Specification and Satisfaction of SLAs in Service Oriented Architectures

Stuart Anderson[†*] Antonio Grau[†*] Conrad Hughes[†*]

February 15, 2005

1 Introduction

This paper presents two essential aspects related to the specification of Quality of Service (QoS) in service oriented architectures. Firstly an overview is presented of the concepts and requirements that a language for the specification of QoS agreements (*Service Level Agreements*, or SLAs) must address, and how the WSLA language supports them. We concentrate on the components that an SLA language should have, and do not address other issues of the SLA life cycle such as negotiation, provisioning and monitoring. Secondly a work-in-progress is described which can provide statistical estimates of multiple dimensions of behaviour of composite services, a facility which will be essential when choosing services and adjusting workflows in order to satisfy SLA requirements.

Section 2 presents SLA language requirements, section 3 gives a brief introduction to WSLA and shows its use in the specification of tradeoffs between the QoS parameters (as well as evaluating WSLA against requirements), and section 4 describes the work on predicting behaviour of aggregate workflows.

2 Requirements for an SLA language

A service level agreement is an agreement between the provider of a service and their customer that defines the set of Quality of Service (QoS) guarantees and the obligations of the parties. An SLA ensures that both parties understand the service to be provided and that it will conform to certain QoS requirements. An SLA language usually defines the following aspects of the SLA: the purpose; the validity period; the parties involved and their respective rôles; the scope (i.e. the service operations covered in the agreement); the set of service level indicators (or QoS parameters), and their associated metrics, over which the QoS levels can be measured; the set of *Service Level Objectives* (SLOs) or guarantees to be fulfilled; and the penalties and actions to be undertaken when these guarantees are not satisfied. QoS parameters refer to observable properties relating to nonfunctional aspects of the service, e.g. availability, performance and reliability. SLOs are constraints defined over the values of those parameters that may be dependent on a precondition.

An SLA language should fulfil the following desirable properties and requirements:

Precision

An SLA must be specified in a precise and unambiguous way. This is essential in order that the service provider and customer clearly understand the offered quality of service, and so that no misunderstandings arise later. Precision is also essential in order to help automate the SLA management process, such as the automatic negotiation, provisioning and monitoring of the SLA. Precision must cover all the elements of an SLA specification, including the definitions of the QoS parameters, their metrics and the agreed SLOs. For example, dependability concepts such as availability and reliability and how these are measured should be well understood by both parties. Issues such as when the SLA should be checked for compli-

[†]School of Informatics, University of Edinburgh

[*]These authors acknowledge the support of EPSRC award no. GR/S04642/01, *Dependable, Service-centric Grid Computing*

ance, over which sample of data and from where the measurements should be taken must be unambiguously specified. Offering an average availability of 99.9% does not entirely assure that the service will be available on 99.9% of client accesses, as availability may depend on the time of access. Also, response time measurements made by the service client are likely to be entirely different to those taken on the network or by the service provider. An average response time may be different depending on the averaging window, for example, five minutes or one hour. The correct definition of QoS parameters corresponds to the establishment of an ontology between a service provider and the client [4]. This ontology can be a definition of terms and the semantics of the relationships between them. The definitions could then be available at referenced URIs on the "Semantic Web" as suggested by the OWL-S approach [7], so that the provider and the client have a means of sharing definitions of the terms and concepts used in the SLA.

Flexibility

Web services can be extremely diverse. The QoS parameters and SLOs referenced in an SLA are often specific to the particular web service under negotiation. For example, the SLA of a service providing streaming multimedia would contain QoS parameters such as bandwidth limits and quality of the streaming media, while the SLA of a e-banking service would have guarantees on the security of the transmitted information. The nature of interactions between the service customer and provider is likely to be diverse too — for instance, single request-reply interactions and interactions that can last for several hours. This diversity demands SLA languages that can be extended to fit the needs of the specific web service domain. Extensibility should cover the definition of new QoS parameters and new metrics as well as other language terms necessary for the definition of SLOs specific to the web service. XML-based languages are good candidates, not only because of their ease of extension but also because they are extensively used in other web service-related specifications such as WSDL and SOAP [3].

