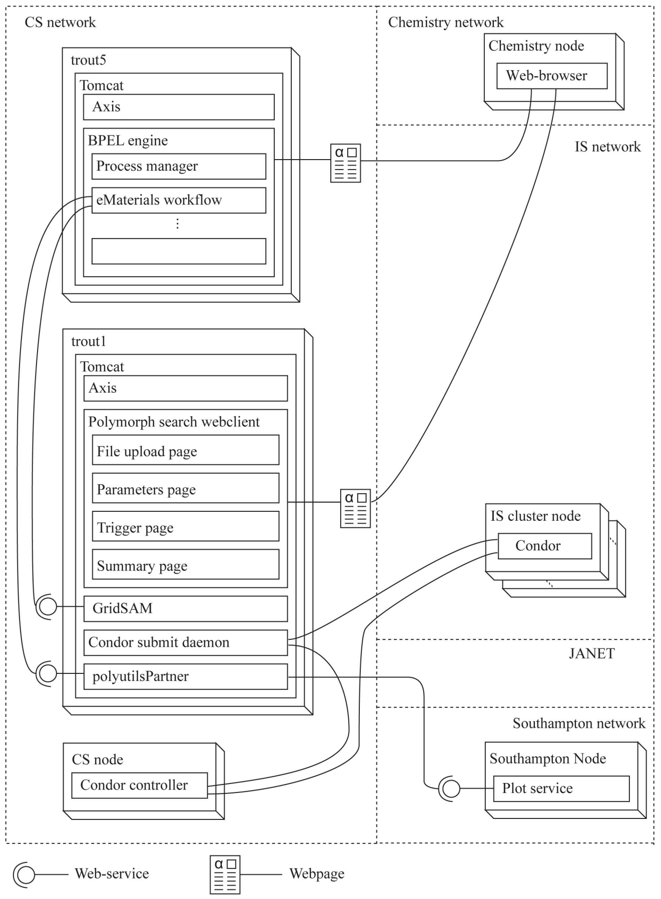


Fig. 8. Deployment of the eMaterials service.

the jobs constituting the simulation to a `Condor submit daemon` [29], wrapped by the `GridSAM` Web service [30]. The daemon maintains a simple queue of jobs and coordinates with a `Condor controller` on another node to find free nodes in the condor cluster and allocate jobs. The cluster nodes are provided by IS and reside within the network maintained by IS. These nodes maintain periodic communication with the `Condor controller` to indicate their status, including information about their current processing load due to other use (the nodes are computers in labs made available to students and staff for general-purpose computing). The nodes will also notify the `Condor submit daemon` to indicate the completion of jobs. Condor nodes communicate using an application-specific presentation-layer protocol over TCP. The workflow instance will periodically poll the submit daemon via `GridSAM` to determine whether jobs have completed, and possibly schedule dependent jobs. The completion of certain jobs indicates the availability of partial analysis results, which are made available on the `Polymorph Search Webclient` incrementally. The results are formatted for presentation on the Web, which includes producing a scatter graph in PNG format, by invoking the `polyutilsPartner` Web service, which resides on the same node as the queue. The `polyutilsPartner` web service invokes the `wsplot` Web service hosted by Southampton University to produce the required graph. The requests and responses from this service pass across the CS, JANET, and Southampton networks.

The principal use case of the service is the configuration and initiation of a simulation, although other use cases also exist, such as the configuration of cluster nodes by CS administrators, which we have not considered in the design of our SLAs. For each of the steps described above, we considered, for each party, what adverse events could occur due to the actions of the other parties, or the services that they provide.

The risks to chemists stem from the possibility that the simulation might not complete, might deliver incorrect results, or that conducting a simulation will be hindered by usability issues affecting the `Polymorph Search Webclient`. The Chemistry Department also assumes some additional risks as a result of the need to interact with other parties. These include security risks due to the need to accept into their own network traffic appearing to originate from within the CS network. Also, if Chemistry regards the results of its simulations to have any proprietary value, then it assumes a risk related to the possibility that simulation data will be stolen when transmitted across the Internet. Finally, Chemistry, by depending on a service provided by one or more second parties, assumes a termination risk, based on the possibility that those parties may choose to render the service permanently unavailable at some point, resulting in reintegration costs for Chemistry.

The Information Services division of UCL initially assumes risks based on its interaction with other parties. These include the security risks of interacting with Chemistry and CS. IS also suffers a risk associated with allowing CS to install and execute software on its computational nodes. IS provides both network and cluster-node services and thus assumes two risks due to the potential volume of legitimate service requests. IS is also exposed to the risk, when providing these services, that it will not be reimbursed for the costs involved.

CS similarly assumes security risks, risks related to resource exhaustion by legitimate service requests, and the risk that it will not be compensated for the resources it contributes to the performance of the simulation.

