Web Service Level Agreement (WSLA) Language Specification

Version: 1.0
Revision: wsla-2003/01/28

Authors:
Heiko Ludwig, IBM T.J. Watson Research Center
Alexander Keller, IBM T.J. Watson Research Center
Asit Dan, IBM T.J. Watson Research Center
Richard P. King, IBM T.J. Watson Research Center
Richard Franck, IBM Software Group

Abstract

This document describes the specification language for service level agreements for Web Services, the Web Service Level Agreement (WSLA) language. WSLAs are agreements between a service provider and a customer and as such define the obligations of the parties involved. Primarily, this is the obligation of a service provider to perform a service according to agreed-upon guarantees for IT-level service parameters (such as availability, response time and throughput) for Web Services.

An SLA also specifies the measures to be taken in case of deviation and failure to meet the asserted service guarantees, for example, a notification of the service customer. The assertions of the service provider are based on a detailed definition of service parameters including the algorithms -- how basic metrics should be measured in systems and how they are aggregated into composite metrics and SLA parameters. In addition, a WSLA can express the operations of monitoring and managing the service. This may include third parties (such as Management Service Providers) that contribute to the measurement of metrics, supervision of guarantees or even the management of deviations of service guarantees. These multi-party constellations necessitate the definition of the interactions among the parties supervising the WSLA.

However, a WSLA only covers the agreed common view of a service between the parties involved. To actually act as a participant in a WSLA, parties have various degrees of freedom to define an implementation policy for a service and its supervision. Typically, the obligations of a WSLA must be translated into system-level configuration information, which can be proprietary to each party involved.

Table of Content

1. Introduction

1.1. Summary

This document describes the Web Service Level Agreement language (WSLA language). A WSLA document (referred to as a WSLA) defines assertions of a service provider to perform a service according to agreed guarantees for *IT-level and business process-level service parameters* such as response time and throughput, and measures to be taken in case of deviation and failure to meet the asserted service guarantees, for example, a notification of the service customer. The assertions of the service provider are based on a detailed definition of the service parameters including how basic metrics are to be measured in systems and how they are aggregated into composite metrics. In addition, a WSLA expresses which party monitors the service, third parties that contribute to the measurement of metrics, supervision of guarantees or even the management of deviations of service guarantees. Interactions among the parties supervising the WSLA are also defined.

The WSLA language is based on XML; it is defined as an XML schema.

WSLA can be used by both service provider and service customer to configure their respective systems to provide and supervise their service. This process is called deployment. Each organization uses its own independent deployment function that interprets the WSLA and takes appropriate action. The deployment step includes creation and parameterization of the relevant service implementing systems and WSLA supervising services. Also, parts of the WSLA (or derived information) can be passed on to third parties that support the WSLA's supervision. After deployment, the WSLA can be enacted by supervising services.

An important aspect of WSLA is its capability to deal with specifics of particular domains and technologies. The language is extensible to include specific types of operation descriptions, (e.g., using WSDL to describe a Web services operation), measurement directive types for specific systems, special functions to compose aggregate metrics and predicates to evaluate specific metrics. The extension mechanism makes use of the ability to create derived types using XML schema. By design, the core of the WSLA language is very compact. To be of immediate use, the WSLA language encompasses a set of *standard extensions* that allow WSLA authors to define complete agreements that relate to Web services and include guarantees for response time, throughput and other common metrics. For other technical fields, for example, online storage based on the Network Attached Storage (NAS) technology, specific extensions may be defined by interested organizations.

1.2. Audience

This specification is targeted at three groups: ***WSLA authors*** use it as a reference to compose their specific agreements. It is also intended at ***implementers*** of deployment tools and other systems that directly interpret WSLA documents or its fragments, such as third party measurement systems. A third target audience are ***authors of domain specific extensions*** of the language. System vendors or service providers can extend the language and capture the specific information that is needed in their domain.

This specification uses the XML Schema language [1] to define the language constructs of WSLA. A good working knowledge of the XML Schema language is a prerequisite of reading beyond section 1. The document also relates to the Web services stack and uses WSDL [5] for the definition of standard services interfaces. Familiarity with Web services in general and the WSDL in particular helps to understand the respective sections.

1.3. Design Objectives

The WSLA language is designed to capture service level agreements in a formal way to enable automatic configuration of both the service implementing system of the service providing organization as well as the system that is used to supervise the agreed quality of service.

To facilitate automatic configuration the WSLA comprises:

- A description of the ***parties***, their roles (provider, customer, third parties) and the action interfaces they expose to the other parties of the contract.
- A detailed specification of the ***service level parameters*** (SLA parameters) guarantees are applied to. SLA parameters are specified by metrics. Metrics define ***how to measure*** an item, in the case of Resource metrics, or ***how to aggregate*** metrics to composite metrics. A metric description also includes ***which party*** is in charge of measuring and aggregating and how the metrics can be retrieved.
- A representation of the parties' obligations. ***Service level objectives*** contain a formal expression of the guaranteed condition of a service in a given period. ***Action guarantees*** represent promises of parties to do something, for example, to send a notification in case the guarantees are not met.

The specification of the service level parameters and metrics represents the ***common understanding*** of the parties. The implementation of the service within the service provider can be more complex in a way that the view in the WSLA is an abstraction of the real underlying system.

The language must be extensible to capture services from particular domains using specific types of service parameters. The WSLA language uses the mechanism of type derivation by XML schema to introduce new types (i.e. syntactical elements with semantics derived from the high-level type) to describe domain specific concepts.

The language should be immediately useful to SLA authors. The standard extensions are part of the WSLA specification. While the core syntax is deliberately kept slim, the standard extensions provide a set of type definitions that cover many usual concepts related to a Web Service SLA.

In order to protect the confidentiality of the service customer, an SLA must be decomposable into the configuration information that is needed for third parties to perform their role in the SLA supervision without having access to the complete SLA.

1.4. Model of the WSLA Environment

1.4.1. WSLA and Service Definitions

A WSLA agreement complements a service definition. While a service description, for example, using WSDL defines the service interface relationship between a service and its using application, the WSLA defines the agreed performance characteristics and the way to evaluate and measure them. WSLA complements service description languages. Service descriptions are input to the design and implementation of the service system and the client application using its service. The WSLA provides input to the measurement and management system of an organization that checks and manages an organization's compliance with a WSLA. The instrumentation of service and service-using application can be instrumented to gain measurements for the evaluation of the WSLA. See Figure 1.

Both service provider and service customer may run their own instrumentation and measurement and management systems. Each organization may access measured metrics from various

sources, such as server-side metrics from the provider and client-side metrics from the customer. This allows parties to determine both a service's performance within a service provider's domain and its performance as experienced by a user.

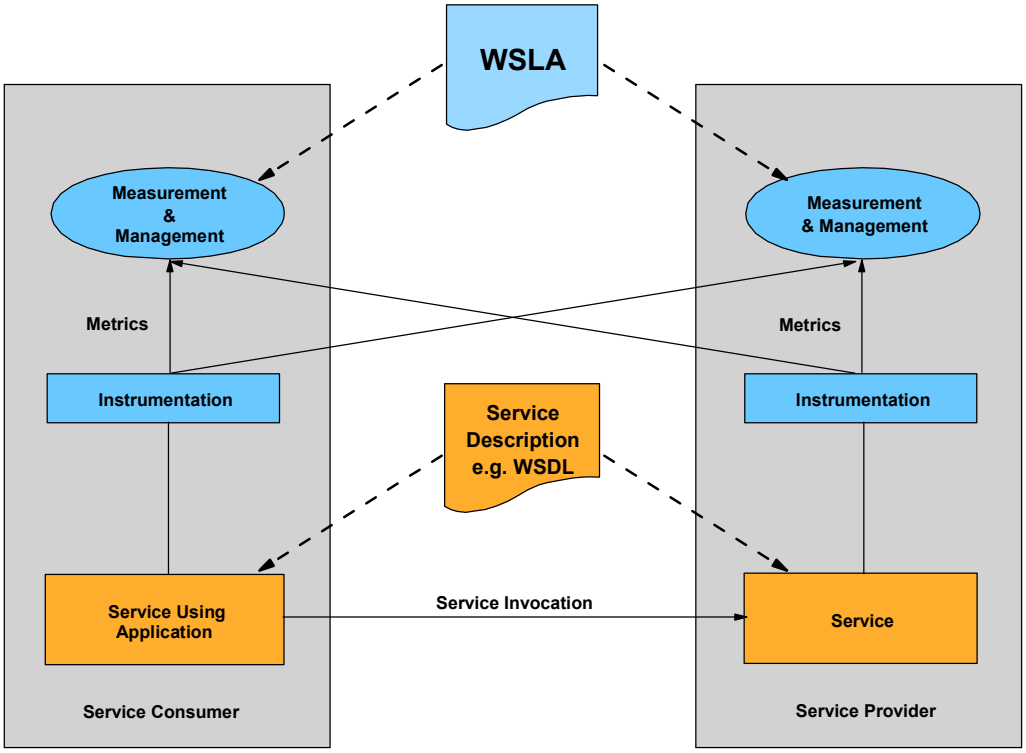

Figure 1: Role of a Web Service Level Agreement.

The relationship between a Web Service Level Agreement and the measurement and management system that deals with it is more dynamic than the relationship between a service description and its implementing service. Usually a service implements one or several service interfaces. It is unusual that service interfaces are added to a service dynamically because this step involves programming in many cases. However, it is easy to imagine that the same service is sold by a provider with different service levels, e.g. at different guaranteed response times. This may be the result of short term, individual negotiation and should be able to be processed flexibly by the measurement and management system.

1.4.2. Runtime Management of a WSLA

This specification assumes an abstract model of the runtime management of a WSLA, which is outlined in Figure 2. We assume that the measurement and management functionality is divided in three groups of functionality:

- The ***measurement*** functionality receives the measured metrics from the system's instrumentation. Instructions on how to measure a particular system parameter are defined in the measurement directives of a WSLA. The role of the measurement functionality is also to compute high level metrics, e.g. the average response time of a complete cluster of servers in a particular period, as defined in the metrics definitions of a WSLA. The measurement functionality must implement the functions that are required to compute the high level metrics.

- The set of metrics that are used in the guarantees of the WSLA are made available by the measurement function as SLA Parameters.
- The ***condition evaluation*** function evaluates the guarantees of the WSLA as defined in the WSLA. Guarantees are defined as predicates over SLA Parameters. The value of these

parameters can be obtained from the measurement function. The condition evaluation function must implement the relevant predicates to perform the guarantee evaluation. In the case of a guarantee violation, an action is invoked on the management function.

- The WSLA has as yet a limited model of ***management*** function. Management implements actions that are invoked upon guarantee violations. The subsequent course of action to remedy the problem is not subject to this specification. We assume that this is implemented as appropriate.

