Precise Service Level Agreements*

James Skene, D. Davide Lamanna, Wolfgang Emmerich
Department of Computer Science, University College London
London, WC1E 6BT, UK
{j.skene, d.lamanna, w.emmerich}@cs.ucl.ac.uk

Abstract

SLAng is an XML language for defining service level agreements, the part of a contract between the client and provider of an Internet service that describes the quality attributes that the service is required to possess. We define the semantics of SLAng precisely by modelling the syntax of the language in UML, then relating the language model to a model that describes the structure and behaviour of services. The presence of SLAng elements imposes behavioural constraints on service elements, and the precise definition of these constraints using OCL constitutes the semantic description of the language. We use the semantics to define a notion of SLA compatibility, and an extension to UML that enables the modelling of service situations as a precursor to analysis, implementation and provisioning activities.

1 Introduction

Increasingly, distributed systems primitives are being used to build mission-critical applications that cross autonomous organizations. Examples include the use of web services in, for example, supply chain management, or computing on demand using distributed component or grid technology as offered by IBM. Because the usefulness, and sometimes even the functioning of a business, depend not only on the functionality but also the quality of these services, for example performance and reliability, and because these qualities not only depend on the behaviour of the service but on that of the client, contracts between the provider and client of a service must contain terms governing their individual and mutual responsibilities with respect to these qualities. Such clauses are called Service Level Agreements (SLAs) and previous work has proposed specialised languages to represent SLAs, for the purpose of easing their preparation, automating their negotiation, adapting services automatically according to their terms, and reasoning about their composition.

*This research is partly funded through the EU project TAPAS (IST-2001-34069)

SLAng [6] is an SLA language that differs from previous work in three significant respects:

Firstly, in contrast with other languages that focus on web services exclusively, it defines SLA vocabulary for a spectrum of Internet services, including Application Service Provision (ASP), Internet Service Provision (ISP), Storage Service Provision (SSP) and component hosting, motivated by the observation that federated distributed systems must manage the quality of all aspects of their deployment.

Secondly, the structure of SLAng is derived from industrial requirements [12]. It resembles SLAs currently in use in an effort to provide realistic terms that are both useful and usable. For example, as well as imposing latency and reliability constraints for application services, SLAng can specify obligations to report monitoring data and provide data backup services.

Thirdly, the meaning of SLAng is formally defined in terms of the behaviour of the services and clients involved in service usage, with reference to a model of service usage. This is in contrast to other SLA languages that depend on the definition of sets of metrics whose correspondence to actual system behaviour is only informally defined and hence open to a greater degree of interpretation.

This paper describes the approach taken to produce a rigorous yet understandable specification of the semantics of SLAng, and the use of the semantics in reasoning about service composition.

We use the Unified Modelling Language (UML) [10] to model the language, producing an abstract syntax. We relate this language model to an abstract model of services, service clients and their behaviour. The presence of SLAs, instances of the language model, constrains the behaviour of the associated services and service clients. The constraints are defined formally using the Object Constraint Language (OCL) [20], with accompanying natural language descriptions, and define the semantics of the language. The semantics are easily understood in the context of the service model. We exemplify the approach in Section 3 by describing SLAng's semantics for ASP SLAs.

Benefits of the formal semantics include the reduction of ambiguity in the meaning of the language, because disputes over the meaning of SLAs are restricted to determining the correspondence between elements in the service model, and services and events in the real world. Having determined this correspondence, the effect of SLA terms is unambiguously dictated by the semantics of the OCL. The model also provides a formal basis for comparisons between SLAs, and an abstract reference model of systems employing SLAs that can guide implementation and analysis efforts. These latter facilities address two types of compositionality for SLAs: *inter-service composition* in which required QoS levels are compared to offered QoS levels, and *intra-service composition* in which the QoS levels offered by a service are related to the levels provided by its components.

Inter-service composition requires the matching of desired service levels, expressed as target SLA terms, with offered service levels. SLAs are deemed to be compatible if the offered SLA permits no behaviours (according to the service model) that would violate the target SLA. Inter-service composition is discussed in Section 4.

Intra-service composition requires a specification of the behaviour of a system extraneous to SLAs. We therefore define an extension of UML to permit the representation of SLAs in system designs. This allows the inclusion of SLA information in the context of detailed designs describing the behaviour of services and the impact of their components on quality attributes. The combined model of SLAng syntax and services describes the semantics of this extension, and should be used as a reference for analysis and implementation activities proceeding from models that employ the extension. The UML extension is provided in the form of a QoS catalogue appropriate to the proposed 'UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms' (henceforth 'QoS profile') [8], and is described in Section 5.

The next section describes SLAng in more detail. Section 3 gives an example of the semantic definition of SLAng by presenting the semantics of ASP SLAs. Section 4 discusses the compatibility of SLAs, addressing inter-service composition. Section 5 presents the QoS catalogue for the ASP part of SLAng, addressing intra-service composition. Section 6 discusses the differences between SLAng and other SLA languages in terms of approach and style of semantic definition. Section 7 summarizes and Section 8 discusses future work.

2 SLAng

SLAng meets the need for an SLA language to support construction of distributed systems and applications with reliable QoS characteristics. In earlier work, the syntactic structure and semantics of SLAng were defined with reference to an informal model of distributed system architecture. The model represents the scope of the language, and has assisted in identifying the service usage scenarios and parameters that SLAng must represent. The informal reference model is shown in Figure 2, slightly modified from [6].