Definition of qualitative parameters

Existing SLA languages refer to observable QoS parameters that can be measured and therefore have an associated metric. However, they do not refer to qualitative unmeasurable parameters such as the chosen security model or supported standards. The reason for excluding these parameters is probably due to the fact that one of the main objectives of formalising SLAs is the automation of the SLA management. Including unmeasurable parameters in the SLAs would break the homogeneity of how the parameters are handled by some of the components of the SLA management framework, such as the provisioning of resources and the measuring and monitoring of the parameters. As SLAs are usually not stand-alone documents but embedded in legal contracts, a possible option for consideration of qualitative parameters would be their direct inclusion in the contract, leaving intact the SLA document. However, this would prevent the use of such qualitative parameters in SLOs, for instance for the specification of tradeoffs between measurable and unmeasurable parameters. It remains an issue how unmeasurable parameters can be expressed in an SLA and how the client can be confident that the service fulfils them.

Definition of relationships between the QoS parameters

An SLA language should allow the specification of guarantees that relate different QoS parameters by providing a set of logical expression constructs. This is essential in order to specify tradeoffs between the parameters. For instance, a client could be interested in making tradeoffs between response time and availability, or between size and quality of the data. Another interesting possibility for relating QoS parameters is to give them a weight or level of importance, allowing customers to establish their priorities. This could prove useful to the SLA provisioning system if excess resources are available after all SLOs are met.

Definition of effects

The SLA must specify the consequences of not meeting an SLO. This includes the specification of a list of actions to perform such as sending notifi-

cations to the parties as well as the penalties for the provider. Penalties would incur a certain cost for the provider and should also include the option of terminating the contract in the case that the service levels are unacceptable. Alternatively, meeting an SLO could also generate a reward for the service provider. An SLA should also facilitate notification of the parties when the service level is near to violating an SLO, so the provider has the opportunity to assign additional resources to guarantee the SLO, and the client has advance warning of possible breaches.

Definition of endogenous and exogenous parameters

The level of service offered for some QoS parameters can be affected by other parameters that are not within the control of the service provider. This is the case, for example, for web services that are accessed through the public Internet, where the service provider has no control over the network properties that are provided by the client's ISP. An SLA must specify under which conditions or values of the exogenous parameters, the SLOs are to be guaranteed by the service provider. The SLA can define different guarantee levels as a function of the values of the exogenous parameters. This affords the service provider protection from liability in case of low QoS levels resulting from circumstances outside their control. Although the client is only interested in the end-to-end QoS, it is important for the provider to identify exogenous components in the parameters and the effect that these components have on the end-to-end values so guarantees can be specified accordingly, e.g. end-to-end availability as a function of network availability.

Exceptions

An SLA must not only specify the level objectives in normal conditions but also in exceptional circumstances. Exceptions can be provoked by many diverse events and can have effects of diverse magnitude on the terms of the SLA. For example, a server attack may produce a denial of service that entails the suspension of all the SLOs established in the SLA, while maintenance operations at the provider side may just affect the levels of some parameters. Exceptions can be caused by the service provider (e.g. a server-side hardware failure), the client, an external provider (e.g. the client ISP), or complete externalities such as natural catastrophes or war. Providers must specify the actions to take when an exception for which they are responsible happens (e.g. recovery mechanisms and notifications in case of failures), and also clearly identify the exceptions that are not under their control, so that no liability problems arise. Exceptions can be classified as follows:

Type	**Examples**
Failures	Hardware failure, software bugs/flaws, telecommunication failure, measurement or monitoring failure
Service maintenance	Hardware upgrades, software upgrades, backups
Network properties (not responsibility of the provider)	Low network availability, high network error rate, ...
Denial of service	Client negligence/ wilful misconduct, network/Internet security breaches (floods, hacks and attacks), Acts of God (fire, earthquakes, ...) and circumstances beyond reasonable control (war, terrorism, strike, ...)