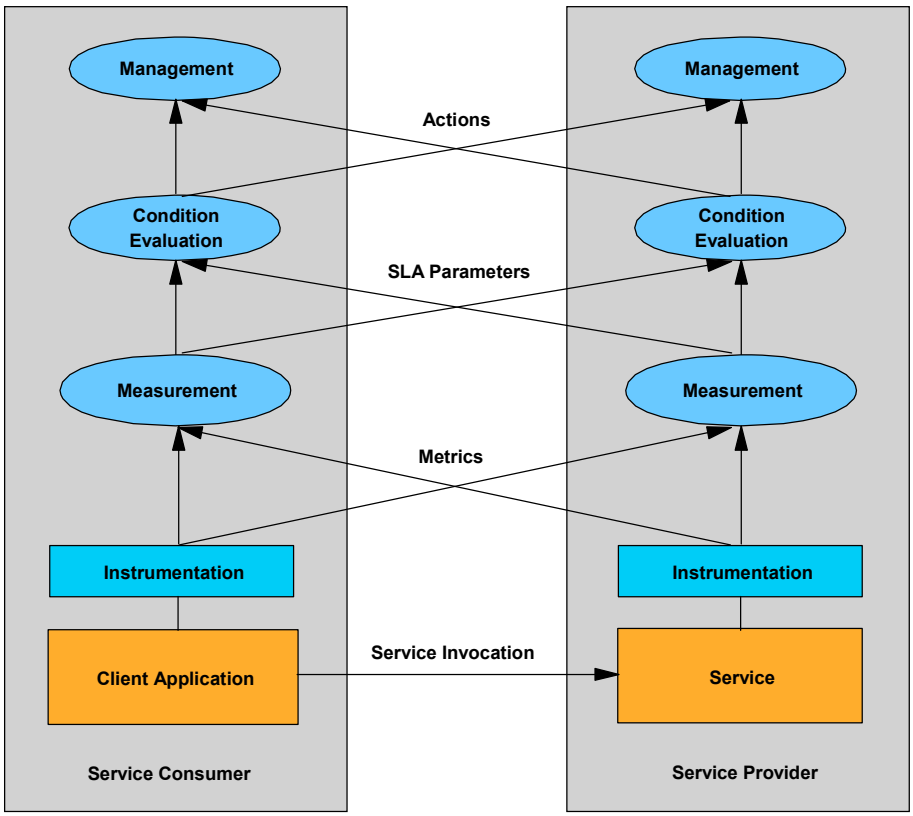

Figure 2: WSLA runtime management.

The figure illustrates that interactions between parties may occur at various function levels , if agreed upon in the WSLA. Measured (resource) and high level metrics may be exchanged by the measurement function, the condition evaluation function may retrieve SLA Parameters from various sources and management actions may be triggered from both provider and customer.

The WSLA runtime infrastructure can be used as well internally by the service provider for internal measurement, management and accounting purposes.

1.4.3. Third, Supporting Parties

In today's distributed Web based environment, one or both parties may choose to commission a part of the WSLA management activity to other parties. These third parties are called supporting parties. They are defined in the WSLA along with the service customer and service provider, the signatory parties to the WSLA. Supporting parties are sponsored by either signatory party or by both of them. Supporting parties can act in any combination of the following roles:

- A ***measurement service*** implements a part or all of the measurement function required by one or both signatory parties.

- A ***condition evaluation service*** implements condition evaluation function that covers all or a part of the guarantees of a WSLA.
- A ***management service*** implements the management actions of a signatory party.

Figure 3 outlines an example configuration

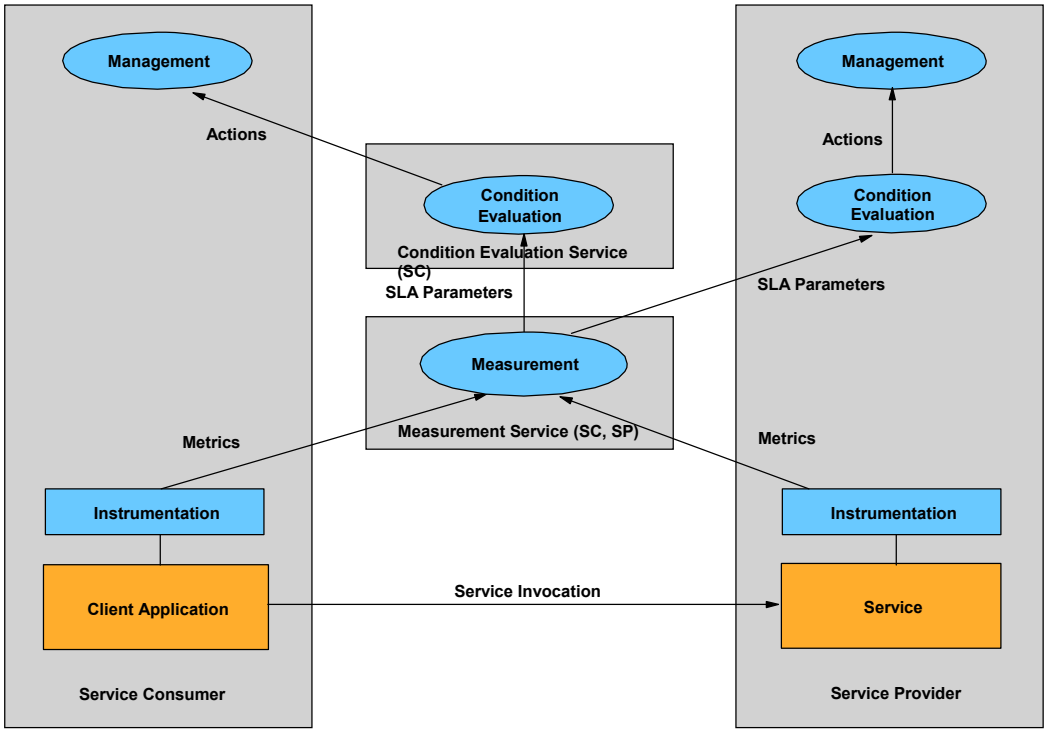

Figure 3: Supporting parties in a WSLA.

In the example depicted in figure 3 we can find 2 supporting parties: A measurement service implements the complete measurement function for both signatory parties (sponsors in parentheses). The service customer also outsources the condition evaluation to a third party whereas the service provider implements this functionality itself. Each signatory party implements its own management function.

1.4.4. WSLA Creation

A complete WSLA document is composed from all the information negotiated and agreed upon by the two parties, service provider and service customer. Information on supporting parties (e.g., roles, details of their actions) is specified by their sponsors, since the sponsored parties do not participate in creating a WSLA and full details of a composed WSLA (other than information related to their roles) are not visible to the sponsored parties. During deployment of a WSLA, however, the appropriate information is passed to various supporting parties by their sponsors.

In many scenarios, one of the parties (e.g., the service provider) will define most of the content of a WSLA and a service customer may simply agree to such information, and provide additional client-specific information (e.g., party information). In a more flexible scenario, while the service provider defines many of the aspects of a service, including definition of specific SLA metrics and measurement directives, it may offer a choice to the customer on the details of the guarantees (i.e., violation thresholds, action to be invoked upon violation, etc.). On the other extreme, the service provider and service customer may need to negotiate not just various sponsored roles, but also take an active role in the definition of SLA metrics (e.g., average response time, availability). In all cases, pre-specified, fixed information and negotiable elements as well as their

choices can be captured in a WSLA template. The representation of such a template is beyond the scope of this specification document.

The authoring process can be off-line, where the information is exchanged between the parties via e-mail or other human communication mechanisms. An interactive authoring tool uses a WSLA template, and presents graphically various input fields and choices to be made by an author [7]. The authoring tool understands the semantics of the WSLA schema as well as additional information captured in the template and assists the author in gathering the correct information in creating a complete WSLA document. At the end of this process the tool generates a WSLA document. Thus, an author can create a very complex WSLA by answering a few specific questions.

Alternatively, the WSLA creation can be negotiated in an online process. A template can be published in a registry such as UDDI. After a sequence of information exchange via negotiation steps, a WSLA document is created. Currently, a workgroup under OASIS is working on defining a generic negotiation protocol for negotiating any electronic document [8].

1.4.5. Deployment Process

The interpretation of the WSLA and the corresponding setup of the components required to supervise the WSLA is called deployment process. Each signatory party is responsible for the deployment of its function and the setup of the supporting parties that it sponsors. The information passed on to the various functions may not be the WSLA or a fragment but derived setup information in a proprietary format. However, in some cases, such as when using a condition evaluation service, passing on information in standard WSLA format may be appropriate.

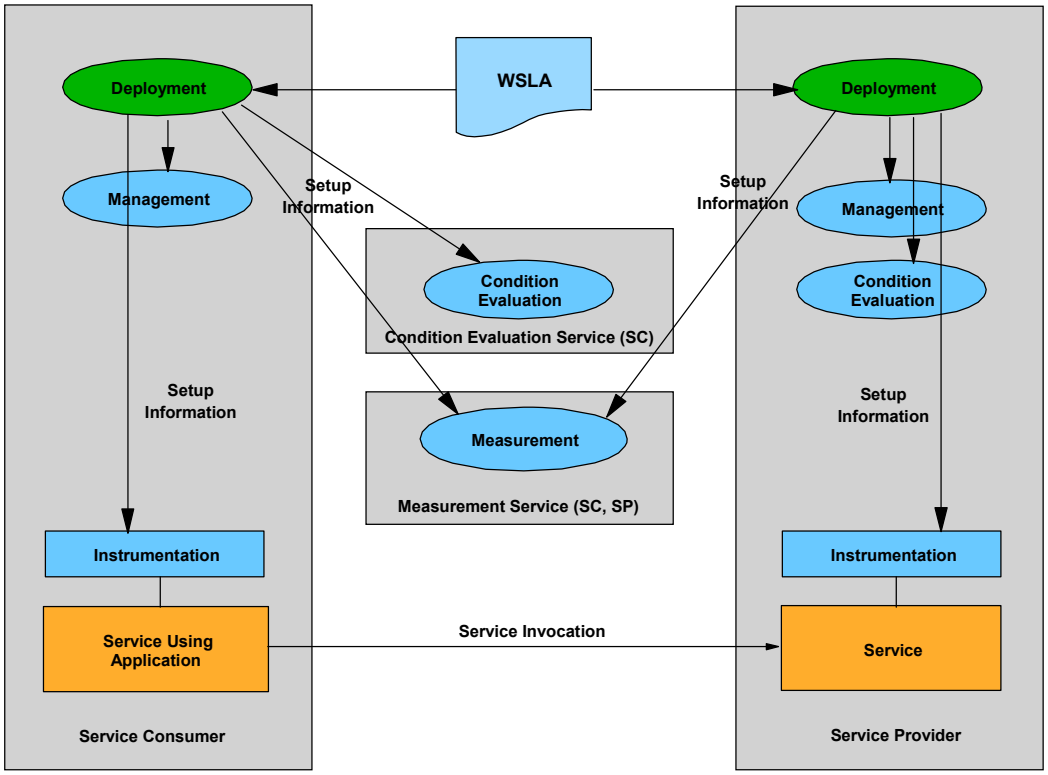

Figure 4: Deployment process.

Figure 4 illustrates the deployment process of the configuration outlined in the previous subsection.

1.5. Scope of the Specification

A WSLA specification covers the definition of the involved parties, the service guarantees and the service description. The party definition includes the set of management actions exposed to the other parties of the WSLA. The service description comprises the definition of the service parameters used in the guarantees and a specification of how the parameters are aggregated from basic metrics, as well as who is in charge of computing these metrics and how can the values be obtained. The guarantees include the management actions to be invoked in case of non-compliance with a guarantee. Optionally, a WSLA can comprise prices of services and penalties. The WSLA language addresses IT resource-level as well as business process SLAs.