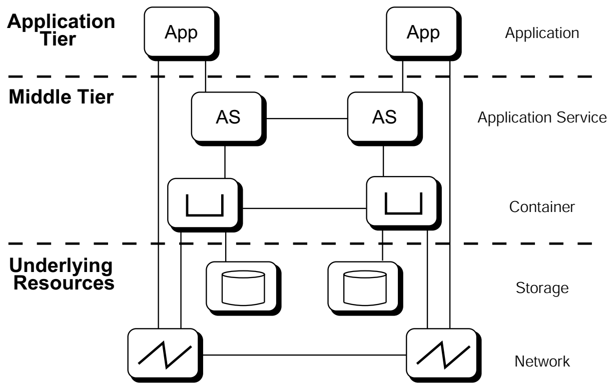

Figure 1. Reference Model of Distributed System Architecture

In our model, applications are clients that use application services to deliver end-user services. Application services are services with an electronic interface, such as web services or J2EE or .NET components. Containers host application services and are responsible for managing the underlying resource services for communication, transactions, security and so forth. Networks provide communication between services and storage can implement persistence for containers.

All SLAng SLAs include: An end-point description of the contractors (e.g. information on customer/provider location and facilities); contractual statements (e.g. start date and duration of the agreement); and Service Level Specifications (SLSs), i.e. the technical QoS description and the associated metrics.

SLAng defines six different types of SLA, corresponding to service usages present in the model. These are informally divided into *vertical* SLAs, in which a service provides infrastructure support for a client, and *horizontal* SLAs, in which the client subcontracts part of its functionality to a service of the same type. The hierarchical structure of SLAng's syntax subdivides the SLS terms into SLA-type specific groups. The terms are further subdivided into client, provider and mutual responsibility clauses.

The vertical SLAs are *Hosting* (between service provider and host), *Persistence* (between a host and storage service provider) and *Communication* (between application or host and Internet service providers). The horizontal SLAs are

ASP (between an application or service and ASP), *Container* (between container providers) and *Networking* (between network providers).

The SLAng syntax is defined using XML Schema. The choice of XML as a basis for the language reflects the popularity of XML in the domain of distributed systems. In particular XML documents are frequently used to provide service meta-data [23, 13] and deployment descriptors [19]. By adopting XML as a basis for SLAng we seek to ease the integration of technologies depending on SLAng statements, such as QoS adaptation and negotiation, with existing Internet service technologies.

We have aimed to provide support for a range of useful types of SLA. [6] describes the validation of the language using a case study from industry. SLAng does not include syntactic statements for extensibility. However, we recognise that there will be users who find that SLAng does not express the agreements that they wish to make. To be confident in the precision of new types of agreement, such users should reproduce our approach to defining SLAng, which is to define a syntax and give it semantics with reference to a model of the service provision involved, as described in the next section. Extensibility mechanisms to support this would necessarily be nearly as general as XML schema, UML and OCL, so we prefer to document our approach, rather than provide extensibility support through the language.

3 SLAng Semantics

3.1 Approach

The approach taken to formalising SLAng adapts the approach of the precise UML group to formalising UML [2]:

1. The language itself is modelled in UML. This creates a meta-model, or abstract syntax. Statements in the language can be regarded as being composed of objects that are instances of the types in this model, or more informally: SLAs can be considered instances of this model. However, in the case of SLAng, the XML schema provides the primary definition of the language. It was therefore necessary to manually translate this into UML. Figure 2 shows the abstract syntax for ASP SLAs. It reflects the hierarchical structure of the XML schema for SLAng, with the top level defining the type of the SLA and separate clauses for provider and client responsibilities (there are no mutual responsibilities in this example).

2. The parties and services involved in the agreement are modelled. In the case of SLAng, the informal reference model of Figure 2 provided a starting point. This was translated into UML and refined. Figure 3 shows the refined reference model for application service provision. Additional concepts are introduced to support the definition of semantics for terms in SLAng. For example, an ASP SLA can specify the types of backup and monitoring solution used. BackupSolution and MonitoringSolution are introduced to allow the assertion that the solutions used in the real world must correspond to those specified in the SLA.

   Defining the type of software used for backup and monitoring may seem to contribute little to the principle goal of ensuring QoS for distributed applications. However, the industrial requirements on which SLAng is based indicated that clients frequently required this. The capacity to formalise clearly such apparently informal constraints is a significant benefit of the approach taken.

3. The behaviour of the parties and services involved in the agreement is modelled, using the reference model as a basis. Figure 4 shows the behaviour of application services and their clients. The primary interaction in this case is the ServiceUsage, an interval of time during which the client is invoking an operation of the service. Operations are capabilities of the service that can be used atomically. The client must be able to detect the beginning of the usage and its successful completion. Also, if the usage appears to have completed, the client can detect whether a failure occurred.

4. The language model is related to the elements that the SLAs are intended to constrain. This can be seen in Figure 2, where the classes ApplicationService, Operation, ServiceClient, BackupSolution and MonitoringSolution are reference model elements, associated with the relevant clauses in the language model.

5. The semantics of SLAng are defined by the constraints imposed on the behavioural model by the presence of SLA elements. These are expressed using OCL constraints defined in the context of the SLA clauses. The SLA is associated with a service and service client, so the constraints can refer to these entities and place conditions upon them. For readability, the constraints are also expressed in natural language.