Table 1: Exceptions

Composition

A service provider can make use of other providers to supply some part of the service functionality. In this case, the values of the SLA agreed by the provider and the end customer will depend on the values of the SLAs agreed by the provider and their suppliers. Composite web services require therefore a precise understanding of how the individual QoS properties of a component contribute to the overall QoS properties of the composite. This will depend on the structure of the composition and the nature of the QoS parameters. A precise QoS ontology must therefore encompass the definition of how QoS

parameters behave in composition.

Definition of parties

The SLA must include information about the parties involved in SLA management and their responsibilities. Apart from the signatory parties, i.e. the service provider and the service client, the SLA should contain information about third parties, i.e. parties supporting part of the management functionality, such as measurement and monitoring.

SLA management

The components of an SLA must be defined in the context of an SLA management framework. SLA management includes tasks such as the negotiation, creation, provisioning, monitoring and compliance checking of the SLAs. The creation of the SLA may be as simple as the customer selecting one of the pre-specified SLAs offered by the provider, or by customisation via a negotiation process. Different service clients may have different requirements and preferences regarding QoS levels, so the service provider must define several offers with different service levels. Pre-specified, fixed and negotiable information about the QoS values offered by the provider can be captured by SLA templates. The structure of an SLA template may be the same as that of an SLA but partially completed and containing an additional section where constraints on the values of the unfilled fields are defined. The constraints must be followed by the customer when negotiating the SLA.

3 SLA specification in WSLA

WSLA (Web Service Level Agreement) is a formal language to define service level agreements that has been developed by IBM [2, 5, 6]. It is based on XML, and an XML schema has been defined for its syntax. The language is extensible and allows derivation of new domain-specific elements from existing language elements. This can easily be done by making use of the ability to create derived types using XML schemas. A WSLA specification is structured in three main sections: the parties, the service description and the obligations.

Parties specification

This section describes the parties involved in the management of the web service. A party can be a signatory party or a supporting party. Signatory parties are either the service provider or the service customer. The information for a party includes the name, contact, and the definition of the interfaces of actions that it offers. The interface definitions are specified in WSDL and describe operations that a party can perform when invoked by the occurrence of an event, e.g. a notification when a guarantee is violated. The information for a supporting party includes additionally the sponsor of the party, either the provider or the customer, and the supporting rôle that it assumes, for example, measurement service or condition evaluation service.

Service definition

A service definition specifies the information needed about the service to define the agreed service level guarantees, i.e. the operations offered by the service provider, the QoS parameters to be considered for each of these operations and the metrics used for measuring these parameters. In WSLA, QoS parameters are called SLA parameters and must be measurable. Each SLA parameter has a name, a type and a unit. In addition, a parameter refers to one metric that describes how the value of the parameter is measured or computed. A metric can be either a resource metric or a metric composed from other metrics. In the former case, this is specified by a measurement directive; in the latter case, this is specified by a function. Examples of resource metrics are system uptime, service outage period and number of service invocations. Examples of composite metrics are maximum response time and average availability of the service. A measurement directive describes how the values are retrieved from the resources. Examples of measurement directives are the URI of a computer program, a command for invoking scripts and database queries. A function represents a measurement formula that specifies how the composite metric is computed. Examples of functions are mean, median, sum, minimum, maximum and time series constructors. For every function a schedule is described. The schedule specifies the time inter-

vals during which the function is executed to compute the metric. The time intervals are defined by means of the start time, duration and frequency. Moreover, SLA parameters can include information about the party that provides the values and the parties that receive them, either by active update (*push*) or by providing access to the parties to retrieve them (*pull*).

Each operation also contains a reference to the service that contains the operation definition to which the WSLA operation refers. This reference depends on the way in which the service is described. In the context of web services, this is described in a WSDL specification. The reference to a WSDL-defined service includes the name of the WSDL file, the kind of binding, i.e. the transport encoding for the SOAP messages, and the operation name as specified in the WSDL file.