However, the WSLA only covers the common view of the service and its performance aspects. The real system may be more complex than the agreed upon view defined in the WSLA. Also, more metrics may be collected internally and evaluated. Management functions can be involved not only for management of non-compliance, but for other purposes as well, such as the prevention of non-compliance by forecasting future behavior from measured metrics. All of this is important but not within the scope of a WSLA.

1.6. Structure of the Document

This document is structured as follows:

- In section 2 the core of the WSLA language is defined and explained. It first gives an overview of the core concepts of WSLA. Subsequently, for each element of the language, its type is defined and an example is given, if useful at that point.

- In section 3, the "standard extensions" are introduced. While the core language is aimed to be slim and not targeted at a particular technology, WSLA provides a mechanism to extend the language using XML schema derivation. The standard extensions section provides the authors of an agreement with a "good initial set" of extensions that facilitate relating the WSLA to WSDL defined services and WSDL defined management actions, defining common metrics in the context of Web Services and a set of standard predicates to define the contractual guarantees.

- The main part is concluded with acknowledgments and references.

- Appendix 1 contains the WSLA Schema.

- Appendix 2 illustrates WSLA in a very simple example.

- Appendix 3 gives the full SLA of the running example used in many parts the specification and introduced in section 1.7.

- Appendix 4 defines standard interfaces between parties that are referred to by the WSLA standard extensions.

- Appendix 5 defines the WSLA Service Deployment Information language, which is an extension of the WSLA language. It is used to represent third-party configuration information.

In the writing style and order of presentation in this specification: this specification features a top-down approach with respect to describing the constituent elements of an SLA -- that is, top-level elements are described first and their constituent parts later. While this approach is more interesting for the reader, it entails that regularly language elements are used that are defined in following sections. Where critical, appropriate forward pointers are given.

1.7. Running Example

To help the reader understand the specification and making it a slightly more pleasant experience, this document uses a running example throughout the specification to illustrate the use of the various elements of the language.

The SLA defines the relationship between a fictitious stock quote service provided by ACMEProvider and its customer, XInc (See Figure 6). The service provider offers the operation getQuote through SOAP over HTTP. The signature and the binding of this service are defined in a file StockQuoteService.wsdl independently of the SLA. The two parties, ACMEProvider and XInc, define a service level agreement that specifies the relevant service level parameters SLAParameters, primarily the response time and availability guarantees associated with the service. Since response time and availability can be estimated in many different ways, the SLA also specifies an exact definition of SLAParameters, and how the SLAParameters are to be measured and computed. The definition of the metrics used in the SLAParameters expresses this relationship between metrics of multiple sources and their aggregation.

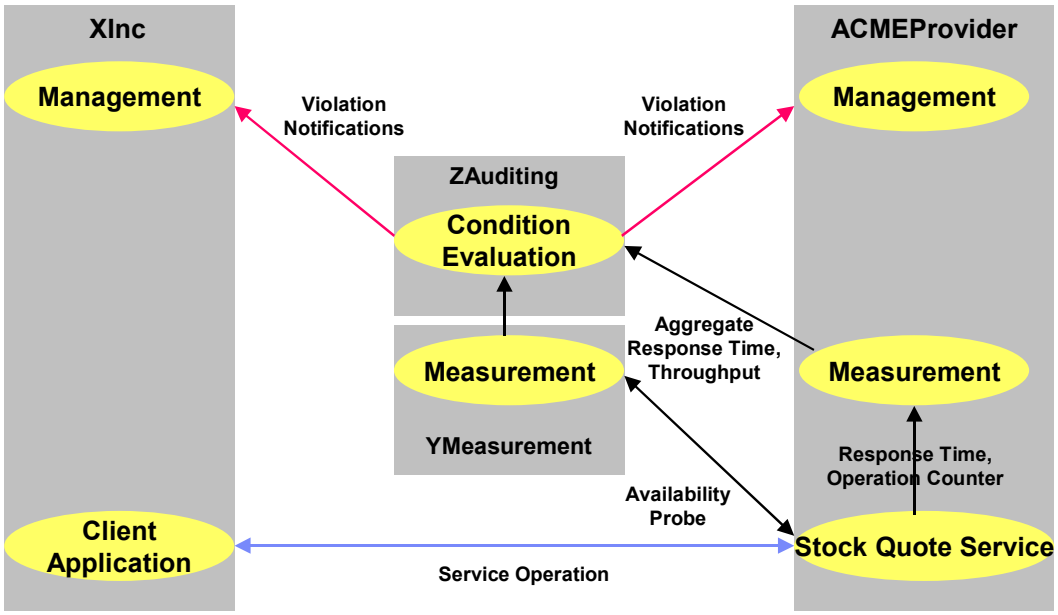

Figure 5: Example Scenario.

Our contracting parties use external parties to measure and enforce the SLA. In the example, the service provider is capable of measuring response time and throughput. It takes basic metrics and computes higher-level SLA parameters (e.g., aggregate response time computed over a specified timeframe). Additionally, it relies on an external organization, YMeasurement, for computing metrics related to availability. In the SLA, ACMEProvider will sponsor this role. A second organization, ZAuditing, sponsored by both the service provider and the customer, performs the condition evaluation and the notification of violations and special conditions that are defined in the SLA. The condition evaluation service provided by ZAuditing reads the metrics from the measurement services YMeasurement and ACMEProvider and evaluates the guarantees of the SLA. The interaction between the parties will be based on standard interfaces as outlined in Appendix 4.

Please find the complete SLA in the appendix.

To explain some language elements that are not really needed for this example, we "manufacture" an example in the spirit of the scenario above. The Stockquote example is chosen

to illustrate the references to service operations specified via WSDL. (The same example is also used to illustrate WSDL.).

1.8. WSLA Compliance Monitor Release Notes

The WSLA compliance monitor that is part of the WSPM has implemented all core language constructs and most of the standard extensions as defined in Section 3. Language elements that are not yet supported are marked with a "♠".

2. Web Service Level Agreement Language

2.1. Overview of the Main Concepts

Before delving into the details of the WSLA language, this overview outlines the main concepts that underlie the WSLA language. A UML class diagram represents the most important conceptual object types and some of their relationships. This diagram is an illustration only. The binding definition of the WSLA language is the XML schema and its explanation in the appendix of this document.

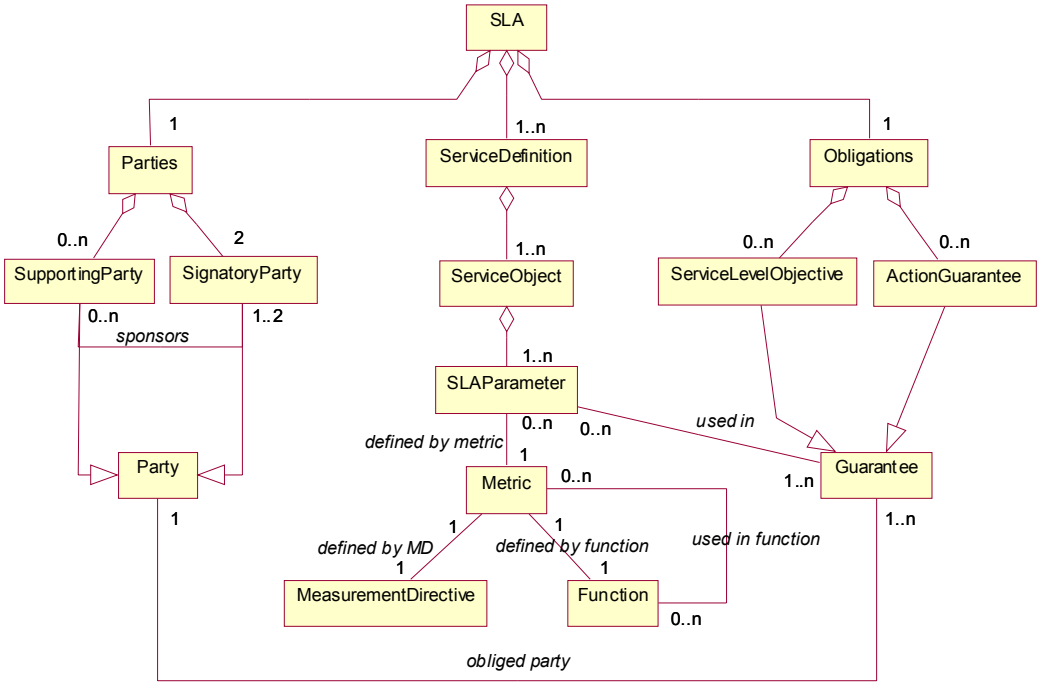

Figure 1: Overview of main WSLA concepts.

An SLA in the WSLA language contains three sections: A section describing the parties, a section containing one or more service definitions and the section defining the parties' obligations.

WSLA know two types of parties: signatory parties, namely service provider and service customer, and supporting parties. Signatory parties are the main parties to the SLA and are assumed to "sign" the SLA, bearing the final liability. Supporting parties are sponsored by one or both signatory parties to provide measurement and condition evaluation services.

A service definition contains one or more service objects. A service object is an abstraction of a service (e.g., a WSDL/SOAP operation, a business process, or an online storage service), whose properties relevant for defining the SLA's guarantees are described as SLAParameters. A service object can have one or more SLAParameters. What SLAParameters actually represent is defined by relating them to a metric. A metric is a specification of either how a value is measured from a source (such as an SNMP-instrumented networking device) by defining a measurement directive or how a metric is computed as defined in a function. The function can take other metrics and other input into account. We also refer to the measured metric as resource metric and to the computed metric as composite metric. Example: A resource metric measures an invocation counter of an application server for a particular operation. How to access the counter is defined in

a measurement directive. A composite metric “average number of invocations in an minute interval” is defined by a function specifying a time series of readings of the resource metric and computing the averages from this time series.

The obligations section contains two types of obligations: A service level objective is a guarantee of a particular state of SLA parameters in a given time period. The SLAParameter “average response time of operation getQuote” is less than 2 seconds between 9 AM and 5 PM EST while the SLA is in force. The other type of guarantee is an action guarantee, the promise to do something in a defined situation. An example: A notification is sent if a service level objective is violated. Every obligation has an obliged party. Service level objectives are typically the obligation of the service provider, particularly not of supporting parties. Action guarantees can have any party as the obliged, in particular also the supporting parties.

The concepts of this section typically appear in two forms in the XML Schema specification, as complexTypes and as elements. As a convention, we suffix the types with the ending type (e.g., SLAParameterType), while the elements just have the name of the concept (e.g., SLAParameter), or another name more specific for the context in which the element is defined.

2.2. Structure

A Web Service Level Agreement comprises the following major parts:

- **Parties** - describes the parties involved in the management of the Web Service. This includes the signatory parties as well as the supporting parties that are brought into the SLA to act on behalf of service provider or customer but cannot be held liable on the grounds of this SLA. The relationship of a sponsored party to their sponsor is not within the scope of this agreement.