The complete set of constraints defining the meaning of the ASP SLA are documented in [17].

3.2 Example

We now present the OCL definition of the constraint used to define the meaning of maximum latency and reliability.

Reliability in the ASP SLA is defined in terms of failed or overdue usages of the system. Each failure gives the

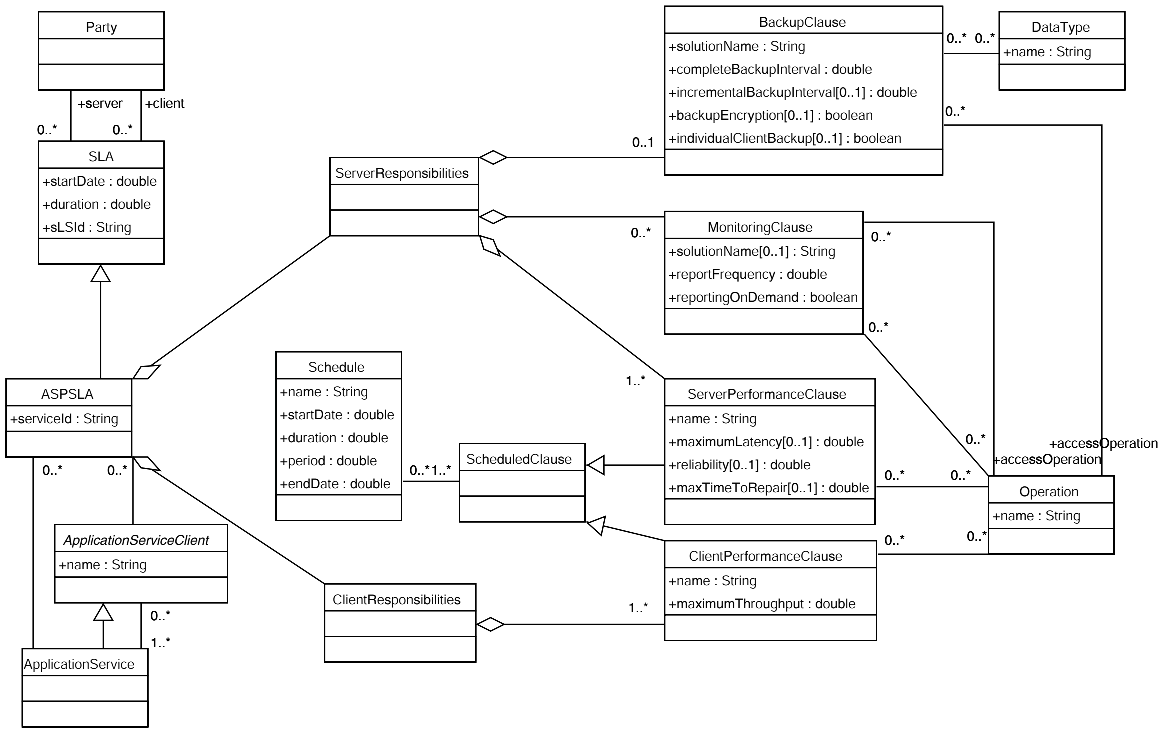

Figure 2. Abstract Syntax of SLAng

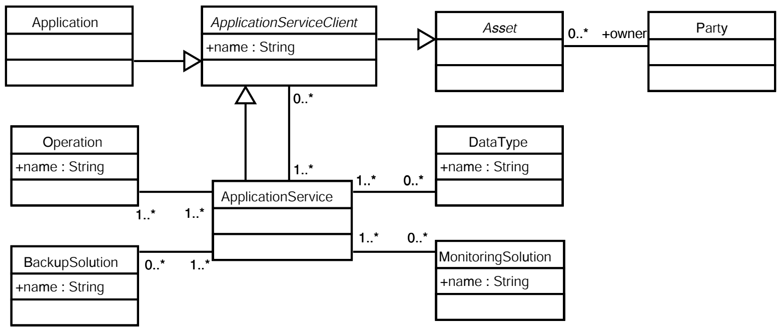

Figure 3. Refined reference model for application service provision

client leave to assume an interval of service outage equivalent to the inter-invocation time if they were using the service at the maximum allowable rate. This is because if the client does not use the service then they cannot reasonably claim that it is unavailable. The maximum allowable rate is the strictest client performance clause applying at the moment the failed operation is invoked.

Server and client performance clauses in the SLA are associated with schedules. A schedule defines when the clause applies, by specifying a start date, an end date, a duration and a period. The clause applies repeatedly, lasting for the duration, then becoming inactive until the period is complete. Multiple schedules can be associated with a clause to allow the specification of complex timing, with the interpretation that the clause applies when any one of its schedules apply. For example, five schedules with durations of 8 hours, periods of 1 week and start dates offset by 1 day can be combined to specify the composite schedule ‘every working day’.