The ISP, in this case JANET [31], assumes security risks and resource exhaustion risks implicit in permitting interactions between CS and Southampton. The ISP must also find a way to charge for the use of its resources.

Southampton also assumes security and resource exhaustion risks providing the `plotws` Web-service. Southampton also wishes to recuperate its costs from providing the service, by our assumption.

We next considered how a system of SLAs could be designed to mitigate these risks. In principle, several different systems of SLAs might be satisfactory to the participants. However, we attempted to design a system offering the highest possible level of monitorability for the SLAs that it contains. According to the results of the monitorability analysis described in Section 3, mutually monitorable SLAs in a three-party service-provision scenario with a client, provider, and network-service provider are the most monitorable SLAs achievable if latency conditions are required. The case-study scenario includes two similar subscenarios: the provision of the `Polymorph Search Webclient` by CS to Chemistry across the IS network, and the provision of the `plotws` web service by Southampton to CS across the Internet. In both cases latency conditions are required in order to mitigate the risk that the production of results is delayed.

These subscenarios differ from that considered in our original analysis in two respects: First, the eMaterials scenario contains five parties, rather than three, and the possible influence on monitorability of the two extra parties should not be neglected; and second, our original analysis assumed that the client and the provider of the service were nodes embedded in the network of the network-service provider. In the eMaterials scenario, in contrast, all computational nodes are embedded within networks controlled by the same organizations that control the nodes. Fortunately, in this case, these differences make only a small practical difference to the monitorability result, allowing the reuse of this result to inform the choice of SLAs in the scenario. In both subscenarios, the additional, uninvolved parties are clearly not respondents to any of the events in the interaction directly because they apparently have no means to perform trustworthy monitoring within the networks operated by their peers. Neither can they monitor the events indirectly by having them reported to them. Only the involved parties could do such reporting, and they are barred from doing so because they will necessarily have to enter SLAs concerning these events, and therefore have an interest, and therefore cannot report.

Service usages in both subscenarios still have end-to-end QoS requirements, such as latency requirements. Requests and responses pass over two extra network segments, owned by the client and the service provider, respectively, in addition to the segment owned by the network provider. However, the interfaces to these peripheral network sections may be regarded as being similar to nodes embedded in the central network. The provider of the electronic service in each case can guarantee the performance of their own network segment, and hence provide good service to the network provider at their mutual interface. The network provider can, therefore, guarantee good service at the interface to the client network, but not beyond. QoS guarantees can only be provided for the client as far as the boundary to the client's network. However, if the client manages their network correctly, this will allow them to guarantee the end-to-end QoS of the service, so this is adequate.

Consequently, we decided that an appropriate system of SLAs for the scenario must include SLAs to govern service provision at the interfaces between the Chemistry, IS, and CS networks, and the Internet and Southampton's network for the two interactions already discussed. These SLAs will be mutually monitorable, which will be the best degree of monitorability obtainable for these subscenarios, without the introduction into the scenario of additional parties, or trusted monitoring solutions. An additional SLA is required at the interface between CS and IS to govern the apparent behavior of the cluster nodes. Note that this SLA will also be mutually monitorable. The resultant system of five SLAs is shown in Fig. 9.

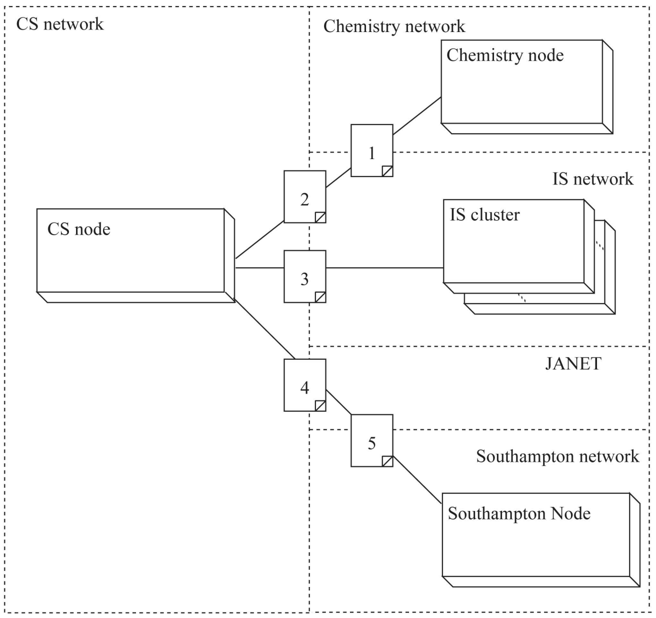


Fig. 9. SLAs for the eMaterials scenario, located at the network boundaries where pertinent events occur.