- **Service Definitions** - describe the services the WSLA is applied to. The service definitions represent the common understanding of the contracting parties of the structure of the service, in terms of operations and the service's parameters and metrics that are the basis of the SLA. It also includes the specification of the measurement of a service's metrics.

- **Obligations** - define the service level that is guaranteed with respect to the SLAParameters defined in the service definition section. The promises to perform actions under particular conditions are also represented in this part.

The broad structure of WSLA is outlined in the type definition below.

Type Definition

```
<xsd:complexType name="WSLAType">
  <xsd:sequence>
    <xsd:element name="Parties" type="wsla:PartiesType"/>
    <xsd:element name="ServiceDefinition"
                 type="wsla:ServiceDefinitionType"
                 maxOccurs="unbounded"/>
    <xsd:element name="Obligations" type="wsla:ObligationsType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="SLA" type="wsla:WSLAType"/>
```

The element `SLA` of type `WSLAType` is the root element of a WSLA definition. An SLA contains one Parties section, multiple ServiceDefinition sections and one Guarantees section. Multiple ServiceDefinition elements allow an SLA to cover services defined in multiple places within one SLA.

As a convention, all WSLA elements and types are defined in the namespace `"wsla"`.

Example

An example top-level document structure looks as follows:

```
<?xml version="1.0">
<wsla:SLA
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:wsla="http://www.ibm.com/wsla"
     name="StockquoteServiceLevelAgreement12345" >
   <Parties>
      ...
   </Parties>
   <ServiceDefinition>
      ...
   </ServiceDefinition>
   <Obligations>
      ...
   </Obligations>
</wsla:SLA>
```

2.3. Parties

The `Parties` section describes the binding and contact info for each of the participants. It distinguishes primary participants, signatory parties that negotiate and sign such a contract, from the supporting role players (i.e., measurement and management service providers) that are sponsored by the primary participants. The sponsored parties provide services to the primary parties and may have separate service agreements with their respective sponsors.

Type Definition

```
<xsd:complexType name="PartiesType">
  <xsd:sequence>
    <xsd:element name="ServiceProvider" type="wsla:SignatoryPartyType"/>
    <xsd:element name="ServiceConsumer" type="wsla:SignatoryPartyType"/>
    <xsd:element name="SupportingParty" type="wsla:SupportingPartyType"
                                        minOccurs="0"
                                        maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

- The ServiceProvider is the party that provides the service and guarantees the associated properties. There is exactly one `ServiceProvider`.
- The `ServiceConsumer` is the recipient of the service. There is exactly one `ServiceConsumer`.
- Any number of `SupportingParties` is involved in measuring the service's parameters, supervising the given guarantees and managing the corrective procedures in case of failure.

2.3.1. Party

The party description captures the properties of a party that are common to all participating organizations, regardless of their particular role.

Type Definition

```
<xsd:complexType name="PartyType" abstract="true">
  <xsd:sequence>
    <xsd:element name="Contact" type="wsla:ContactInformationType"/>
    <xsd:element ref="wsla:Action" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

The Party type is abstract because each party is either a signatory or a supporting party, which requires additional attributes in each case. Thus, party cannot be instantiated directly.

- `Contact` is the relevant information to interact with the party.
- `Action` is an operation that a party can perform when invoked upon the occurrence of an event, for example, violations of a guarantee. Information associated with an event such as the WSLA ID, violated SLAParameter names, violation values, etc. are passed in invoking these actions.
- The `name` is the unique identifier of the party. Typically, organizations want to use unique naming and identification systems that are generally accepted in their respective country or region, such as DUNS number in the US or VAT number in the UK. In a later version we shall change the ID convention to two attributes, a naming schema identifier and the ID in this particular schema. The name is used primarily to refer to parties later in the WSLA document. Since it is used in XML lists use only Strings not containing blanks.

2.3.2. Signatory Party

Signatory parties are the main contracting parties. Each contract has exactly one service provider and one service consumer of type signatory party.

Type Definition

```
<xsd:complexType name="SignatoryPartyType">
  <xsd:complexContent>
    <xsd:extension base="wsla:PartyType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The `SignatoryPartyType` extends the common `PartyType`.

Example

```
    <ServiceProvider
        name="ACMEProvider">
        <Contact>
            <Street>PO BOX 218</Street>
            <City>Yorktown, NY 10598, USA</City>
        </Contact>
        <Action name="notification"
                partyName="ACMEProvider"
                xsi:type="WSDLSOAPActionDescriptionType">
           <WSDLFile>notification.wsdl</WSDLFile>
           <SOAPBindingName>soapnotification</SOAPBindingName>
           <SOAPOperationName>notification</SOAPOperationName>
        </Action>
    </ServiceProvider>
    <ServiceConsumer
        name="XInc">
        <Contact>
            <Street>30 Saw Mill River RD</Street>
            <City>Hawthorne, NY 10532, USA</City>
        </Contact>
        <Action name="notification"
                partyName="XInc"
                xsi:type="WSDLSOAPActionDescriptionType">
           <WSDLFile>notification.wsdl</WSDLFile>
           <SOAPBindingName>soapnotification</SOAPBindingName>
           <SOAPOperationName>notification</SOAPOperationName>
        </Action>
    </ServiceConsumer>
```

The example shows the definition of a service provider and a consumer of type `SignatoryPartyType`. It only contains a name attribute, the contact information and an `Action`, a `notification` action in both cases.

2.3.3. Supporting Parties

Supporting parties are contributors to the execution of the SLA, which are neither consumer nor provider. Examples are measurement service providers and management service providers. Parties of this type are *sponsored* by either one or both signatory parties.

Type Definition

```
<xsd:complexType name="SupportingPartyType">
  <xsd:complexContent>
    <xsd:extension base="wsla:PartyType">
      <xsd:sequence>
        <xsd:element name="Sponsor" type="xsd:string" maxOccurs="2"/>
      </xsd:sequence>
      <xsd:attribute name="role" type="wsla:RoleType"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The `SupportingPartyType` extends the common `PartyType` by specific attributes.

- The `Sponsor` element contains references to signatory parties. Either one or both signatory parties can act as a sponsor.
- A SupportingParty assumes a particular role of RoleType.

Example

```
  <SupportingParty
        name="YMeasurements"
        role="MeasurementService">
        <Contact>
            <Street>Saeumerstrasse 4</Street>
            <City>CH-8803 Rueschlikon, Switzerland</City>
        </Contact>
        <Sponsor>ACMEProvider</Sponsor>
  </SupportingParty>
  <SupportingParty
        name="ZAuditing"
        role="ConditionEvaluationService">
        <Contact>
            <Street>Hursley Park</Street>
            <City>Winchester, England, SO21 2JN</City>
        </Contact>
        <Action name="parameterUpdate"
                partyName="ce"
                xsi:type="WSDLSOAPActionDescriptionType">
           <WSDLFile>parameterUpdate.wsdl</WSDLFile>
           <SOAPBindingName>soapnotification</SOAPBindingName>
           <SOAPOperationName>parameterUpdate</SOAPOperationName>
        </Action>
        <Sponsor>ACMEProvider</Sponsor>
        <Sponsor>XInc</Sponsor>
  </SupportingParty>
```

The example shows two supporting parties. Party `YMeasurements` is a measurement service sponsored by the service provider `ACMEProvider`. Party `ZAuditing` computes aggregated parameters and is sponsored by both service provider and consumer.

2.3.4. Contact Information

The contact information can be represented as any XML structure suitable to capture the required information. In future versions of this specification this should follow an international standard for representing contact information, such as ones specified by the ebXML core components working group or the DMTF Common Information Model (CIM).

Type Definition

```
<xsd:complexType name="ContactInformationType">
  <xsd:sequence>
    <xsd:any/>
  </xsd:sequence>
</xsd:complexType>
```

2.3.5. Role Type

RoleType represents the set of roles that supporting parties can assume - the roles of the signatory parties are explicitly spelled out in their `ServiceProvider` and `ServiceConsumer` elements.

Type Definition

```
<xsd:simpleType name="RoleType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="MeasurementService"/>
    <xsd:enumeration value="ManagementService"/>
    <xsd:enumeration value="ConditionEvaluationService"/>
  </xsd:restriction>
</xsd:simpleType>
```

This type is a simple type consisting of an enumeration of eligible role strings:

- `MeasurementService` represents a 3rd Party that measures parameter metrics according to measurement directives.
- `ConditionEvaluationService` is a role that reads input from multiple `MeasurementServices` and notifies relevant parties, (e.g. a `ManagementService`).
- `ManagementService` is informed about violations of the SLA and takes the corresponding corrective measures.

2.3.6. Action

An action represents an activity that is undertaken by a particular party in the course of dealing with the SLA and which is triggered by another party to the contract, either signatory or supporting. ActionDescriptionType is an abstract type that provides the common umbrella for all action specifications. The scope and sophistication of actions can vary widely. Simple actions include "send notification" or "open trouble ticket". Advanced actions may be immediate assignment of resources.

Type Definition

```
<xsd:complexType name="ActionDescriptionType" abstract="true">
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="partyName" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="Action" type="wsla:ActionDescriptionType"/>
```

`ActionDescriptionType` is an abstract class that defines the commonalties of all actions:

- A unique `name` that identifies it.

- A reference to the `organization` that provides the action.

Example

```
<Action xsi:type="WSDLSOAPOperationDescriptionType"
        name="notification"
        partyName="ZAuditing">
<WSDLFile>Notification.wsdl</WSDLFile>
  <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
  <SOAPOperationName>Notify</SOAPOperationName>
</Action>
```

The example shows an `Action` named `"Notification"` of the specific subtype `"WSDLSOAPActionDescription"` as defined in the standard extensions in section 3. The `Action` is implemented by the organization with the reference `"support2"`, the `ViolationDetection` organization. All elements of this `Action` are subtype-specific.

2.4. Service Definitions

The service definitions capture the common view of the service that is shared by the contracting parties. This includes references to the service operations and bindings as well as the information needed to define the service level guarantees in the subsequent part of the SLA. Primarily, the common domain specification consists of definitions of `SLAParameter`s. Since `SLAParameter`s can be attached to different kinds of objects, we introduce an abstract object that is specialized by numerous objects of the service definitions part.

2.4.1. Service Definition

A service definition captures the information needed about a service to define the service level guarantees in the subsequent part of the SLA.