Obligations

This section defines the guarantees and constraints that are imposed on the SLA parameters. Two kinds of obligation arise: *Service Level Objectives* (SLOs) and *action guarantees*.

SLOs are restrictions on the values of the SLA parameters in a given period of time. An SLO specifies the party that is responsible for delivering what is imposed in this guarantee, a validity period defining when the restriction is applicable and a logical expression defining the assertion to be tested. In addition, an SLO may contain information about when the assertion should be evaluated. This can be done by defining either an evaluation event expressing when the assertion should be evaluated (for example, every time a new value for an SLA parameter included in the assertion is available), or a schedule according to which the assertion is evaluated.

An action guarantee specifies the actions that must be performed by the parties in the case that a given precondition is met. A typical predicate in a precondition is the violation predicate that expresses whether an SLO has been violated. The definition of an action guarantee contains the name of the party in charge of this guarantee, a logical expression defining the precondition, an evaluation event or schedule describing when the precondition should be evaluated, and the actions to be invoked at particular parties in the case that the precondition holds. The interface of the actions is defined in the parties' specification section of the WSLA document and examples of actions are the notification of events, problem reports and payment of penalties and premiums. An action guarantee may also include an execution modality that expresses the frequency with which the action must be executed depending on the value of the precondition, for instance always, on entering a condition, and on entering and leaving a condition.

Tradeoffs specification in WSLA

A main feature of WSLA is that it allows the use of logical expressions in the specification of the SLOs and action guarantees. WSLA provides logical expressions that follow first order logic including logic operators and predicates but not quantifiers. The logic operators are `And`, `Or`, `Not` and `Implies`. Tradeoffs between QoS parameters (i.e. dependencies between the values of the parameters) can be specified using the `Implies` operator. Predicates are functions that return true or false. The predicates needed may vary depending on the SLA parameters and metrics defined for a particular domain. WSLA allows the definition of new predicates to fit the needs of the specific application domain by defining the predicate type as an abstract type that can be extended in the XML schema. WSLA also provides a set of built-in predicates: `Violation`, `Greater`, `Less`, `Equal`, `GreaterEqual`, `LessEqual`, `True` and `False`. As an example to illustrate the concepts, suppose we have a web service providing text-to-speech audio streaming. Customers send requests in the form of texts to be synthesised. The provider generates the corresponding audio signal and transmits it to the customers over the Internet in a compressed stream format. A key requirement in the quality of the service is the continuous playback of the audio stream by the client, i.e. elimination of the gaps or silences which occur (usually as a result of network congestion) when the rate of audio delivery drops below real time for long enough to exhaust the client's buffers. Assuming an idealistic case in which the provider promises that no gaps in transmission will occur, the SLO within the SLA document that guarantees this requirement could be specified as follows:

```
<Obligations>
  <ServiceLevelObjective name = "GapOccurrence">
    <Obliged>TTSProvider</Obliged>
    <Validity>
      <Start>2005-01-01T09:00:00-00:00</Start>
      <End>2005-01-31T09:00:00-00:00</End>
    </Validity>
    <Expression>
      <Predicate xsi:type = "Equal">
        <SLAParameter>BitsDelay</SLAParameter>
        <Value>0</Value>
      </Predicate>
    </Expression>
    <EvaluationEvent>NewValue</EvaluationEvent>
  </ServiceLevelObjective>
  ...
  </Obligations>
```

The complete WSLA specification can be found in appendix A. This SLO specifies first the obliged party in charge of assuring the level of service, and then the validity period. Assuming we have an SLA parameter, `BitsDelay`, which represents the difference of bits between the expected and the actual bits received by the client, the SLO asserts that this difference must be equal to zero, i.e. there is no gap during the transmission.