We next designed conditions for each of these SLAs. Usually, the violation of these conditions resulted in the need to pay a penalty, and often this implied an SLA safety risk for some party. For example, if a webpage was not returned promptly, IS could be required to pay a penalty to Chemistry under the terms of SLA 1. However, the fault may have been due to CS. Therefore, when designing the SLAs it was necessary to extend the risk analysis to consider SLA safety risks, and then add additional conditions to mitigate these risks. For example, a latency condition was added to SLA 2 to balance the risk to IS of the latency condition in SLA 1. As our monitorability result implied, it was eventually possible to design a completely safe system of SLAs mitigating all of the original risks and all derived SLA safety risks. At this point, we deferred consideration of how SLAs could address security risks to future work.

Having determined the purpose for each SLA, we next considered how these could be formalized using SLAng. At this point, we discovered that, without a much greater effort of analysis and measurement, it was difficult to decide on realistic parameter values for the SLAs, even though the nature of the risks and the conditions required to mitigate them were well understood. This was due to the effort required to determine both the usual performance of the service and the magnitude of financial risks associated with the service. We, therefore, focused on specifying the SLAs, assuming these parameters could be determined (or negotiated).

We noted that SLAs 1 and 2 have essentially the same structure, differing only in the parameter values required, because SLA 1 essentially resells the service sold by SLA 2, possibly relaxing latency and reliability conditions somewhat. The same is true of SLAs 4 and 5. Since we were largely ignoring parameter values, this limited the number of SLAs that we needed to author to SLAs 1, 3, and 4. We subsequently also abandoned the effort to specify SLA 3 due to the effort required to reverse engineer the protocols used by `Condor`, which are largely undocumented.

We were successful in our efforts to specify SLAs 1 and 4 using SLAng. In both cases, as expected, it was necessary to develop extensions to the core language in order to specify what was required. We now give a brief description of SLA 1, the more complex of the two.

Conditions included in SLA 1 are as follows:

1. latency conditions on the setup and invocation operations of the webclient;
2. an availability condition relating to announced service outages;
3. reliability conditions on the setup and invocation operations of the webclient;
4. a latency condition on the amount of time taken for simulation results to become available;
5. a reliability condition on results retrieval operations;
6. throughput conditions on all operations;
7. a payment scheme, charging chemistry for use of the service;
8. a limit on the rate at which simulations can be started;
9. a guarantee concerning the performance of the simulation executables provided by chemistry;
10. termination penalties applicable to either party.

In addition, the SLA needed to describe constraints on its administration and what standards of accuracy the parties must adhere to when gathering evidence for the calculation of violations. We decided that it would be appropriate for SLA 1 to be administered on a weekly basis. Penalties in the SLA were expressed as payments in Pounds Sterling, to be settled within a specified period of the administration in which the corresponding violation is decided.

Fig. 10 gives some examples of the extension classes required to specify SLA 1. These include classes for specifying penalty payments and several varieties of service-behavior restriction condition clauses. Minimal restrictions assign a penalty to any sufficiently long sequence of the restricted behavior, whereas maximal restrictions assign a single penalty to a sufficiently long sequence regardless of its total length. A minimal restriction with a fixed penalty is associated with a behavior definition identifying successful initiation of a simulation, and is used to implement per-use charging in the SLA. The various maximal restrictions implement latency, reliability, and throughput conditions. Latency and reliability conditions have a stepped-penalty regime, dependent on the length of the violating sequence of events. Throughput conditions apply no penalties. However, service-usage records can only be associated with various behaviors, such as latency and reliability failures, if no concurrent throughput violation has occurred.

In addition, complex conditions were required to express the latency conditions on the availability of results. Results become available in a deferred-synchronous manner, with the chemists needing to poll the webclient to recover them. Clearly, no party should be penalized for late results if the chemists do not poll. Moreover, the production of timely and correct results depends on the performance of executables provided by the Chemistry Department itself. Therefore, these conditions must be depended on the correct operation