Type Definition

```
<xsd:complexType name="ServiceDefinitionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ServiceObjectType">
      <xsd:sequence>
        <xsd:element ref="wsla:Operation" minOccurs="0"
                     maxOccurs="unbounded"/>
        <xsd:element ref="wsla:OperationGroup" minOccurs="0"
                     maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

A `ServiceDefinitionType` extends a `ServiceObjectType` (see 2.4.2), meaning that a service definition can have `SLAParameters` attached to it. Its additional elements are:

- An `Operation` contains the description of a particular operation that the service provider implements. This contains its observable properties as well as the binding-specific definition of how to call the operation. There can be any number of `Operations`. The core of this language definition can support any kind of service description. However, since our primary focus is Web services we define in section 3.3.1 a specific extension needed to refer to a WSDL-defined service.

- An `OperationGroup` is a set of `Operations` that share the same `SLAParameters`. There can be any number of `OperationGroups`.

Example

```
<ServiceDefinition
  name="StockQuoteservice">
  <SLAParameter>
    ...
  </SLAParameter>
  <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType">
    ...
  </Operation>
  <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType">
    ...
  </Operation>
  <OperationGroup>
    ...
  </OperationGroup>
</ServiceDefinition>
```

The example fragment shows a service `"StockQuoteService"` with one `SLAParameter`, two `Operations and` an `OperationsGroup`.

2.4.2. Service Object

The service object provides an abstraction for all conceptual elements of an SLA for which `SLAParameters` and the corresponding Metrics can be defined and which are subject to service level objectives. Subtypes of `ServiceObject` include `ServiceDefinition`, as discussed above, `Operations` and `OperationGroups`. It can be extended to additional types, e.g., to a Web hosting service defined by a URL.

Type Definition

```
<xsd:complexType name="ServiceObjectType" abstract="true">
  <xsd:sequence>
    <xsd:element ref="wsla:Schedule" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="wsla:Trigger" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="wsla:Constant" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="wsla:MetricMacroDefinition" minOccurs="0"
                                                  maxOccurs="unbounded"/>
    <xsd:element ref="wsla:MetricMacroExpansion" minOccurs="0"
                                                 maxOccurs="unbounded"/>
    <xsd:element ref="wsla:SLAParameter" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="wsla:Metric" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

ServiceObject is an abstract type containing the following elements:

- Any number of `Schedules` can be part of a service object. A `Schedule` describes a sequence of points in time that can be referred to for building time series, compute derived metrics or verify conditions. Schedules are defined independently of their using elements because several items of the SLA may want to refer to the same schedule, for example, two different values should be measured at the same intervals.

- A `ServiceObject` can have any number of `Triggers` defined. A `Trigger` is a specification of a point in time to which the execution of activities, the evaluation of guarantees etc. can be bound.

- A service object can contain any number of `Constants`. `Constants` can be of any type.

- Service objects can contain any number of `MetricMacroDefinitions` that can be used to define metrics of similar structure in a more efficient way. See `Metric`.

- `MetricMacroExpansion` make use of the `MetricMacroDefinitions`. A set of metrics is defined very efficiently in this way. See `Metric`.
- Any number of `SLAParameters` can be defined. A `SLAParameter` describes an observable property of a service that can be retrieved at execution time.
- Any number of `Metrics` can be defined. Metrics are input to the definition of a parameter and define how the values of `SLAParameters` are to be calculated or measured.
- The attribute `name` defines the unique name of the `ServiceObject`.

2.4.3. Schedule

A Schedule is a set of points in time with a defined begin and end and an interval between two points in time. It can be referred to at any point in a WSLA.

Type Definition

```
<xsd:complexType name="ScheduleType">
  <xsd:sequence>
    <xsd:element name="Period" type="wsla:PeriodType"/>
    <xsd:element name="Interval" type="wsla:IntervalType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="Schedule" type="wsla:ScheduleType"/>
```

- A Schedule has a `Period` that is described by a begin and end time.
- The `Interval` element defines a duration in time at the desired granularity.
- The attribute `name` assigns a name to the schedule.

Example

```
    <Schedule name="MainSchedule">
        <Period>
          <Start>2001-11-30T14:00:00.000-05:00</Start>
          <End>2001-12-31T14:00:00.000-05:00</End>
        </Period>
        <Interval>
          <Hours>2</Hours>
          <Minutes>30</Minutes>
        </Interval>
    </Schedule>
```

The example shows a Schedule named `"MainSchedule"` that lasts one month and whose scheduling interval is 2 hours and 30 minutes.

2.4.4. Trigger ♠

A `Trigger` defines a point in time to which the execution of monitoring activity can be linked. Typically, this comprises the reading of a measurement from the infrastructure or the evaluation of an obligation. A trigger may be referred to from multiple places in a WSLA.

Type Definition

```
<xsd:complexType name="TriggerType">
      <xsd:choice>
        <xsd:element name="Time" type="xsd:dateTime"/>
      </xsd:choice>
  <xsd:attribute name="name" type="xsd:string"/>
```

```
</xsd:complexType>

<xsd:element name="Trigger" type="wsla:TriggerType"/>
```

- The `Time` element defines the date and time of the trigger execution.
- The attribute `name` assigns a name to the trigger.

Example

```
<Trigger name="EvaluationTrigger">
   <Time>2002-12-31T14:00:00.000-05:00</Time>
</Trigger>
```

The example shows a `Trigger` named `"EvaluationTrigger"` that executes on December 31, 2002, midnight, EST.

2.4.5. Constant ♠

A `Constant` assigns a name to a simple value that can be referred in other definitions.

Type Definition

```
<xsd:complexType name="ConstantType">
   <xsd:choice>
     <xsd:element name="String" type="xsd:string" minOccurs="0"/>
     <xsd:element name="Integer" type="xsd:integer" minOccurs="0"/>
     <xsd:element name="Float" type="xsd:float" minOccurs="0"/>
   </xsd:choice>
   <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="Constant" type="wsla:ConstantType"/>
```

- Either a String, Integer or Float element can be defined or its respective type.
- The attribute `name` assigns a name to the constant.

Example

```
    <Constant name="ResponseTimeThreshold">
      <Float>2.5</Float>
    </Constant>
```

The example shows a Constant named `"ResponseTimeThreshold"` having the float value 2.5.

2.4.6. SLAParameter

Parameters are the main elements of description of a service. A parameter describes an observable property of a service whose value can be obtained from a source of measurement. An SLAParameter can be used in a `Guarantee` of the SLA.

Type Definition

```
<xsd:complexType name="SLAParameterType">
  <xsd:sequence>
      <xsd:element name="Metric" type="xsd:string"/>
      <xsd:element name="Communication"
                   type="wsla:SLAParameterCommunicationType"
                   minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="type" type="wsla:Type"/>
  <xsd:attribute name="unit" type="xsd:string"/>
</xsd:complexType>
```

```
<xsd:element name="SLAParameter" type="wsla:SLAParameterType"/>
```

- A `SLAParameter` has one `Metric` assigned to it. It refers to it using its unique name. The metric defines how the value of the SLAParameter is computed. This metric is used to determine the `SLAParameter's` value at execution time.

- The `Communication` element defines how other parties get access to values of the `SLAParameter`. This can be either by active (push) update using a `ParameterUpdate` operation (see Appendix 4) or by providing access (pull) for other parties to retrieve the current SLAParameter's value using the `GetSLAParameterValue` method (see Appendix 4). The communication section can be omitted if no interaction with other parties is expected (e.g. condition evaluation is performed within the same party's domain).

- The attribute `name` is a unique ID of the parameter.

- The attribute `type` defines the domain of the metric's value.

`SLAParameter` is also defined as a global element for further reuse.

Example

```
      <SLAParameter name="TransactionRate"
                    type="float"
                    unit="transactions / hour">
        <Metric>Transactions</Metric>
        <Communication>
          <Source>ACMEProvider</Source>
          <Pull>ZAuditing</Pull>
          <Push>ZAuditing</Push>
        </Communication>
      </SLAParameter>
```

The example shows an `SLAParameter` named `"TransactionRate"` that is based on the metric `"Transactions". ACMEProvider` is in charge of providing this value. See next subsection for `Communication` details.

2.4.7. SLAParameter Communication

The `SLAParameter`'s Communication Type represents information on how the current values of an SLAParameter can be interchanged between parties. This interaction can be either proactive, that is, an update is sent to defined parties each time a new value is computed (push), or other parties can retrieve the current value of an SLAParameter whenever they want (pull).

For the interaction of `SLAParameter` values it is assumed that standard operations are used to facilitate this interaction. The `ParameterUpdate` operation is the standard operation for "pushing" values to other parties; the `GetSLAParameterValue` operation facilitates the "pull". The interfaces of both operations are defined as Web service in WSDL (see Appendix 4). For all parties listed in the "pull" list, the party definition must comprise a specification of a `ParameterUpdate` Action definition that points to a WSDL file containing the interface specification and this party's implementation specification of the `ParameterUpdate` operation.

Type Definition

```
<xsd:complexType name="SLAParameterCommunicationType">
  <xsd:sequence>
      <xsd:element name="Source" type="xsd:string"/>
      <xsd:element name="Pull" type="wsla:StringList" minOccurs="0"/>
      <xsd:element name="Push" type="wsla:StringList" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

- The `Source` is the unique name of the party that is in charge of computing and providing the values of an `SLAParameter`. This can be either a signatory party or a supporting party.
- The `Pull` element defines the list of parties that are allowed access to values of the SLAParameter using the `GetSLAParameterValue` method (see Appendix 4). The list contains the IDs of the parties.
- The Push element defines the list of parties to which `ParameterUpdate` operations are sent (see Appendix 4) when a new value of an `SLAParameter` is produced by the party in charge defined in the `Service` element.

Example

```
<Communication>
  <Source>ACMEProvider</Source>
  <Pull>ZAuditing</Pull>
  <Push>ZAuditing</Push>
</Communication>
```

The example shows the `SLAParameterCommunicationType` used in an `SLAParameter` definition. The party `"ACMEProvider"` is commissioned to compute the SLAParameter's value and it must send all new values to the party `"ZAuditing"`. In addition, `"ZAuditing"` may retrieve the value (pull), too.

2.4.8. Metric

Metrics are definitions of values of service properties that are measured from a service providing system or computed from other metrics and constants. Metrics are the key instrument to describe exactly what `SLAParameters` mean by specifying how to measure or compute the parameter values.

Type Definition

```
<xsd:complexType name="MetricType">
  <xsd:sequence>
      <xsd:element name="Source" type="xsd:string"/>
      <xsd:element name="MetricURI" type="xsd:anyURI" minOccurs="0"/>
      <xsd:choice>
        <xsd:element name="MeasurementDirective"
                     type="wsla:MeasurementDirectiveType"
                     minOccurs="0"
                     maxOccurs="unbounded"/>
        <xsd:element name="Function"
                     type="wsla:FunctionType"
                     minOccurs="0" />
        <xsd:element name="MeasurementDirectiveVariable"
                     type="wsla:MDVariableType"/>
      </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="type" type="wsla:Type"/>
  <xsd:attribute name="unit" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="Metric" type="wsla:MetricType"/>
```

A metric is a complex type that defines all relevant information about the type of the metric's value, where to find it and how to gain it, be it by measuring it or by computing it from other metrics.

- The `Source` is a reference to the party that is in charge of gaining the metric's value and making it available.

- The `MetricURI` is the locator where organizations can retrieve the metric's value at runtime, if they are entitled to it.
- A metric description can contain one `MeasurementDirective`. This element describes how the metric's value should be measured. Typically, `MeasurementDirectives` are instructions for a Measurement Service. A `MeasurementDirective` is alternative to a `Function`.
- If a metric is computed from other metrics a metric's description contains a `Function` element detailing how to compute the value.
- If the metric is defined in the context of a metric macro, a `MeasurementDirectiveVariable` can be defined to leave this definition to be filled in by the macro expansion. See `MetricMacroDefinition` and `MetricMacroExpansion`.
- The attribute `name` is the unique ID of the metric.
- The attribute `type` represents its domain, e.g. integer, float, etc.
- The attribute `unit` contains a string, e.g. `"transactions/second"`, `"percent"`, etc.

Example