The following OCL definitions refer to types present in

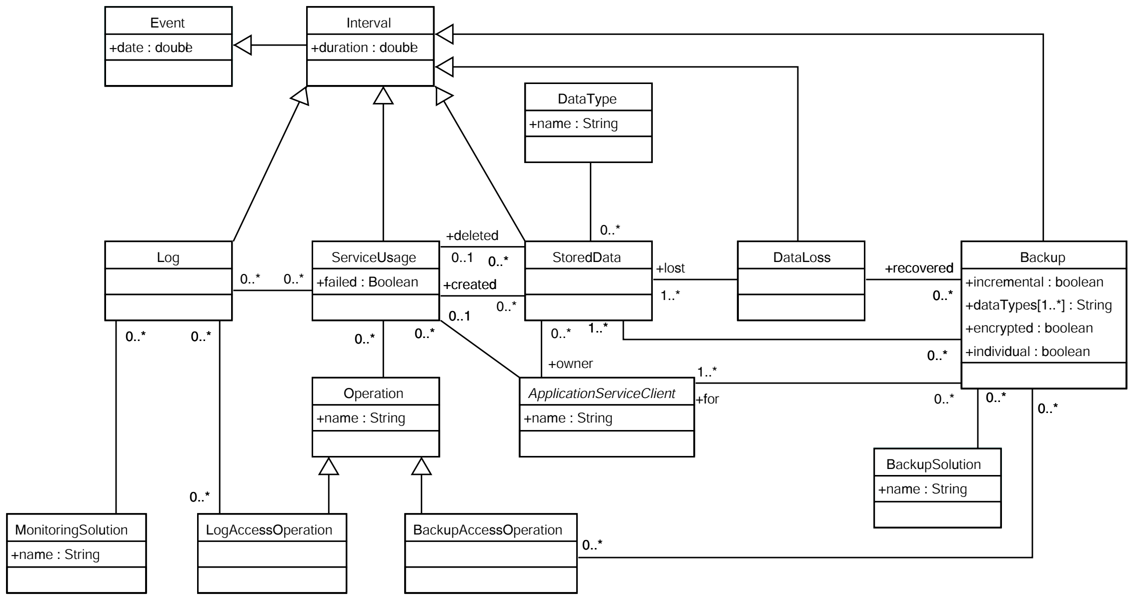

Figure 4. Behavioural model for application service provision

the models presented in Figures 2, 3 and 4. Each is defined in the context of the class ServerPerformanceClause in Figure 2, meaning that the constraints apply to all instances of that class, i.e. all server performance clauses in the real world. The operations and usages associated with the clause are referred to in the constraints by navigating across the associations present in the UML models, using the dot operator (.) and the name of the opposite association end, usually the name of the associated class with a lower-case first character. Attribute values and OCL operations are referred to using the same syntax. OCL operations are side-effect free operations defined in the context of classes, and we have used them to decompose complex constraints. By convention we have omitted their signatures from the operations compartment of our diagrams.

Reliability constraint: Proportion of downtime observed to total time that the clause applies must not be greater than the percentage of permitted failures (1 - the reliability).

context ServerPerformanceClause **inv**:
self.operation→forAll(o |
totalDowntime(o) < (applicationTime ⋆ (1 - reliability)))

The following additional OCL operations support the definition of the reliability constraint:

context ServerPerformanceClause **def**:
−− *Returns the client performance clauses governing the performance of operation o at time t.*
let applicableClientClauses(t : double, o : Operation) =
sla.clientResponsibilities.clientPerformanceClause→select(c |
c.schedule→exists(s | s.applies(t)))

−− *An expression for the maximum throughput with which the client can use an operation at time t, or −1 if there is no limit*
let minThroughput(t : double, o : Operation) =
if applicableClientClauses→isEmpty() **then** −1
else applicableClientClauses→iterate(
c : ClientPerformanceClause, minTP : double |
minTP.min(c.maxThroughput))

−− *Amount of downtime observed for a failure at time t. This is 1 / the most restrictive throughput constraint applicable at the time*
let downtime(t : double, o : Operation) : double =
if minThroughput(t, o) <= 0 **then** 0
else 1 / minThroughput(t, o)

−− *Total amount of downtime observed for the operation*
let totalDowntime(o : Operation) : double =
o.serviceUsage→select(u |
(u.failed or u.duration > maximumLatency) **and**

```
  schedule→exists(s | s.applies(u.date))
 )→collect(u | downtime(u.date, o))→iterate(
  p : double, sumP : double | sumP + p)
```

The constraint also relies on the definition of the following operations: *applicationTime*, defined in the context of ScheduledClause, the superclass of ServerPerformanceClause, which evaluates to the total time for which a ScheduledClause is applicable; and *applies*, defined in the context of Schedule, which evaluates to true if the schedule applies at the specified time and date.

The complete set of operations, combined with the constraint, defines the effect of the SLA on the environment. The definition can be unambiguously interpreted according to the semantics of the OCL, providing a strong reference for parties employing SLAs.

4 Inter-service Composition of SLAs

We say that an SLA, A, is *compatible* with another B if the set of allowable behaviours for A is a subset of those for B. In other words, a system conforming to A would never violate B, and would hence be perfectly acceptable to a client requiring B.

The notion of compatibility supports inter-service composition. One service can require another and express its requirements using an SLA. Any service that both provides the required functionality and offers a compatible SLA can be composed to fulfil the requirements.

The opportunity to definite compatibility between required and provided QoS levels in this way is a direct consequence of the style of semantic definition used, namely that the meaning of QoS terms is defined in terms of behaviours explicitly identified by a model. The definition potentially provides a method for comparing required levels of service with offered levels of service in a way that leads to safe decisions, because the definition does not rely on any subjective judgment concerning the behaviour of the system by a human user. However, it currently suffers from two drawbacks that we will attempt to address in future work.