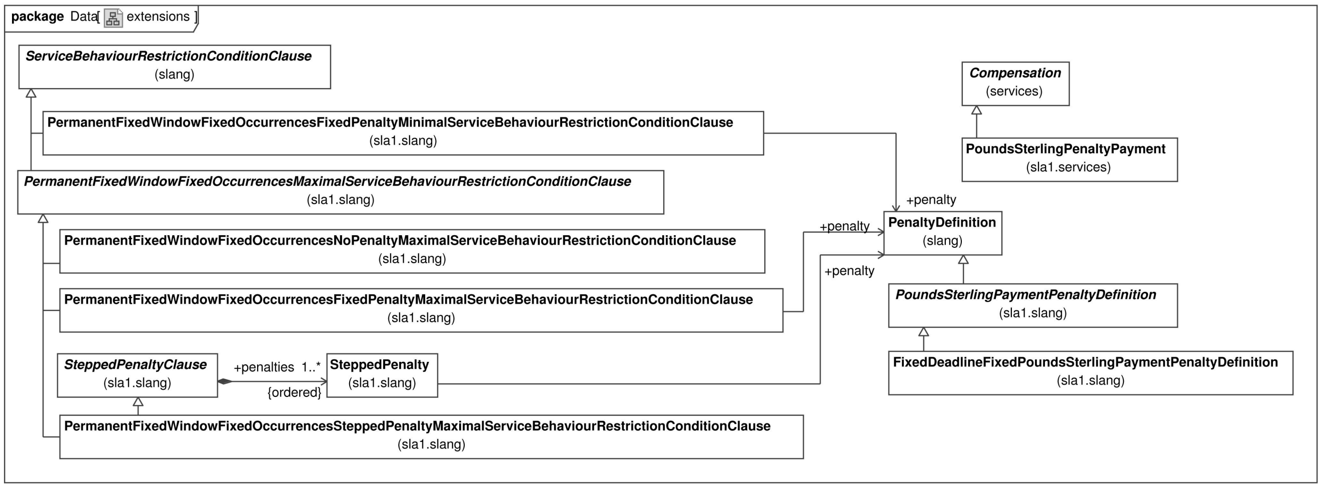

package Data[extensions]
ServiceBehaviourRestrictionConditionClause (slang)
PermanentFixedWindowFixedOccurrencesFixedPenaltyMinimalServiceBehaviourRestrictionConditionClause (sla1.slang)
PermanentFixedWindowFixedOccurrencesMaximalServiceBehaviourRestrictionConditionClause (sla1.slang)
PermanentFixedWindowFixedOccurrencesNoPenaltyMaximalServiceBehaviourRestrictionConditionClause (sla1.slang)
PermanentFixedWindowFixedOccurrencesFixedPenaltyMaximalServiceBehaviourRestrictionConditionClause (sla1.slang)
SteppedPenaltyClause (sla1.slang)
+penalties 1..*
{ordered}
SteppedPenalty (sla1.slang)
PermanentFixedWindowFixedOccurrencesSteppedPenaltyMaximalServiceBehaviourRestrictionConditionClause (sla1.slang)
Compensation (services)
PoundsSterlingPenaltyPayment (sla1.services)
+penalty
+penalty
+penalty
PenaltyDefinition (slang)
PoundsSterlingPaymentPenaltyDefinition (sla1.slang)
FixedDeadlineFixedPoundsSterlingPaymentPenaltyDefinition (sla1.slang)

Fig. 10. Example extensions to the SLAng language specification.

of these executables. We found it to be quite possible to formalize these rules in extensions using our metamodeling approach. Full details of both SLAs are given in [15].

In conclusion, the case study demonstrated that, although it may be difficult to determine exactly what needs to be expressed in SLAs for a given scenario, our approach to specifying the SLAs is practical, even for complex constraints. We were able to apply our monitorability result to design a mutually monitorable system of SLAs for the scenario. Our core SLAng language, which anticipates the need for timeliness, reliability, and throughput conditions, is an appropriate basis for defining the extensions and SLAs needed in the scenario. Following our metamodeling approach, this could be achieved with precision. In [15], we showed, with the assistance of a number of measurements, that the core SLAng language contributed a large part of the information conveyed by the SLAs in the case study. The formalization appears to be capturing significant knowledge, and may be reused in any number of SLAs to reduce the cost of their preparation without compromising precision.

6 Related Work on SLAs for Electronic Services

SLAs are currently employed in a variety of contexts, including in relation to electronic services, although the practice is far from ubiquitous. Also, SLAs are typically not highly precise instruments to mitigate risk, as we have suggested that they could be, but instead support a broader Service-Level Management (SLM) approach. In this section, we review the use of SLAs for SLM, and also prior academic work that has discussed SLAs for electronic services.

SLM is primarily concerned with maintaining the relationship between a business and an IT service provider, largely through the use of SLAs. Providers are either IT departments within an organization or companies specializing in the provision of IT services. In [32], the benefits of SLM are stated as being primarily related to managing client expectations, allowing the service provider to control the provisioning of a service (and avoid overprovisioning), and defend themselves against unwarranted criticism from users.

SLM generally assumes long-lived and high-value relationships between client and provider. In SLM, SLAs are created following a process of feasibility analysis and negotiation that, in itself, takes time and is costly. A typical term for an SLA is cited as being two years because shorter terms would render the cost of producing the SLA prohibitive. The need for a technical language to author SLAs is not emphasized. SLAs for SLM also tend to make availability an objective of prime concern, which, as we discuss in Section 3, is, at best, only monitorable by the provider.

At the inception of a service-provisioning relationship, an SLA may be regarded as an incentive offered by the service provider for the client to enter the relationship. However, once service provisioning commences, the SLA is a much greater liability for the provider than the client. Therefore, SLM can be regarded as primarily a marketing activity on the part of the provider. Negotiating service levels serves to build the relationship between the client and provider. However, once the SLA has been agreed, because it is informally written, and can only be monitored by the provider, the provider will have disproportionately more control over the relationship than the client; it will be able to decide on both the interpretation of the SLA and when events indicate that penalties should be paid.