```
 <Metric name="ResponseTime" type="long" unit="milliseconds">
    <Source>ACMEProvider</Source>
    <Function xsi:type="Minus" resultType="double">
        <Operand>
          <Function xsi:type="TSSelect" resultType="long">
            <Operand>
              <Metric>SumResponseTimeTimeSeries</Metric>
            </Operand>
           <Element>0</Element>
          </Function>
        </Operand>
        <Operand>
          <Function xsi:type="TSSelect" resultType="long">
            <Operand>
              <Metric>SumResponseTimeTimeSeries</Metric>
            </Operand>
            <Element>-1</Element>
          </Function>
        </Operand>
    </Function>
</Metric>
 <Metric name="MeasuredStatus" type="integer" unit="">
    <Source>YMeasurement</Source>
    <MeasurementDirective xsi:type="StatusRequest"
                          resultType="integer">
         <RequestURI>
           http://ymeasurement.com/StatusRequest/GetQuote
         </RequestURI>
    </MeasurementDirective>
</Metric>
```

The example shows two metrics, the first one being a composite metric computed from another metric, the second one being a measured metric:

Metric `"ResponseTime"` is a metric of type long, which is computed using a "complex" function that refers to other metrics. Details of functions are explained in 2.3.8 and a set of standard functions is listed in section 3.

Metric `"MeasuredStatus"` is measured using the `MeasurementDirective "StatusRequest"`. This measurement directive indicates the URI where the measurement can be read from a resource.

2.4.9. Measurement Directives

Measurement directives define how parameter values are to be measured by the organization that makes a metric's value available. How the measurement is conducted and which information is needed for this purpose depends strongly on the particular system to which measurement is applied. Some systems expose standard management interfaces such as SNMP [9] or CIM [10], in other cases information can be collected through interfaces of systems management software such as Tivoli's offerings. Other types of measurement require probing, which entails that the probing interface needs to be captured. This variety asks for a very flexible approach to describe the information that is necessary to specify how measurements should be taken from a particular source. WSLA propose a Web services-based approach to access instrumentation in Appendix 4.5, but the WSLA concept of a measurement directive is open to all kinds of interfaces.

The measurement directive for a particular metric corresponds to a data item that is exposed by the instrumentation of a service. We assume the interpretation of the relationship of exposed data by instrumentation to a measurement directive of a particular service object is understood by the instrumentation of this service. For example, if an application server exposes a service invocation counter, it should be understood how to describe this counter in a measurement directive. Apparently, measurement directives can only describe measurements that can be provided by the instrumentation of a system or can be probed. In the example cases used in this specification, service level objectives are given on a per-operation basis. This requires that, for example, an application server actually provides per-operation data for response times through its instrumentation or in a log. While some application servers or gateways offer this capability today, others might need to be extended. It will be helpful to define a suitable set of basic metrics that consumers can expect to be available in all systems of a similar kind. For example, all application servers should provide counters for different kinds of operations and the corresponding performance metrics.

To provide necessary flexibility, WSLA defines an abstract type that can and must be extended to suit the needs of particular environments. It is conceivable that over time standard measurement directives for particular classes of systems evolve. In section 3 there are a number of definitions of simple common measurement directives.

Type Definition

```
<xsd:complexType name="MeasurementDirectiveType" abstract="true">
  <xsd:attribute name="resultType" type="wsla:Type"/>
</xsd:complexType>
```

MeasurementDirectiveType is an abstract type that needs to be extended by concrete measurement types.

- Common to all measurement directives is the availability of type information as defined in the attribute `resultType`.

This specification contains a number of standard measurement directives in section 3.

Example

```
      <MeasurementDirective xsi:type="wsla:ResponseTime"
                            resultType="double">
        <MeasurementURI>http://support1.com/testResponse
        </MeasurementURI>
      </MeasurementDirective>
```

The example shows a measurement directive of the specific subtype `"wsla:ResponseTime"`, which extends `MeasurementDirectiveType`. Its `resultType` is `"double"`. There is one subtype-

specific element `MeasurementURI`. This measurement directive refers to a measurement to be conducted by probing a specified test interface.

2.4.10. Function

A Function specifies how to compute a metric's value from the values of other metrics and constants. Functions are central for describing exactly how `SLAParameters` are computed from resource metrics. Similarly to measurement directives, there are a number of potentially domain specific functions that describe concrete computations for a particular type of aggregation. Thus, a function is also an abstract type that needs to be extended to meet the requirements of the application domain. However, many applications require a similar set of aggregation functions. In section 3 this specification defines a set of basic arithmetic functions and time series-related functions that enable expressing the most common computations, sufficient for many SLAs. This set can be extended as needed.

Type Definition

```
<xsd:complexType name="FunctionType" abstract="true">
  <xsd:attribute name="resultType" type="wsla:Type"/>
</xsd:complexType>
```

The type definition is abstract and serves as a common umbrella for all derived specific function types.

- The attribute `"resultType"` indicates the type of value that the function returns. This is important for consistency checks.

Depending on the particular function, functions can be nested to contain other functions, if appropriate.

2.4.11. Operation Description

An operation of a service description (e.g., defined in WSDL) is the atomic unit to which service characteristics can be attached to in a SLA. `OperationDescriptionTypes` relate the SLA to these underlying service operations, thus representing the link between an SLA and the service it refers to. `OperationDescriptionType` is an abstract type that provides the common umbrella for all operation specifications. For specific methods of service description (e.g., WSDL) this type must be extended by elements that refer to operations in this particular service description method.

Type Definition

```
<xsd:complexType name="OperationDescriptionType" abstract="true">
  <xsd:complexContent>
    <xsd:extension base="wsla:ServiceObjectType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Operation" type="wsla:OperationDescriptionType"/>
```

OperationDescription extends `ServiceObjectType`, that is, it can carry `SLAParameters`. This type is abstract. Operation description types used in an SLA must extend this type.

Example

```
 <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType"
            name="WSDLSOAPGetQuote">

   <SLAParameter name="AverageResponseTime"
                 type="float"
                 unit="seconds">
     ....
   </SLAParameter>
   ....
```

```
   <Metric name="SumResponseTimeTimeSeries" type="TS"
           unit="milliseconds">
       <Source>ACMEProvider</Source>
       <Function xsi:type="TSConstructor" resultType="TS">
           <Schedule>hourlyschedule</Schedule>
           <Metric>SumResponseTime</Metric>
           <Window>2</Window>
       </Function>
   </Metric>
   ....
 <!-- This part is the wsdl-specific definition of the operation. -->
   <WSDLFile>StockQuoteService</WSDLFile>
   <SOAPBindingName>GetQuoteSoapBinding</SOAPBindingName>
   <SOAPOperationName>getQuote</SOAPOperationName>
 </Operation>
```

This is an example of an `Operation` specification. The particular sub-type of this example operation is `"wsla:WSDLSOAPOperationDescriptionType"`, a type of the standard extension of WSLA as defined in section 3. The operation has one `SLAParameter` associated with it, `"AverageResponseTime"`. Also, the metrics `"SumResponseTimeTimeSeries"` is shown in this example. Towards the end of the specification of the operation, there are some subtype-specific elements such as `"WSDLFile"`.

2.4.12. Operation Group

An `OperationGroup` is a shortcut to describe the common `SLAParameters` of a set of `Operations`; the `SLAParameters` exist for each of the member `Operations` of the `OperationGroup`.

Type Definition

```
<xsd:complexType name="OperationGroupType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ServiceObjectType">
      <xsd:sequence>
        <xsd:element ref="wsla:Operation" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="OperationGroup" type="wsla:OperationGroupType"/>
```

The `OperationGroupType` extends `ServiceObjectType`, meaning it can have `SLAParameters`. It extends it by:

- At least one, but reasonably several `Operation` elements.

Example

```
 <OperationGroup name="ReadOperations">

   <SLAParameter name="AverageResponseTime"
                 type="float"
                 unit="seconds">
     <Metric>AverageResponseTime</Metric>
   </SLAParameter>

   <Metric name="SumResponseTimeTimeSeries" type="TS"
           unit="milliseconds">
       <Source>ACMEProvider</Source>
       <Function xsi:type="TSConstructor" resultType="TS">
           <Schedule>hourlyschedule</Schedule>
           <Metric>SumResponseTime</Metric>
           <Window>2</Window>
       </Function>
```

```
   </Metric>
   .......
  <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType"
             name="WSDLSOAPGetStockQuote">
  </Operation>
  <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType"
             name="WSDLSOAPGetIndex">
  </Operation>
</OperationGroup>
```

An OperationGroup `"ReadOperations"` defines an `SLAParameter` "`AverageResponseTime"` and several metrics. These SLAParameters are applied to the operations named `"WSDLSOAPGetStockQuote"` and `"WSDLSOAPGetIndex"`. In the specific case shown here the operations have the particular sub-type `"wsla:WSDLSOAPOperationDescriptionType"` that is part of the standard extensions and defined in section 3 of this document.

Runtime naming convention

Operation groups are a shortcut to define metrics and SLA parameters that are equivalent for a group of operations without the need to repeat the same kind of definition for each individual operation. A major aspect of metrics and SLA parameter definitions is the introduction of names that can be used at runtime to identify information that is sent between services. However, when using operation groups, we do not introduce individual names for metrics and SLA parameters for each individual operation. We introduce a simple convention for individual metrics and SLA parameter names of particular operations at runtime:

To refer to a metric or SLA parameter of a particular operation at runtime that has been defined as part of an operation group definition, we prefix individual metric and SLA parameter names with the name of the particular operation.

In the example given above: The SLA parameter `"AverageResponseTime"` in the operations group definition is referred to as "`WSDLSOAPGetStockQuote:AverageResponseTime"` for the operation `"WSDLSOAPGetStockQuote"`.

2.4.13. Metric Macro Definition

A `MetricMacroDefinition` is a definition of a set of, potentially related, metrics in a way that this definition can be reused multiple times for different "resource" metrics. A typical use case is as follows: An SLA is defined for two types of operations, fast and slow ones, each of which is captured in an operation group. For each operation group, an SLA parameter is defined that is based on the same structure of metrics, that is, resource metrics are read at the same interval, put in time series and, for example, an average is taken. Although the structure of the metrics is the same, the resource measurements are taken from different sources. Rather than defining the whole measurement structure multiple times, parties to a contract can define a metric macro and declare some measurement directives as to be filled in when the macro is expanded. The macro can be expanded, for example, in the operation groups of the fast and slow operations. In this expansion, it must be declared which measurement directives should be used.

Type Definition

```
<xsd:complexType name="MetricMacroDefinitionType">
  <xsd:sequence>
      <xsd:element name="Metric"
          type="wsla:MetricType"
          maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

- A `MetricMacroDefinition` contains any number of metrics. The metric definitions may contain `MeasurementDirectiveVariable` definitions.