Firstly, compatibility is difficult to check. One possible procedure for checking if an SLA, A is compatible with a target SLA, B is to employ the model of services as a data model in a checking program. All behaviours acceptable to A could be generated (which might be thought of as the set of all traces of system behaviours), then checked according to the criteria defined in B. Clearly this approach is potentially extremely computationally expensive, and unworkable if the possible behaviours of A are infinite. It would be preferable to employ a theorem prover to establish the validity of $c(A) \rightarrow c(B)$ where $c(A)$ is the union of the constraints in A and $c(B)$ the equivalent for B. Future work will investigate this approach.

Secondly, our definition only allows the comparison of fully specified SLAs, which include restrictions on both client and server behaviours. In practice, parties expressing QoS requirements may not be interested in restrictions on their own behaviour, so it may be preferable to compare partial SLAs. However, the definition of compatibility reveals a potential danger in this, due to interactions between SLA terms. For example, in the definition of reliability presented in the previous section there is a relationship between the invocation rate constraint on the client and the reliability. Therefore, an SLA offering high reliability at a faster maximum rate is not necessarily compatible with an SLA requiring lower reliability, but at a lower rate, as the absolute number of failures may be the property of concern for the client, rather than the absolute number of successes. Future work will consider the types of judgment that can be safely made by comparing the behaviour of systems constrained by SLAs in which some terms are not specified.

Our definition of compatibility is similar to that of *conformance*, defined in relation to the language QML [4], in which a contract A conforms to another B if its constraints are *stronger*. Each SLA dimension defined for a contract in QML has a user-defined direction indicating an ordering over values of the metric, with higher values implying stronger constraints. Conformance of contracts can therefore be assessed by comparing the values of corresponding dimensions in two contracts. In comparison, our definition of compatibility is hard to check and somewhat inflexible. However, its reliance on the semantic definition of the SLA terms, rather than on user-defined ordering of metric spaces suggests that the concept offers safer guarantees that requirements will be met, particularly in the presence of dependencies between SLA terms as discussed above.

5 QoS Catalogue for SLAng

Reasoning about internal composition of SLAs is a special case of QoS prediction, in which some components are governed by SLAs and the system as a whole will conform to, or offer an SLA. QoS prediction requires a view of the behaviour of the system, and the effect of system components on quality attributes. It may also require sophisticated analysis to determine the emergent QoS values.

UML potentially offers a view of systems including structural and behavioural aspects, SLA information, and QoS information related to components. It can represent the logical structure, deployment and behaviour of a service. It can also be extended using profiles to enable the description of SLAs, and QoS properties. In this section we describe how SLAng SLAs may be modelled using UML.

A profile is a semantic extension for UML allowing it to naturally model domains of interest [10]. It is common to define the semantics of a profile with reference to a domain

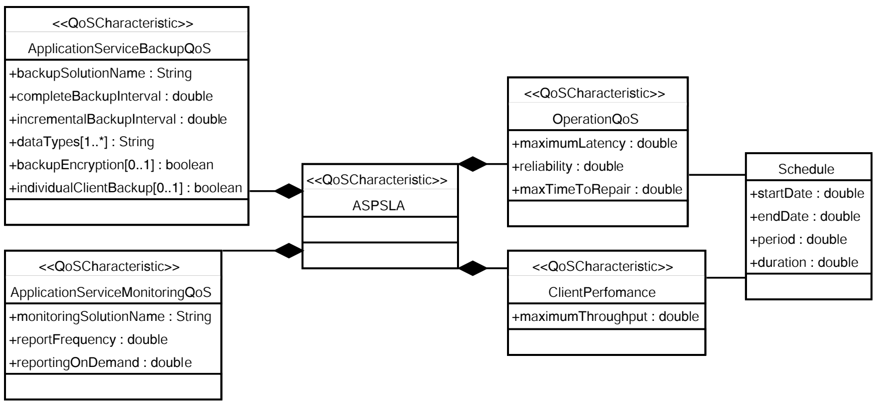
<<QoSCharacteristic>>
ApplicationServiceBackupQoS
+backupSolutionName : String
+completeBackupInterval : double
+incrementalBackupInterval : double
+dataTypes[1..*] : String
+backupEncryption[0..1] : boolean
+individualClientBackup[0..1] : boolean
<<QoSCharacteristic>>
ApplicationServiceMonitoringQoS
+monitoringSolutionName : String
+reportFrequency : double
+reportingOnDemand : double
<<QoSCharacteristic>>
ASPSLA
<<QoSCharacteristic>>
OperationQoS
+maximumLatency : double
+reliability : double
+maxTimeToRepair : double
<<QoSCharacteristic>>
ClientPerfomance
+maximumThroughput : double
Schedule
+startDate : double
+endDate : double
+period : double
+duration : double

Figure 5. ASP QoS Characteristics for SLAng catalogue

model [3]. The interpretation is that elements in the UML model employing an extended syntax that include stereotype labels and tagged values, correspond to objects from a subset of the types defined in the domain model. Other types in the domain model disambiguate the role of the elements in the real world. We use exactly this approach to introduce SLAng SLA information into the UML: We establish a correspondence between elements of extended UML syntax and types in the model of SLAng syntax; the relationships to the model of services then establish the meaning of these syntactic elements.