A more realistic example is the assurance by the provider of no gaps only after a percentage of the audio has been transmitted. This helps the provider to first negotiate and calculate the appropriate encoded content for the best audio quality depending on the customer's real connection bandwidth. The SLO specifying this guarantee could be defined as follows:

```
<ServiceLevelObjective name = "GapOccurrence">
 <Obliged>TTSProvider</Obliged>
 <Validity>
  <Start>2005-01-31T09:00:00-00:00</Start>
  <End>2005-01-31T9:00:00-00:00</End>
 </Validity>
 <Expression>
  <Implies>
   <Expression>
    <Predicate xsi:type="Greater">
     <SLAParameter>StreamTransmittedPercentage
     </SLAParameter>
     <Value>10</Value> <!-- 10% -->
    </Predicate>
   </Expression>
   <Expression>
    <Predicate xsi:type = "Equal">
     <SLAParameter>BitsDelay</SLAParameter>
     <Value>0</Value>
    </Predicate>
   </Expression>
  </Implies>
 </Expression>
 <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

Given an SLA parameter, `StreamTransmitted-Percentage`, which represents the percentage of the audio stream that has been transmitted to the client, this SLO asserts that there are no gaps after 10% of the audio stream has been transmitted. Note that as WSLA does not allow the use of variables with values that can be computed nor retrieved from a resource, the percentage of the audio stream has been defined as an SLA parameter.

WSLA allows the specifications of actions to be executed in the parties when a guarantee or SLO is violated. In our example, the following action guarantee specifies the execution of a notification to the provider indicating that a gap in the transmission has violated the SLO:

```
<ActionGuarantee name = "GapOcurrenceGuarantee">
 <Obliged>AuditingCompany</Obliged>
 <Expression>
  <Predicate xsi:type = "Violation">
   <ServiceLevelObjective>GapOccurrence
   </ServiceLevelObjective>
  </Predicate>
 </Expression>
 <EvaluationEvent>NewValue</EvaluationEvent>
 <QualifiedAction>
  <Party>TTSProvider</Party>
  <Action actionName = "notification"
  xsi:type = "Notification">
   <NotificationType>Violation</NotificationType>
   <CausingGuarantee>GapOccurrenceGuarantee
   </CausingGuarantee>
   <SLAParameter>BitsDelay</SLAParameter>
  </Action>
 </QualifiedAction>
 <ExecutionModality>Always</ExecutionModality>
</ActionGuarantee>
```

Analysis of WSLA against requirements

WSLA is one of the most suitable QoS languages for achieving the requirements discussed in the previous section. In particular, the language presents the following strengths:

- It allows the precise definition of the QoS parameters and their metrics, i.e. how and under which schedule the measurement values are retrieved from the resources and how they can

be combined by functions to obtain meaningful parameters such as the mean and the sum of a series of values.

- It is flexible. The XML schema specification allows the definition of new functions, predicates and measurement directives by deriving them from their abstract types. Furthermore, the inherent extensibility of XML makes the definition of new elements straightforward.
- The language provides logical expressions to relate QoS parameters and specify tradeoffs between them.
- It allows the specification of actions to be performed by the parties when some QoS values or service level objectives are not met. However, it does not specify any penalty/reward policy.
- It gives full support to the definition of parties and their responsibilities.

Nevertheless, some of the requirements are not supported:

- It does not provide any specific construct for the specification of penalties/rewards.
- It does not allow the specification of logical or non-quantifiable parameters.
- It does not support the possibility of giving priorities or weight to the QoS parameters.
- Although the logical expressions provided by the language allow the specification of SLOs in different conditions (normal, maintenance, failure/exception cases, etc.), the language does not support the explicit definition of these conditions. Different cases could be specified by the definition of variables, or predicates on the values of the variables, that could be instantiated during the service execution. However, WSLA only allows the declaration and definition of QoS parameters.

These issues need to be addressed in order to obtain more complete SLA specifications. Many of the language concepts are being currently used and extended to cover grid services in the context of the WS-Agreement standard from the Global Grid Forum, where the WSLA authors are active members [1].

4 Behaviour of composite services

The DIGS project seeks to automate as much as possible the composition of services in pursuit of QoS requirements through choice of appropriate components (*diversity*) and modification of workflow (*structure*). This can only be done if the effect of differing service choices and workflow modifications can be understood. Consequently we are developing a tool which predicts the aggregate behaviour of a composition in terms of its workflow structure and the known behaviour of its individual services. Once such predictions are possible, candidate compositions can be evaluated against client SLA demands and duly accepted or disregarded. Our solution breaks the problem into the following four elements:

Properties

These simply correspond to the QoS parameters in terms of which SLOs are written — things like time to complete, availability, accuracy, peak bandwidth usage, perhaps even cost.

Workflow

While so sophisticated a representation of composition structure as BPEL or a scripting language may not be necessary, at least the basics of parallel, conditional and serial operation need to be modelled. Parallel operations range from simple variations like waiting for the first or all responses through to more complicated ones such as voting or waiting for a certain number of responses to satisfy a condition (such as succeeding rather than failing).

Behaviours

Each property will respond to different workflow operations in different ways, for example time to complete is additive under serial composition, maximising under all-of parallelism (i.e. the time to complete of an all-of parallel composition will be the maximum of its components' times), minimising under one-of parallelism; network bandwidth is maximising under serial composition and additive for all forms of parallelism; probability of success is multiplicative in series and "complement-

multiplicative" $1 - \prod_i (1 - p_i^{\text{success}})$ in parallel; etc. Considered in this light, a "property" becomes a mapping from workflow operations to (numerical or logical) operations on values.

Values

Workflow-induced behaviours of properties must be executed on actual *values* if useful predictions are to be made; the choice of underlying representation for these values will be critical, as (for example) it will be very difficult to say anything about the slowest 10% of a class of operation if all that's known about them is their average performance. Possible representations for numeric quantities include expected value; minimum-maximum range; mean and variance of best normally distributed approximation; probability density function; Markov model of states; etc. Even within an individual problem it will almost certainly make sense to use different value representations for different properties. At the moment the main value representation used is an approximation of the variable's statistical distribution using a variable number of uniform segments. This allows for moderately accurate representations of a number of situations, including delta functions, bimodal variables, etc.

Processes

A process is a bundle of properties representing the known behaviour of an actual workflow element - usually a service call. These are the objects with which aggregation computations are made. It is necessary to process all properties at once in bundles because, once elements such as conditional operations and parallel-first-response are admitted to the workflow, individual properties start to affect the probabilities with which other processes are executed, hence affecting *all* properties for dependent elements of the workflow.

Examples

In reference to the text-to-speech service, it might be reasonable to expect the difference between the received and expected quantity of data to be normally distributed. The likelihood of a gap in audio manifesting on the recipient's machine will then be determined by the size of the recipient's buffer: if the received-versus-expected difference exceeds buffer capacity then the buffer will be empty and a gap will occur — the probability of this event can be exactly determined once the difference's distribution (normal or otherwise) is known. If two sufficiently similar text-to-speech sources can be sourced then the likelihood of both buffers emptying simultaneously will be the product of the each individual service's gap likelihoods. Obviously this use of correlated parallel data sources will be significantly more reliable than a single source. However, it clearly also uses twice as much network bandwidth, which in a bandwidth-constrained situation could *cause* the very situation it seeks to mitigate.

Similarly, as reported in last year's DIRC conference, making queries simultaneously against several diversely implemented databases can substantially improve response times — you can forget about the other queries as soon as you have any result that's good enough. This has been demonstrated in practice, but also follows logically from the mathematical fact that the minimum of two random variables will always have a lower mean than either variable individually (as long as their distributions overlap — if one variable is *always* lower than the other this strategy cannot have any useful effect). The caveat here is that in a commercial situation this improved performance will come at the — likely financial — cost of always making two queries instead of one.

A further example: simply waiting for a small period for an answer from one service then (absent said answer) making the same request of another service will decrease the rate of failures (*both* services must fail in order to break this strategy), probably also slightly improve response times (the balance here will depend on the failure rates), and — a tradeoff again — slightly increase costs. Calling the more expensive service first will have a higher average cost than starting with the less expensive service, but it might be expected that this more expensive service will also offer a better response time: another choice for the user to make.

As can be seen in all of the above examples, there exist situations where using multiple services in place of one can improve certain properties of the system, usually at a tradeoff against other properties — a tradeoff heavily influenced by the manner in which the services are combined. The tool we are developing here will greatly facilitate the analysis of such situations.

Status

The aggregate prediction tool is only now approaching useful functionality, as a result of which no analysis has yet been done of how it compares with (for example) simulation, or whether the choice of distribution function as underlying value gives significantly better results than the alternatives in practical situations. This analysis will be undertaken over the coming months.

5 Conclusion

This paper has presented the requirements for a precise SLA specification language and how the WSLA language meets them. Although the language supports many of the requirements, it still needs to be extended in order to support the specification of richer service level objectives, such as the definition of logical parameters and exceptions. We expect that combining SLAs with the ability to predict behaviour of composite workflows should offer substantial gains in efficiency of resource usage as well as allowing service providers and consumers to start thinking about their compositions at a much higher level than before.

References

[1] A. Andrieux et al. Web Services Agreement Specification (WS-Agreement). Draft. Global Grid Forum, August 2004.

[2] A. Dan et al. Web Services on Demand: WSLA-driven Automated Management. *IBM Systems Journal*, 43(1):136–158, 2004.

[3] G. Dobson. Quality of Service in Service-Oriented Architectures, 2004. http://digs.sourceforge.net/papers/qos.html.

[4] G. Dobson and R. Lock. Developing an Ontology for QoS, 2005.

[5] A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *IBM Research Report*, May 2002.

[6] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web Service Level Agreement (WSLA) Language Specification, Version 1.0. Technical report, IBM Corporation, January 2003.

[7] D. Martin et al. Bringing Semantics to Web Services: The OWL-S Approach. In *SWSWPC04*, 2004.

A Example SLA: text-to-speech