Example

```
<MetricMacroDefinition name="TheMetricTree">

     <Metric name="AverageResponseTimeLastHour" type="double"
               unit="sconds / transactions">
         <Source>Provider</Source>
         <Function xsi:type="Divide" resultType="double">
               <Operand>
                 <Metric>ResponseTime</Metric>
               </Operand>
               <Operand>
                 <Metric>Transactions</Metric>
               </Operand>
         </Function>
     </Metric>

     <!-- Compute the number of transactions in the last hour.
     -->
     <Metric name="Transactions" type="long" unit="transactions">
         <Source>Provider</Source>
         <Function xsi:type="Minus" resultType="double">
           <Operand>
               <Function xsi:type="TSSelect" resultType="long">
                 <Operand>
                   <Metric>SumTransactionTimeSeries</Metric>
                 </Operand>
                 <Element>0</Element>
               </Function>
           </Operand>
           <Operand>
               <Function xsi:type="TSSelect" resultType="long">
                 <Operand>
                   <Metric>SumTransactionTimeSeries</Metric>
                 </Operand>
                 <Element>-60</Element>
               </Function>
           </Operand>
         </Function>
     </Metric>

     <!-- Get a reading of the current number of transactions.
          Keep the last 61 readings.
     -->
     <Metric name="SumTransactionTimeSeries" type="TS" unit="transactions">
         <Source>Provider</Source>
         <Function xsi:type="TSConstructor" resultType="TS">
             <Schedule>theschedule</Schedule>
             <Metric>SumTransactions</Metric>
             <Window>61</Window>
         </Function>
     </Metric>

     <!-- Compute the time spent processing request in the last hour.
     -->
     <Metric name="ResponseTime" type="long" unit="milliseconds">
         <Source>Provider</Source>
         <Function xsi:type="Minus" resultType="double">
             <Operand>
               <Function xsi:type="TSSelect" resultType="long">
                 <Operand>
                   <Metric>SumResponseTimeTimeSeries</Metric>
                 </Operand>
```

```
                <Element>0</Element>
              </Function>
            </Operand>
            <Operand>
              <Function xsi:type="TSSelect" resultType="long">
                <Operand>
                  <Metric>SumResponseTimeTimeSeries</Metric>
                </Operand>
                <Element>-60</Element>
              </Function>
            </Operand>
        </Function>
    </Metric>

    <!-- Get a reading of the new aggregated response time every minute.
         Keep the last 61 readings.
    -->
    <Metric name="SumResponseTimeTimeSeries" type="TS" unit="milliseconds">
        <Source>Provider</Source>
        <Function xsi:type="TSConstructor" resultType="TS">
            <Schedule>theschedule</Schedule>
            <Metric>SumResponseTime</Metric>
            <Window>61</Window>
        </Function>
    </Metric>

    <!-- It is assumed the resource measurements are gained from the log and
         there is no further specification needed how exactly to get them
    -->
    <Metric name="SumTransactions" type="long" unit="transactions">
        <Source>Provider</Source>
        <MeasurementDirectiveVariable"/>
    </Metric>

    <Metric name="SumResponseTime" type="long" unit="milliseconds">
        <Source>Provider</Source>
        <MeasurementDirectiveVariable"/>
    </Metric>

  </MetricMacroDefinition>
```

The example defines a structure of metrics based on the resource metric “SumTransactions” and “SumResponseTime”. Those resource metrics declare their measurement directives as variable.

2.4.14. Metric Macro Expansion

A `MetricMacroExpansion` defines a set of metrics by referring to a metric macro definition and providing measurement directives for those metrics in the macro that contain measurement directive variables rather than measurement directives. This macro expansion is equivalent to a (repeated) definition of the metrics contained in the macro, the measurement directive variables replaced by the proper measurement directives.

Type Definition

```
<xsd:complexType name="MetricMacroExpansionType">
  <xsd:sequence>
      <xsd:element name="MeasurementDirectiveAssignment"
          type="wsla:MDAssignmentType"
          minOccurs="0"
          maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="macroName" type="xsd:string"/>
```

```
  <xsd:attribute name="expansionName" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="MetricMacroExpansion" type="wsla:MetricMacroExpansionType"/>
```

- A `MetricMacroDefinition` contains any number of `MeasurementDirectiveAssignments`, for each of the metrics of the macro that uses measurement directive variables.
- The attribute `macroName` refers to the metric macro definition that is being expanded.
- The `expansionName` is the unique name assigned to this expansion.

Example

```
<MetricMacroExpansion macroName="TheMetricTree"
                      expansionName="Fast">

        <MeasurementDirectiveAssignment metricName="SumResponseTime">
          <MeasurementDirective xsi:type="ResponseTimeCounter"
                                resultType="long"/>
               <!—detailed measurement directive →
        </MeasurementDirectiveAssignment>

        <MeasurementDirectiveAssignment metricName="SumTransactions">
          <MeasurementDirective xsi:type="TransactionCounter"
                                resultType="long"/>
               <!—detailed measurement directive →
        </MeasurementDirectiveAssignment>

      </MetricMacroExpansion>
```

The example expands the macro “MetricTree” defined in the section above. It assigns measurement directives to the metrics “SumResponseTime” and “SumTransactions”.

Naming Convention

Each metric must have a unique name. A metric defined in the context of a macro expansion is referred to by the name `<expansionName:metricName>` where `expansionName` is the unique name of the metric macro expansion and `metricName` is the name of the metric given in the metric macro definition.

2.4.15. Measurement Directive Assignment

A `MeasurementDirectiveAssignment` associates the name of a metric in a metric macro definition with a measurement directive. It is used in Metric Macro Expansions.

Type Definition

```
<xsd:complexType name="MDAssignmentType">
  <xsd:sequence>
      <xsd:element name="MeasurementDirective"
          type="wsla:MeasurementDirectiveType"/>
  </xsd:sequence>
  <xsd:attribute name="metricName" type="xsd:string"/>
</xsd:complexType>
```

- A `MeasurementDirective` is a measurement directive for the metric.
- The attribute `metricName` refers to the metric of the metric macro definition that must be measured according to the measurement directive.

2.4.16. Service Level Objective

A service level objective expresses a commitment to maintain a particular state of the service in a given period. The state is defined as an expression on predicates that refer to the SLA Parameters defined in the Service Definition section of the SLA. Any party can take the obliged part of this guarantee. However, this is typically the service provider.

Type Definition

```
<xsd:complexType name="ServiceLevelObjectiveType">
  <xsd:sequence>
    <xsd:element name="Obliged" type="xsd:string" />
    <xsd:element name="Validity" type="wsla:PeriodType" maxOccurs="unbounded"/>
    <xsd:element name="Expression" type="wsla:LogicExpressionType"/>
    <xsd:choice>
      <xsd:element name="EvaluationEvent" type="wsla:EvaluationEventType"/>
      <xsd:element name="Schedule" type="xsd:string"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

- The `Obliged` element is a reference to a party that is in charge of delivering what is promised in this obligation.
- A service level objective has one or many `Validity` periods, defining when the guarantee is applicable. (♠ Multiple validity periods are not supported in the current WSLA Compliance Monitor Implementation.)
- An `Expression` of the `LogicExpressionType` defines the actual content of the obligation, that is, what is asserted by the service provider to the service customer.
- A service level objective may have an `EvaluationEvent` associated. It defines the case in which the expression of the service level objective is to be evaluated.
- Alternatively, the expression may be evaluated according to a `Schedule`. In this case, the `Schedule` element contains a reference to a schedule defined in the service definition section of the SLA.
- The attribute `name` assigns a unique name to each guarantee.

Example

```
<ServiceLevelObjective name="g1">
  <Obliged>provider</Obliged>
  <Validity>
     <Start>2001-11-30T14:00:00.000-05:00</Start>
     <End>2001-12-31T14:00:00.000-05:00</End>
  </Validity>
  <Expression>
    <Predicate xsi:type="wsla:Less">
      <SLAParameter>AverageResponseTime</SLAParameter>
      <Value>5</Value>
    </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

The example shows a service level objective that has one validity period and an expression that contains a predicate of type `"wsla:Less"` defining an average response time guarantee of less than 5 seconds. In the case of this example, `"AverageResponseTime"` is a reference to an

`SLAParameter` defined in the service definition section of the SLA. The expression is evaluated according to the `EvaluationEvent "NewValue"`.

2.4.17. Action Guarantee

An action guarantee expresses a commitment to perform a particular activity if a given precondition is met. The precondition is defined as an expression on predicates that refer to the SLA Parameters defined in the Service Definition section of the SLA. Any party can be the obliged of this kind of guarantee. This particularly includes the supporting parties of the contract and the service customer.

Type Definition

```
<xsd:complexType name="ActionGuaranteeType">
  <xsd:sequence>
    <xsd:element name="Obliged" type="xsd:string"
                 maxOccurs="unbounded"/>
    <xsd:element name="Expression" type="wsla:LogicExpressionType"/>
    <xsd:choice>
      <xsd:element name="EvaluationEvent"
                   type="wsla:EvaluationEventType"/>
      <xsd:element name="Schedule" type="xsd:string"/>
    </xsd:choice>
    <xsd:element ref="wsla:QualifiedAction" maxOccurs="unbounded"/>
    <xsd:element name="ExecutionModality"
                 type="wsla:ExecutionModalityType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

- The `Obliged` is a reference to a party that is in charge of delivering what is promised in this guarantee.
- An `Expression` of the `LogicExpressionType` defines the precondition of the action.
- An action level guarantee may have an `EvaluationEvent` associated with it. It defines the case in which the expression of the service level guarantee is to be evaluated.
- Alternatively, the expression may be evaluated according to a `Schedule`. In this case, the `Schedule` element contains a reference to a schedule defined in the service definition section of the SLA.
- The `QualifiedAction` contains a definition of the action to be performed at a particular party. One or more qualified actions can be part of an action guarantee.
- The `ExecutionModality` of type `ExecutionModalityType` is an additional means to control the execution of the action. It can be defined whether the action should be executed if a particular evaluation of the expression yields true. The purpose is to reduce, for example, the execution of a notification action to a necessary level if the associated expression is evaluated very frequently but the notification only needs to be sent the first time the expression evaluates to true.
- The attribute `name` assigns a unique name to each guarantee.

Example

```
<ActionGuarantee name="ga3">
     <Obliged>ZAuditing</Obliged>
     <Expression>
       <Predicate xsi:type="Violation">
         <ServiceLevelObjective>ga1</ServiceLevelObjective>
```

```
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>XInc</Party>
        <Action actionName="notification" xsi:type="Notification">
          <NotificationType>Violation</NotificationType>
          <CausingGuarantee>ga1</CausingGuarantee>
          <SLAParameter>Availability_UpTimeRatio</SLAParameter>
        </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>
```

The example shows an action guarantee that obliges a party `"ZAuditing"`, a condition evaluation service in this case, to execute a notification action on the party `"XInc"` if the expression holds. In our case the expression contains a predicate of type `"wsla:Violation"` relating to the service level guarantee `"ga1"`. In other words: If a violation of `"ga1"` occurs, a notification will be sent to `"XInc"`. The expression is evaluated according to the `EvaluationEvent "NewValue"`. The execution modality is `"Always"`, that is, each time a violation condition is evaluated a notification will be sent. For details on the marshaling of an action (`NotificationType`, `CausingGuarantee`, `SLAParameter`, in this case), refer to section3.7.1.

2.4.18. Obligation Group

A named `ObligationObjectType` is an `ObligationGroup`. As any `ObligationObject`, it can contain any number of service level objectives and action guarantees.

Type Definition