Such relationships are acceptable when there is a tolerable level of trust between the parties, perhaps in addition to external forces that compel the provider to act in good faith. In contrast, we make no such assumptions about the culture in which an SLA is to be deployed. Precise, monitorable SLAs should be more attractive to clients than the kinds of SLAs in common use today. This could help grow the market in electronic-service provisioning. The drawback for the provider is that it will be obliged to implement services that meet strict standards.

SLAs for electronic services have received considerable attention in prior academic works, the majority of which are concerned with defining formal languages for stating SLAs. To our knowledge, our work given in [15] provides the first systematic enumeration of requirements for such SLAs, the most important of which we have discussed in this paper.

Requirements for contract languages are considered in relation to the Business Contract Language (BCL) in [33]. Here, the authors touch on high-level requirements for business contracts such as the inclusion of security provisions, access-control and obligation policies, specifications of standards for precision of measurements, feasibility of checking contract provisions, conditions relating to administration, and the need to have flexible specifications for states, events, and temporal constraints.

The inclusion of requirements related to policies (e.g., access control) indicates a wider intended scope for BCL than merely SLAs. However, the need to express security requirements is potentially of relevance to SLAs. Some security requirements can best be addressed by the functional behavior of a service—for example, authentication. However, others such as data privacy constraints may need to be expressed in an agreement. We have relegated the consideration of security conditions to later work. Overall, the requirements stated for BCL are largely in accord our own, although they are not systematically enumerated and, due to a lack of a definitive specification for BCL, it is hard to assess how many of these requirements have been met by the language.

Requirements for agreements of several types are also considered in work related to the X-contracts language [34]. Once again, these requirements cover much of the same ground as our own, although precision requirements are not emphasized, and measurement errors are not considered. Also, some of the requirements seem dubious as they appear to be based on the assumption that earlier requirements can be met. For example, a need to specify third-party monitoring solutions is cited. However, trustworthy third-party monitoring may not be feasible without further advances.

Some discussion of expressiveness requirements is also provided in relation to the Web-Service Management Language (WSML) [35], in which it is observed that SLA conditions may be related to arbitrary external factors, and several example conditions are given to support this argument.

Most recent approaches to defining SLA languages have provided, or asserted the availability of, an XML schema for their language. This includes the Web-Service-Level Agreement language (WSLA) [36], the Web-Services Offering Language (WSOL) [37], the Rule-Based Service-Level Agreement Language (RBSLA) [38], the Web-Services Agreement Specification (WS-Agreement) [39], BCL, and WSML. The use of XML is clearly intended to ease integration with other Web-services technology, such as WSDL or SOAP, that are also dependent on XML.

WSLA, WSOL, WSML, RBSLA, and WS-Agreement all rely on the provision of extensions of some kind to permit the complete expression of an SLA, with WSLA, WSML, and WS-Agreement providing abstract data types in their schemas to guide extensions, and RBSLA and WSOL relying on the use of externally provided ontologies (although the precise requirements for these remain unclear in both cases). The languages surveyed provide very little support for expressing latency, reliability, or throughput conditions. In each case, either syntax for such conditions is missing (therefore, the expression of such conditions relies entirely upon language extensions), or syntactic elements exist but are not accompanied by semantic definitions of sufficient precision to support the calculation of violations. The support provided by the abstract schema types is slight and guidance in producing extensions is not provided. In contrast, SLAng provides base classes that encode much of the required semantics for these types of conditions, with extensions required only to provide those details that are SLA specific. These extensions are largely straightforward to define as they involve the overriding of well-documented abstract operations. In addition, examples provided of the use of the languages are universally hypothetical, in contrast to SLAng, the expressiveness of which has been demonstrated in a case study involving a real service.

All of these languages suffer from imprecision due to a number of factors. Universally, a separation exists between the XML schema definition of the language and the language specification document or documents. This hinders traceability between SLAs and the definition of their semantics. WSLA, WSOL, WSML, and WS-Agreement all have informally defined semantics expressed solely using natural language. The semantics for RBSLA are incompletely specified. WSOL, RBSLA, and BCL have no definitive language specification document, and instead are described in collections of academic publications. Neither BCL nor X-contracts benefit from a publicly available definition of their syntax. Clearly, it is not currently feasible to adopt these languages as the basis for specifying SLAs with genuine financial implications, as the parties to these SLAs would have no strong basis for arguing for any particular interpretation of the SLAs in the event of a disagreement.

BCL, RBSLA, and X-contracts are principally concerned with the expression of rules in a similar manner to, or explicitly based on, deontic logic. Deontic logic allows the statement of permissions and obligations for parties to perform various actions [40]. This emphasis tends to create a language that makes it easy to describe the protocols by which interacting parties are bound, and results in useful SLA terminology in situations where the risk is primarily related to violations of this protocol. For example, the failure to deliver goods following the submission of a purchase order may result in an obligation to pay a penalty.