```
<?xml version = "1.0" encoding = "UTF-8"?>

<!-- TTS sample -->

<SLA xmlns = "http://www.ibm.com/wsla"
 xmlns:xsi = http://www.w3.org/2001/XMLSchema-instance
 xsi:schemaLocation = "file:///C:/WSLA093.xsd"
 name = "TTSSample">

<!-- Definition of the Involved Parties, the signatory
 parties as well as the supporting ones -->

<Parties>
 <ServiceProvider name = "TTSProvider">
  <Contact>
   <Street>NeSC</Street> <City>Edinburgh, UK</City>
  </Contact>
  <Action xsi:type = "WSDLSOAPOperationDescriptionType"
   name = "notification" partyName = "TTSProvider">
   <WSDLFile>Notification.wsdl</WSDLFile>
   <SOAPBindingName>SOAPNotificationBinding
     </SOAPBindingName>
   <SOAPOperationName>Notify</SOAPOperationName>
  </Action>
 </ServiceProvider>

 <ServiceConsumer name = "TTSCustomer">
  <Contact>
   <Street>JCMB, King's Buildings</Street>
   <City>Edinburgh, UK</City>
  </Contact>
  <Action xsi:type = "WSDLSOAPOperationDescriptionType"
   name = "notification" partyName = "TTSCustomer">
   <WSDLFile>Notification.wsdl</WSDLFile>
   <SOAPBindingName>SOAPNotificationBinding
     </SOAPBindingName>
   <SOAPOperationName>Notify</SOAPOperationName>
  </Action>
 </ServiceConsumer>

 <SupportingParty name = "AuditingCompany"
  role = "MeasurementService">
  <Contact>
   <Street>BP 1</Street>
   <City>Edinburgh, UK</City>
  </Contact>
  <Sponsor>TTSProvider</Sponsor>
 </SupportingParty>
```