The Object Management Group (OMG) has standardise profiles for particular application areas. Using standard profiles ensures reusability for models, and for tools that operate on annotated model data. The QoS profile [8] is currently a proposed standard. It allows the modelling of QoS characteristics as classes, and QoS values as instances of these classes. QoS values can be associated with other model elements to indicate behaviour or requirements.

Rather than define a new profile to represent services and SLAs we have reused the QoS profile by defining a QoS catalogue for SLAng. Figure 5 shows the SLAng catalogue for ASP SLAs. The SLA terms are defined as QoS characteristics, rather than as first-class profile elements themselves. They nevertheless inherit the definitions provided by the semantic model.

The QoS profile allows QoS values, corresponding to defined QoS characteristics, to be attached to messages in a UML 2 communication diagram using one of three stereotypes: QoSContract, QoSRequired and QoSOffered. In all cases we state that the recipient of the message corresponds to the service associated with the SLA in the semantic model, and the sender corresponds to the service client. These elements are assumed to behave in accordance with the SLA terms. Where QoSOffered is defined together with either QoSRequired or QoSContract there is the opportunity for a tool to check the compatibility of SLAs according to the compatibility criteria defined in the previous section.

Figure 6 shows an example model including a SLAng ASP contract governing the interaction between a client and an online auction service. Constraints on the bidding operation are shown.

The ability to represent SLAs in UML is a necessary but insufficient condition to enable reasoning about QoS properties. It is also necessary to represent the impact of system components not directly governed by SLAs, and in some cases to support an analysis method for determining the emergent characteristics of the system. The 'Profile for Schedulability, Performance and Real-Time Specification' (henceforth 'real-time profile') [11] provides these facilities for systems in which performance is an issue and resource utilisation and scheduling have the greatest performance impact. It supports the derivation of models such as queuing networks or Petri nets. Other efforts are underway to extend UML to describe reliability properties [15].

[8] shows how QoS characteristics can be defined using the domain model of the real-time profile, allowing direct analysis. Because we use an alternative semantic model this approach is not directly available to us. However, it is perfectly possible to use our QoS characteristics simultaneously with real-time profile annotations with the interpretation that the real-time measures are an approximation of the service levels (future work will consider how this can be formalised). This provides a complete picture of the behaviour of the system in terms of performance, including SLA terms and sufficiently detailed to permit analysis.

UML is the design language of choice when adopting a Model Driven Architecture (MDA) [3] development strategy. In this methodology models of business knowledge are maintained separately from technical models and artifacts,

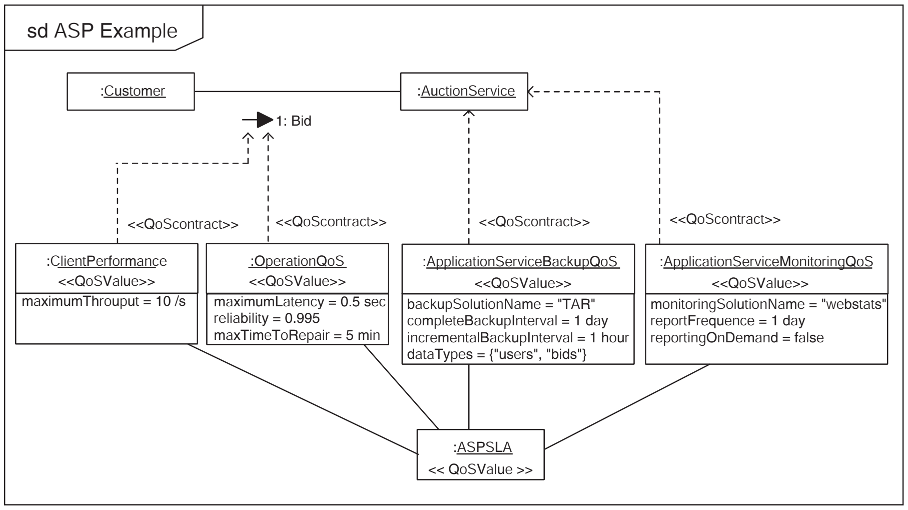

Figure 6. Example communication diagram including SLAng QoS values

enhancing their reusability across multiple platforms. The MDA relies on families of domain-specific languages, representing business and technical domains, and system properties such as QoS. Because these models must be reused in multiple contexts, an unambiguous definition of every language construct must be provided, and the use of domain models as described in this section is emerging as the preferred method. By providing a rich semantic and representation for SLA-aware systems our QoS catalogue enables the design and documentation of such systems in a manner appropriate to the MDA approach. Moreover, in [16] we show how MDA techniques can be used to enable automatic analysis, giving an example in which performance analysis proceeds from designs expressed using the real-time profile. The SLAng semantics therefore provides a reference for both analysis and implementation efforts relating to internal composition of SLAs.

6 Related Work

Our approach to defining the semantics of SLAng is derived from the work of the Precise UML group, who define an abstract syntax for UML and lend it semantics by associating a semantic domain. Our contribution to the approach is to demonstrate its application at a high level of abstraction, and including sophisticated quantitative properties. In this section we compare the semantic definition of SLAng to that offered by other SLA languages. We also briefly describe features of other languages lacking in SLAng, and their potential impact on the semantic definition.