However, it is not clear how easy it is to use such semantics as a basis for precisely describing more quantitative conditions, such as a reliability condition limiting the number of failed service requests within a sliding window. In [41], formulas for calculating violations of quality-of-service constraints are referred to as "definitional components," and it is suggested that they should be treated separately from deontic norms.

Conversely, the highly expressive combination of EMOF and OCL, used to define SLAng and its extensions, supports the categorization of events and data in a very understandable way, thanks to its reliance on object orientation, and can also represent permissions and obligations implicitly by associating violations with the history of monitored events. BCL and RBSLA tend to obscure the semantics of complex conditions by relying on external definitions. X-contracts, which have a representation and semantics based on finite-state machines, are likely to require unfeasibly complicated statements to represent conditions pertaining to a large amount of service history due to the state-explosion problem.

These contract languages do highlight the advantages of a more restricted formal underpinning in work relating to the validation of contracts: for example, checking for conflicting obligations or asserting liveness properties of the protocols being described. EMOF and OCL, by contrast, allow testing of these properties.

X-contracts are the only prior work to consider monitorability in any sense. The authors recommend that a middleware supporting nonrepudiable message exchange should be used to monitor contracts. This would make it impossible to deny violations related only to positive actions (for example, violations of prohibitions). However, this does not seem to solve monitorability issues related to obligations or temporal constraints because of the difficulty of attributing the cause of delays or faults to the action of a single party in the case where interaction is with multiple remote parties, for example, both an electronic and network-service provider.

To our knowledge, SLAng is unique in providing support for accuracy constraints in relation to the gathering of evidence. SLAng's support for, and emphasis on conditions relating to the termination of SLAs is also novel despite the importance of mitigating termination risks. Both requirements are mentioned in early work related to BCL, but not elaborated upon in later work describing the language.

Older related work is principally concerned with describing QoS for various types of electronic-service interface. This includes OWL-S [42], QML [43], and QuO-QDL [44]. This work relies on the implicit assumption that service providers can be trusted to describe the quality-of-service provided by their own services, and then deliver services to that level. In our view, this is a fundamentally unrealistic assumption, and the languages are unsuitable for describing SLAs since they do not allow the specification of penalties. Moreover, the definitions of the metrics are typically informal, hindering reliable analysis.

7 Conclusions and Future Work

In this paper, we have described our motivation and progress in working to develop language support for SLAs with the principal purpose of mitigating outsourcing risks related to electronic services. We have argued that an inability to control the magnitudes of these risks is an inhibiting factor in the adoption of the outsourcing model for electronic services. SLAs can be used to mitigate these risks. For these SLAs to be reliable from the client's point of view, they need to be precise, understandable, and monitorable so that the meaning of the SLA cannot be disputed, and trustworthy evidence can be brought to bear in the event of a dispute between the parties. In current practice, SLAs are typically neither precise nor monitorable. This may in part be because, once agreed upon, SLAs represent a liability to a service provider. However, it is also the case that prior work has not shown how these properties can be incorporated in a DSL for SLAs, whereas ours has. We have therefore provided a facility such that if providers wish to provide their clients with a greater incentive to enter outsourcing agreements, potentially growing the market in outsourcing of electronic services, albeit assuming a greater responsibility for respecting the SLAs that they offer, then they may.

In our discussion of monitorability, we observed that, under our assumptions concerning the intrinsic visibility of events and what parties may be trusted to report events, achieving highly monitorable SLAs will require ISPs to become involved with the resale of individual electronic services. They will also need to implement monitoring at the edges of their networks. Clearly the costs in doing so are undesirable. This mandates further research into trusted monitoring platforms, which may modify assumptions concerning visibility, or consideration as to how trust can be achieved in situations where monitorability is reduced. However, in lieu of further progress on these topics, ISPs may wish to consider the possibility of reselling services.

Other avenues for future research are to consider how SLAs can contribute to mitigating security risks, and to continue the maturation of SLAng toward a version that can be used in many cases without requiring significant effort to define extensions. This latter effort will require the identification of SLA terminology that is commonly required based on further practical investigation of electronic-service-provisioning scenarios. This could perhaps be best achieved in the context of a standardization effort.

Acknowledgments

Jason Crampton, Allan Skene, and Domenico Davide Lamanna were joint authors of some of the papers, the results of which we have summarized here. This research was funded by the IST project TAPAS and the EPSRC Divergent Grid project. We are also grateful for the contributions of our partners in the IST project PLASTIC to the continued development of SLAng and to Zuhlke Engineering Ltd. for supporting the preparation of this manuscript.

References

[1] A. Susarla, A. Barua, and A.B. Whinston, "Understanding the 'Service' Component of Application Service Provision: An Empirical Analysis of Satisfaction with ASP Services," *Information Systems Outsourcing,* Springer, pp. 481-521, 2006.