```
</Parties>

<!-- The definition of the service in terms of the
     service parameters and their measurement. -->

<ServiceDefinition name = "TTSService">

 <Schedule name="MainSchedule">
  <Period>
   <Start>2005-01-01T09:00</Start>
   <End>2005-01-31T09:00</End>
  </Period>
  <Interval>
   <Seconds>5</Seconds>
  </Interval>
 </Schedule>

 <Operation name = "GetAudioStream"
  xsi:type = "WSDLSOAPOperationDescriptionType">

  <SLAParameter name = "BitsDelay"
   type = "long" unit = "kbits">
   <Metric>BitsDelayMetric</Metric>
  </SLAParameter>

  <Metric name = "BitsDelayMetric"
   type = "long" unit = "kbits">
   <Source>AuditingCompany</Source>
   <Function xsi:type="wsla:Minus"
     resultType = "long" unit = "kbits">
    <Operand>
     <Schedule>MainSchedule</Schedule>
     <Metric>IdealBitsReceived</Metric>
    </Operand>
    <Operand>
     <Schedule>MainSchedule</Schedule>
     <Metric>BitsReceived</Metric>
    </Operand>
   </Function>
  </Metric>

  <Metric name = "BitsReceived"
   type = "long" unit = "kbits">
   <Source>AuditingCompany</Source>
   <MeasurementDirective xsi:type =
    "wsla:Counter" resultType = "long">
    <MeasurementURI>
     http://nesc.ed.ac.uk/TTS/ipKbitsIn
    </MeasurementURI>
   </MeasurementDirective>
  </Metric>

  <Metric name = "IdealBitsReceived"
   type = "long" unit = "kbits">
   <Source>AuditingCompany</Source>
   <MeasurementDirective xsi:type = "wsla:Counter"
    resultType = "long">
    <MeasurementURI>
     http://nesc.ed.ac.uk/TTS/IdealKbitsIn
    </MeasurementURI>
   </MeasurementDirective>
  </Metric>

  <WSDLFile>TTSService.wsdl</WSDLFile>
  <SOAPBindingName>
   SOAPNotificationBinding
  </SOAPBindingName>
  <SOAPOperationName>getAudioStream</SOAPOperationName>
 </Operation>
</ServiceDefinition>

<!-- The TTSProvider assures that there
 are no gaps during the transmission -->

<Obligations>

 <ServiceLevelObjective name = "GapOccurrence">
  <Obliged>TTSProvider</Obliged>
  <Validity>
   <Start>2005-01-01T09:00:00-00:00</Start>
   <End>2005-01-31T09:00:00-00:00</End>
  </Validity>
  <Expression>
   <Predicate xsi:type = "Equal">
    <SLAParameter>BitsDelay</SLAParameter>
    <Value>0</Value>
   </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
 </ServiceLevelObjective>

 <ActionGuarantee name = "GapOcurrenceGuarantee">
  <Obliged>AuditingCompany</Obliged>
  <Expression>
   <Predicate xsi:type = "Violation">
    <ServiceLevelObjective>
     GapOccurrence
    </ServiceLevelObjective>
   </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
  <QualifiedAction>
   <Party>TTSProvider</Party>
   <Action actionName = "notification"
    xsi:type = "Notification">
    <NotificationType>Violation</NotificationType>
    <CausingGuarantee>GapOccurrenceGuarantee
    </CausingGuarantee>
    <SLAParameter>BitsDelay</SLAParameter>
   </Action>
  </QualifiedAction>
  <QualifiedAction>
   <Party>TTSCustomer</Party>
   <Action actionName = "notification"
    xsi:type = "Notification">
    <NotificationType>Violation</NotificationType>
    <CausingGuarantee>GapOccurrenceGuarantee
    </CausingGuarantee>
    <SLAParameter>BitsDelay</SLAParameter>
   </Action>
  </QualifiedAction>
  <ExecutionModality>Always</ExecutionModality>
 </ActionGuarantee>
</Obligations>
</SLA>
```