The QuA project adopts the most rigorous approach to defining the semantics of QoS properties [18], although to our knowledge they have yet to define a concrete syntax for representing SLAs. According to their model, all QoS properties are related to the performance of a service, which supplies a set of operations. Input and output messages are causally related by operation invocations. Output messages are characterised by a set of variables. A set of error functions are defined over the difference vector between an observed output trace for a particular input trace, and the ideal trace as it would be observed were the service deployed on infinitely fast equipment operating without error. SLAs are defined using constraints on the values of error functions.

It is possible to see correspondences between our semantics and the QuA approach. In our case the service model defines the information available concerning service operation, and the OCL constraints provide a concrete representation of the error functions. Features of our semantics not obvious in the QuA approach are constraints independent of a service model assumption, such as the constraint that the service provider must be capable of providing the monitoring solution specified in an ASP SLA, and the ability to constrain client behaviour (in QuA terms, the input trace).

QML[4]defines a type system for SLAs, allowing the user to define their own dimension types. Whilst this makes the language highly extensible the meaning of the individual metrics in the context of the system is not formally established. Since exchange of SLAs between parties requires a common understanding of such metrics, this can be regarded as a serious deficit. QML does define a rigorous semantic for both its type system and its notion of SLA conformance however.

QML and WSOL [22] both provide type systems for SLAs, allowing the same SLA to be described in abstract and also instantiated with specific values. This provides more guidance to developers of new SLAs than our use of XML schemas. Because SLAs require common understanding of terms between parties, it is to be hoped that new types of SLA will be defined only infrequently. However, the generalisation relationships between SLAs are potentially helpful in structuring a family of SLA types. WSOL provides additional reuse facilities [14], including template instantiation and reuse of definitions.

WSOL [21] and UniFrame [1] SLA specifications both rely on the specification of measurements in external ontologies. These ontologies are structured natural language descriptions of measurements, including advice on how they should be taken and their interdependencies.

WSLA [5] is another XML based web services SLA language. All measurements are assumed to be provided by a web service encapsulating monitors. No constraints are placed on the implementation of such monitors, so a common understanding of their role remains external to the definition of the language.

WSLA provides the ability to create new metrics defined as functions over existing metrics. This is useful to formalise requirements expressed in terms of multiple QoS characteristics, without impacting on notions of compatibility of SLAs. The semantic for expressions over metrics is not formally defined, but no barrier prevents such a definition.

WSLA is an XML language, structured in such a way that monitoring clauses can be separated from contractual terms for distribution to a third party. We are interested in supporting third party monitoring schemes with SLAng. Precisely how such schemes will work will require careful modelling to inform the design of the metrics. However, this principle of syntactic separation is clearly useful.

WSOL provides an additional syntax to interrelate service offerings. Relationships indicate substitutability of SLAs in the event of a violation. Such facilities are clearly useful for specifying situations in which the offered QoS is permitted to vary, perhaps in synchrony with varying financial arrangements.

WSOL and WSLA allow the definition of management information, including financial terms associated with SLAs. These are not presented with a defined semantic (although WSOL claims the need for financial ontologies), but are clearly a desirable feature of SLA languages, and lacking in SLAng. Both languages also define management actions, including notifications in the event of SLA violations.

The CORBA Trading Object Service [9] allows the advertisement and selection of service offers based on constraints over typed properties. These properties can include QoS specifications, and generally can take any IDL type. Their semantics is not formally defined; neither are external ontologies specified. It is therefore up to the trader and its clients to agree an interpretation for the properties.

QuO [7] is a CORBA specific framework for QoS adaptation based on proxies. It includes a quality description language used for describing QoS states, adaptations and notifications. Properties in the language are defined to be the result of invoking instrumentation methods on remote objects. Like WSLA, no formal constraints are placed on the implementation of these methods.

7 Conclusions

We have presented the approach taken to defining the semantics of SLAng. The advantages of the approach are to reduce the ambiguity of the language, to explicitly document the semantics of the language and to support SLA composition. The model forms the basis for a common understanding of SLA terms between parties to an SLA, reducing the risk of outsourcing functionality to other organisations, and providing an unambiguous reference for negotiation and arbitration of agreements. These facilities are vital to the emerging Internet service business model.

We have introduced the concepts of inter- and intra- service composition of SLAs. Inter-service composition is supported by the notion of compatibility of SLAs, defined as a relationship between possible service behaviours according to the SLAng semantics. Intra-service composition relies on the semantics to provide meaning for models of services in UML diagrams, supporting analysis and implementation activities.

We have found the use of UML and OCL to be a natural and powerful means to define the semantics of our language. The activity of determining and describing the domain of the language, and the intended effect of the language, is reduced to a conventional object-oriented analysis, and results in documentation that is easily understood by the types of professional who will be required to reason about service composition and SLA terms. Clearly, for a language such as SLAng to be widely adopted in industry, it must be validated in wider practice, and consensus must be established as to its domain of applicability and expressiveness. The use of explicit models to describe the language should make such a consensus easier to establish.[1]

8 Future Work

SLAng is an evolving language. The related work section highlights desirable features of previous SLA languages that SLAng could benefit from incorporating. These include: Payments related to service violations; reuse features, including SLA templates, clause reuse and generalisation hierarchies; the specification of management actions,

[1]Thanks to Vladimir Tosic and Richard Staehli for guidance to related work.

performed in response to QoS state changes or SLA violations; the ability to define new parameter types in terms of existing types; the ability to distribute clauses to third parties without exposing sensitive details of the client and server; and the ability to define dynamic relationships between service levels, effectively defining the permissible states for a system capable of adapting its QoS behaviour.