[2] D. Greschler and T. Mangan, "Networking Lessons in Delivering 'Software as a Service'—Part I," *Int'l J. Network Management,* vol. 12, no. 5, pp. 317-321, 2002.

[3] M.A. Rappa, "The Utility Business Model and the Future of Computing Services," *IBM Systems J.,* vol. 43, no. 1, pp. 32-42, 2004.

[4] D. Bogatin, *Google CEO's New Paradigm: "Cloud Computing and Advertising Go Hand-in-Hand,"* ZDNet, http://blogs.zdnet.com/micro-markets/?p=369, Aug. 2006.

[5] H. Davulcu, M. Kifer, L.R. Pokorny, C.R. Ramakrishnan, I.V. Ramakrishnan, and S. Dawson, "Modeling and Analysis of Interactions in Virtual Enterprises," *Proc. Ninth Int'l Workshop Research Issues on Data Eng.: Information Technology for Virtual Enterprises,* pp. 12-18, Mar. 1999.

[6] M.N. Huhns and M.P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing,* vol. 9, no. 1, pp. 75-81, Jan./Feb. 2005.

[7] *Web Services Architecture,* The World Wide Web Consortium (W3C), http://www.w3.org/TR/ws-arch/, Feb. 2004.

[8] *Architecture of the World Wide Web,* vol. 1, W3C, http://www.w3.org/TR/webarch/, Dec. 2004.

[9] *Programming with ONC RPC,* Digital Equipment Corporation (DEC), http://www.cs.arizona.edu/computer.help/policy/DIGITALunix/AA-Q0R5B-TET1_html/TITLE.html, 1992.

[10] A. Sinha, "Client-Server Computing," *Comm. ACM,* vol. 35, no. 7, pp. 77-97, July 1992.

[11] *SOA Practitioners Guide Part I,* BEA Systems, Inc., http://dev2dev.bea.com/2006/09/SOAPGPart1.pdf, 2006.

[12] "Amazon Elastic Compute Cloud (Amazon EC2)," http://aws.amazon.com/ec2, 2009.

[13] "Salesforce.com," http://www.salesforce.com/, 2009.

[14] P. Malinverno, "How to Get Value Now (and in the Future) from SAP's Enterprise SOA," Technical Report G00150358, Gartner, Inc., Sept. 2007.

[15] J. Skene, "Language Support for Service-Level Agreements for Application-Service Provision," PhD dissertation, Univ. of London, Oct. 2007.

[16] D.A. Menascé and V.A.F. Almeida, *Capacity Planning for Web Services.* Prentice Hall, Inc., 2001.

[17] J. Skene, A. Skene, J. Crampton, and W. Emmerich, "The Monitorability of Service-Level Agreements for Application-Service Provision," *Proc. Sixth Int'l Workshop Software and Performance,* pp. 3-14, 2007.

[18] J. Skene, D.D. Lamanna, and W. Emmerich, "Precise Service Level Agreements," *Proc. 26th Int'l Conf. Software Eng.,* pp. 179-188, May 2004.

[19] *MDA Guide Version 1.0.1,* omg/2003-06-01 ed., The Object Management Group (OMG), June 2003.

[20] A.S. Evans and S. Kent, "Meta-Modelling Semantics of UML: The pUML Approach," *Proc. Second Int'l Conf. Unified Modeling Language,* pp. 140-155, 1999.

[21] J. Skene and W. Emmerich, "Specifications, Not Meta-Models," *Proc. 2006 Int'l Workshop Global Integrated Model Management,* pp. 47-54, 2006.

[22] *Meta-Object Facility Core Specification Version 2.0,* formal/2006-01-01 ed., The Object Management Group (OMG), Apr. 2002.

[23] *UML 2.0 Superstructure Final Adopted Specification,* ptc/03-08-02 ed., The Object Management Group (OMG), 2002.

[24] *UML 2.0 OCL Final Adopted Specification,* ptc/03-10-14 ed., The Object Management Group (OMG), Oct. 2003.

[25] J. Skene and W. Emmerich, "Generating a Contract Checker for an SLA Language," *Proc. Int'l IEEE Enterprise Distributed Object Computing Conf. Workshop Contract Architectures and Languages,* 2004.

[26] "The SLAng Open-Source Project," http://uclslang.sourceforge.net/, 2009.

[27] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S.L. Price, "Grid Service Orchestration Using the Business Process Execution Language (BPEL)," *J. Grid Computing,* vol. 3, no. 3, pp. 283-304, Sept. 2005.

[28] "The ActiveBPEL Open Source Engine Project," http://www.active-endpoints.com/active-bpel-engine-overview.htm, 2008.

[29] "The Condor Project," http://www.cs.wisc.edu/condor/, 2009.

[30] "GridSAM—Grid Job Submission and Monitoring Web Service," http://gridsam.sourceforge.net/2.0.1/index.html, 2009.