These features would enhance the utility and usability of the language without modifying the semantic definition of the underlying parameters, although potentially requiring extensions to the existing semantics. Additional work will further realise the value of the semantic definition: By automating consistency checking of SLAs; by implementing application monitors that rely on the definition of SLAs to assess conformance to SLAng terms; by defining additional relationships between SLAs, and automating their comparison; and by investigating methodologies and designs for systems governed by SLAng SLAs.

References

[1] G. Brahnmath, R. R. Raje, A. Olson, M. Auguston, B. R. Bryant, and C. C. Burt. A Quality of Service Catalogue for Software Components. In *Southeastern Software Engineering Conference*, pages 513–520, Huntsville, Alabama, USA, April 2002.

[2] A. S. Evans and S. Kent. Meta-modelling semantics of UML: the pUML approach. In *2nd International Conference on the Unified Modeling Language*, volume 1723 of *Lecture Notes in Computer Science (LNCS)*, pages 140–155. Springer-Verlag, 1999.

[3] D. Frankel. *Model Driven Architecture - Applying MDA to Enterprise Computing*. OMG Press. Wiley Publishing, Inc., 2003.

[4] S. Frolund and J. Koistinen. QML: A Language for Quality of Service Specification. Technical Report TR-98-10, HP Laboratories, Palo Alto, California, USA, 1998.

[5] International Business Machines (IBM), Inc. *Web Service Level Agreement (WSLA) Language Specification*, January 2003.

[6] D. D. Lamanna, J. Skene, and W. Emmerich. Specification Language For Service Level Agreements. TAPAS Project Deliverable D2, University College London, March 2003.

[7] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken. Specifying and Measuring Quality of Service in Distributed Object Systems. In *1st International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 43–52, Kyoto, Japan, April 1998. IEEE Press.

[8] The Object Management Group (OMG). *Joint Submisison to the QoS for Fault Tolerance RFP, I-Logix, Open-IT, and THALES*, realtime/03-08-06 edition.

[9] The Object Management Group (OMG). *The CORBA Trading Service*, formal-97-04-01 edition, May 1997.

[10] The Object Management Group (OMG). *The Unified Modeling Language v1.5*, formal/2003-03-01 edition, March 2003.

[11] The Object Management Group (OMG). *UML Profile for Schedulability, Performance and Real-time Specification, Final Draft*, ptc/03-03-02 edition, March 2003.

[12] M. Oleneva and W. Beckmann. Application Hosting Requirements. TAPAS Project Deliverable D1, Adesso AG, Dortmund, September 2002.

[13] Organization for the Advancement of Structured Information Standards (OASIS). *Universal Description Discovery and Integration (UDDI)*, July 2002.

[14] B. Pagurek, V. Tosic, and K. Patel. WSOL - Web Service Offerings Language. In *Web Services, e-Business, and the Semantic Web — CAiSE 2002 Int. Workshop, WES 2002, Toronto, Canada*, Lecture Notes in Computer Science (LNCS), pages 57–67. Springer-Verlag, June 2002.

[15] G. N. Rodriguez, G. Roberts, W. Emmerich, and J. Skene. Reliability Support for the Model Driven Architecture. In *Workshop on Software Architectures for Dependable Systems (ICSE-WADS)*, Portland, Oregon, USA, May 2003. ICSE 2003.

[16] J. Skene and W. Emmerich. A Model-Driven Approach to Non-functional Analysis of Software Architectures. In *Proc. of the* 18^{th} *IEEE Conference on Automated Software Engineering, to appear*, Montreal, Canada, October 2003.

[17] J. Skene and D. D. Lamanna. Semantics of SLAng ASP SLAs, 2003. `http://www.cs.ucl.ac.uk/ staff/ j.skene/ slang`.

[18] R. Staehli, F. Eliassen, J. O. Aagedal, and G. Blair. Quality of Service Semantics for Component-Based Systems. In *2nd International Conference on Reflective and Adaptive Middleware Systems*, volume 2672 of *Lecture Notes in Computer Science (LNCS)*, pages 153–157. Springer-Verlag, June 2003.

[19] Sun Microsystems, Inc. *Enterprise Java-Beans (EJB) Specification v2.0*, August 2001.

[20] The Object Management Group (OMG). *UML 2.0 OCL 2nd revised submission*, ad/03-01-07 edition, January 2003.

[21] V. Tosic, B. Esfandiari, B. Pagurek, and K. Patel. On Requirements for Ontologies in Management of Web Services. In *Web Services, e-Business, and the Semantic Web — CAiSE 2002 Int. Workshop, WES 2002, Toronto, Canada*, Lecture Notes in Computer Science (LNCS), pages 237 – 247. Springer-Verlag, June 2002.

[22] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma. Management Applications of the Web Service Offerings Language (WSOL). In *15th International Conference on Advanced Information Systems Engineering (CAiSE), Velden, Austria*, volume 2681 of *Lecture Notes in Computer Science (LNCS)*, pages 468–484. Springer-Verlag, June 2003.

[23] The World Wide Web Consortium (W3C). *Web Services Description Language (WSDL) 1.1*.