[31] "JANET—The UK's Education and Research Network," http://www.ja.net/, 2009.

[32] R. Sturm, W. Morris, and M. Jander, *Foundations of Service Level Management.* SAMS, 2000.

[33] S. Neal, J. Cole, P. Linington, Z. Milosevic, S. Gibson, and S. Kulkarni, "Identifying Requirements for Business Contract Language: A Monitoring Perspective," *Proc. Seventh Int'l Enterprise Distributed Object Computing Conf.,* M. Steen and B.R. Bryant, eds., pp. 50-61, Sept. 2003.

[34] C. Molina-Jimenez, J. Pruyne, and A. van Moorsel, "The Role of Agreements in IT Management Software," *Architecting Dependable Systems III,* R. de Lemos, C. Gacek, and A. Romanovsky, eds., pp. 36-58, Springer, 2005.

[35] A. Sahai, A. Durante, and V. Machiraju, "Towards Automated SLA Management for Web Services," Technical Report HPL-2001-310R1, HP Laboratories, http://www.hpl.hp.co.uk/techreports/2001/HPL-2001-310R1.html, 2001.

[36] *Web Service Level Agreement (WSLA) Language Specification,* Int'l Business Machines (IBM), Inc., Jan. 2003.

[37] V. Tosic, "Service Offerings for XML Web Services and Their Management Applications," PhD dissertation, Ottawa-Carleton Inst. for Electrical and Computer Eng., 2002.

[38] A. Paschke, "RBSLA—A Declarative Rule-Based Service Level Agreement Language Based on RuleML," *Proc. Int'l Conf. Intelligent Agents, Web Technology and Internet Commerce,* 2005.

[39] *Web Services Agreement Specification (WS-Agreement) Version 2005/09,* Open Grid Forum, http://www.ogf.org/Public_Comment_Docs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf, 2006.

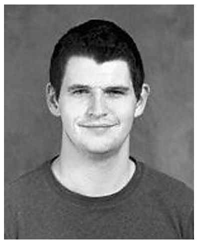
[40] G.H. von Wright, "Deontic Logic," *Mind,* vol. 60, pp. 1-15, 1951.

[41] A.J.I. Jones and M. Sergot, "On the Characterization of Law and Computer Systems: The Normative Systems Perspective," *Deontic Logic in Computer Science: Normative System Specification,* pp. 275-307, John Wiley & Sons, 1993.

[42] *OWL-S: Semantic Markup for Web Services,* The World Wide Web Consortium (W3C), http://www.w3.org/Submission/OWL-S/, Nov. 2004.

[43] S. Frolund and J. Koistinen, "QML: A Language for Quality of Service Specification," Technical Report TR-98-10, HP Laboratories, 1998.

[44] J.P. Loyall, R.E. Schantz, J.A. Zinky, and D.E. Bakken, "Specifying and Measuring Quality of Service in Distributed Object Systems," *Proc. First Int'l Symp. Object-Oriented Real-Time Distributed Computing,* pp. 43-52, Apr. 1998.



James Skene received the PhD degree in computer science from University College London (UCL) in 2007 for his work on language support for service-level agreements. He received the BSc degree in computer science from UCL in 2000. He currently works as a researcher and consultant software engineer in Auckland, New Zealand. His research interests focus on domain-specific languages, domain modeling, and programming. He is the author of a number of publications on topics related to language design, performance modeling, and monitoring. He recently served on the program committee of QoSCSOA '08. He is a member of the IEEE.

Franco Raimondi received the PhD degree in computer science from University College London (UCL) in 2006, and the BSc and MSc degrees in physics from the University of Milan. He is currently a Senior Lecturer in the School of Engineering and Information Sciences at Middlesex University in London. His research interests include model checking for extensions of temporal logics, multiagent systems, and modal logics. He is the author of various publications in software engineering, AI, and formal methods. He is actively involved in the organization of various conferences and workshops.

Wolfgang Emmerich received the PhD degree in computer science from the University of Paderborn and the MSc degree in informatics from the University of Dortmund in Germany. He is a professor of distributed computing at University College London (UCL), a leading British university. He heads the Software Systems Engineering Research Group in the Department of Computer Science, where he is currently also the director of research. He is a member of London Software Systems. Prior to joining UCL, he held a lectureship at the City University in London and was a visiting research fellow in the Software Verification Research Centre at the University of Queensland in Brisbane, Australia. His research interests are in the area of software architectures for large-scale distributed and mobile systems. He is a member of the editorial board of the *IEEE Transactions on Software Engineering*. He has served on numerous program committees of international conferences in software engineering and distributed systems. He is a chartered engineer and a member of the IEEE Computer Society, the ACM, and the Institution of Engineering and Technology. He is also a cofounder and partner of the Zuhlke Technology Group, a medium-sized pan-European service provider of systems engineering services.