```
<xsd:complexType name="ObligationGroup">
  <xsd:complexContent>
    <xsd:extension base="wsla:ObligationObjectType">
    </xsd:extension>
  </xsd:complexContent>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="ObligationGroup" type="wsla:ObligationGroup"/>
```

- `name` is a unique name of this ObligationGroup.

An obligation group is an obligation object with a name assigned to it. Obligations may be grouped for a number of reasons, including to facilitate the creation of templates containing groups of guarantees, to associate financial penalties with groups of guarantees, and to create “classes of service” to facilitate the creation of run-time scheduling or configuration policies.

2.4.19. Obligations

The complete set of obligations of an SLA is captured in the `ObligationsType`. The whole set of obligations may contain service level objectives, action guarantees, and obligation groups.

Type Definition

```
<xsd:complexType name="ObligationsType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ObligationObjectType">
      <xsd:sequence>
        <xsd:element ref="wsla:ObligationGroup" minOccurs="0"
                                                maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
```

```
  </xsd:complexContent>
</xsd:complexType>
```

- This type extends the `ObligationObjectType` by any number of ObligationGroups that it may contain.

2.4.20. Logic Expression

A logic expression defines a condition and is a key part of an obligation. Logic expressions follow first order logic, including predicates and logic operators such as `AND` and `NOT` but no quantifiers.

Type Definition

```
<xsd:complexType name="LogicExpressionType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:element name="Predicate" type="wsla:PredicateType"/>
      <xsd:element name="And" type="wsla:BinaryLogicOperatorType"/>
      <xsd:element name="Or" type="wsla:BinaryLogicOperatorType"/>
      <xsd:element name="Not" type="wsla:UnaryLogicOperatorType"/>
      <xsd:element name="Implies" type="wsla:BinaryLogicOperatorType"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

- The simplest form a logic expression is a plain `Predicate` of type `PredicateType`.
- The conjunction `AND` of type `BinaryLogicOperatorType` connects two logic expressions.
- The disjunction `OR` of type `BinaryLogicOperatorType` also connects two logic expressions.
- The unary `NOT` negates an expression. Type `UnaryLogicOperatorType`.
- The implication `IMPLIES` is equivalent to the logic subjunction (`A IMPLIES B` is the same as `NOT A OR B`). Type `BinaryLogicOperatorType`.

Example

```
       <Expression>
        <Or>
          <Expression>
            <Predicate xsi:type="Less">
              <SLAParameter>ResponseTimeThroughPutRatio</SLAParameter>
              <Value>0.5</Value>
            </Predicate>
          </Expression>
          <Expression>
            <Predicate xsi:type="Greater">
              <SLAParameter>TransactionRate</SLAParameter>
              <Value>10000</Value>
            </Predicate>
          </Expression>
        </Or>
      </Expression>
```

The example shows an expression stating that the response time must be less than 0.5 unless the transaction rate is greater than 10000.

2.4.21. Binary Logic Operator

A binary logic operator connects two expressions.

Type Definition

```
<xsd:complexType name="BinaryLogicOperatorType">
  <xsd:sequence>
      <xsd:element name="Expression" type="wsla:LogicExpressionType"
                   minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:complexType>
```

- `Each Expression` of the binary operator is again of type `LogicExpressionType`. There are exactly two expressions.

Example

See logic expression type.

2.4.22. Unary Logic Operator

A unary logic operator relates to one logic expression.

Type Definition

```
<xsd:complexType name="UnaryLogicOperatorType">
  <xsd:sequence>
      <xsd:element name="Expression" type="wsla:LogicExpressionType"/>
  </xsd:sequence>
</xsd:complexType>
```

- `The Expression` of the unary operator is again of type `LogicExpressionType`. There is exactly one expression.

Example

See logic expression type.

2.4.23. Predicates

A predicate is a function that returns true or false. It captures the assertions with respect to an `SLAParameter` that are given in a Guarantee. As it is the case with measurement directives and functions, there are a large number of predicates that may vary in each context. We follow the approach of an abstract type that can and must be extended to fit the needs of a particular application domain. However, a set of common predicates such as `Less`, `Greater`, `Equal`, etc. are defined in the standard extensions of section 3.

Type Definition

```
<xsd:complexType name="PredicateType" abstract="true"/>
```

This abstract type contains no elements or attributes. It is a common umbrella for all specific predicate type, which can be special to needs of a particular domain.

2.4.24. Qualified Action

A QualifiedAction defines an action to be invoked at a particular party as part of a guarantee.

Type Definition

```
<xsd:complexType name="QualifiedActionType">
  <xsd:sequence>
    <xsd:element name="Party" type="xsd:string"/>
    <xsd:element name="Action" type="wsla:ActionInvocationType"/>
 </xsd:sequence>
</xsd:complexType>
```

```
<xsd:element name="QualifiedAction" type="wsla:QualifiedType">
  <xsd:keyref name="partyref" refer="wsla:partyKey">
    <xsd:selector xpath="."/>
    <xsd:field xpath="Party"/>
  </xsd:keyref>
</xsd:element>
```

- `Party` refers to the party of the SLA on which the `Action` should be called.
- `Action` refers to an action invocation of this party, defining which action to be called and how the parameters are to be marshaled.

Both elements are keyrefs to keys defined on parties and actions.

2.4.25. Action Invocation

An `ActionInvocationType` defines how an Action is to be invoked as part of a `QualifiedAction` of an `ActionGuarantee`. This refers to the marshaling of parameters of this action. Marshaling parameters is different for each particular type of operation because different operation support different parameter sets. Therefore, `ActionInvocationType` is a supertype that needs to be extended for particular actions used between organizations, if there is ambiguity on which parameters should be expected. An `ActionInvocation` refers to a particular `ActionDefinition` of a particular party.

Remark: At a glance, it appears odd to define an additional structure to "use" actions in guarantees, similar to its use in programming languages. However, while WSDL provides us with a means to define signatures and descriptions of signature implementation and wire formats such as SOAP define the transport representation at runtime, "using" an action in what appears to be similar to a programming context requires yet another syntax to describe how to marshal parameters in this particular case. In a programming environment the programming language deals with this. The analog in Java/RMI: The Java interface syntax describes method signatures. RMI defines a wire format for the transport of method invocations and their parameters from client to server and vice versa. The actual parameterization of a method call with constants or variables of the environment of the method invocation are described in the Java programming language.

We did not find a reasonable algorithmic language in an XML syntax from which we could borrow, and we did not want to invent one for this purpose. Thus, we define a specific invocation syntax for each action as a "cheaper" approach.

Type Definition

```
<xsd:complexType name="ActionInvocationType">
  <xsd:attribute name="actionName" type="xsd:string"/>
</xsd:complexType>
```

- `actionName` is an attribute that refers to an `ActionDefinition` of a party in the `Parties` section of the SLA.

An `ActionInvocation` extension for the standard operation "Notification" is defined in section 3.7.1 of this document.

Example

```
<Action actionName="notification" xsi:type="Notification">
   <NotificationType>Information</NotificationType>
   <CausingGuarantee>g3</CausingGuarantee>
   <SLAParameter>ResponseTimeThroughPutRatio TransactionRate</SLAParameter>
</Action>
```

The example shows the usage of the Notification action in a guarantee specification. See 3.7.1 for details of the `"Notification"ActionInvocationType` and Appendix 4 for its interface

specification.

2.5. Additional Data Types

2.5.1. Period Type ♠

A period defines a time span. It is used in other types such as the Schedule.

Type Definition

```
<xsd:complexType name="PeriodType">
  <xsd:choice>
  <xsd:sequence>
    <xsd:element name="Start" type="xsd:dateTime"/>
    <xsd:element name="End" type="xsd:dateTime"/>
  </xsd:sequence>
  <xsd:sequence>
    <xsd:element name="ConditionTime" type="xsd:string"/>
    <xsd:element name="ConditionMonthOfYearMask" type="xsd:string"/>
    <xsd:element name="ConditionDayOfMonthMask" type="xsd:string"/>
    <xsd:element name="ConditionDayOfWeekMask" type="xsd:string"/>
    <xsd:element name="ConditionTimeOfDayMask" type="xsd:string"/>
    <xsd:element name="ConditionTimeZone" type="xsd:string"/>
    <xsd:element name="ConditionLocalOrUtcTime" type="xsd:string"/>
  </xsd:sequence>
  </xsd:choice>
</xsd:complexType>
```

The `PeriodType` enables specifications of either simple time periods (first sequence) or more sophisticated validity periods, as defined by the IETF RFC 3060 [11].

In the simple case WSLA foresees two elements:

- `Start` is the point in time when the period begins, in the date time format of the standard xsd schema.
- At `End` the period is over.

The IETF RFC 3060-based way of specifying time periods is described in detail in [11].

Example

```
   <Validity>
    <StartDate>2001-08-15:1400</StartDate>
    <EndDate>2001-09-15:1400</EndDate>
   </Validity>
```

This `PeriodType`, used in a `Validity` element of a `Guarantee`, starts at August 15, 2001, 2pm and ends at September 15, 2001 at the same time, according to the simple time period model.

2.5.2. Type

This type `"Type"` comprises the set of eligible types that `SLAParameters` and `Metrics` can assume and that `Functions` and `MeasurementDirectives` produce. The type set is very similar to the types of `xsd:simpleType`, which the XSL uses, with some additions to deal with time series.

Type Definition

```
<xsd:simpleType name="Type">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="TS"/>
```

```
    <xsd:enumeration value="Q"/>
    <xsd:enumeration value="integer"/>
    <xsd:enumeration value="float"/>
    <xsd:enumeration value="double"/>
    <xsd:enumeration value="long"/>
    <xsd:enumeration value="byte"/>
    <xsd:enumeration value="boolean"/>
    <xsd:enumeration value="string"/>
    <xsd:enumeration value="time"/>
  </xsd:restriction>
</xsd:simpleType>
```

The `Type` is an enumeration of strings that represent types commonly known from programming languages or the XML Schema Language itself. Technically, it is a restriction of the string type.

- `TS` is a time series that contains values taken at equidistant intervals.
- `Q` indicates a queue of values ordered by time of entry, but not necessarily taken at equidistant intervals.
- `integer` as defined in XSD.
- `float` as defined in XSD.
- `double` as defined in XSD.
- `long` as defined in XSD.
- `byte` as defined in XSD.
- `boolean` as defined in XSD.
- `string` as defined in XSD.
- `time` as defined in XSD.

2.5.3. Interval Type

A period defines a time span at the required granularity.

Type Definition

```
<xsd:complexType name="IntervalType">
      <xsd:sequence>
        <xsd:element name="Hours" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="Minutes" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="Seconds" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="MilliSeconds" type="xsd:integer"
                     minOccurs="0"/>
      </xsd:sequence>
</xsd:complexType>
```

The suitable elements can be chosen to represent an amount of time.

- `Hours` are the number of hours.
- `Minutes` are the number of minutes.
- `Seconds` are the number of seconds.
- `MilliSeconds` are the number of milliseconds.

Example

```
    <Interval>
      <Minutes>2</Minutes>
      <Seconds>30</Seconds>
    </Interval>
```

2.5.4. Evaluation Event

An Evaluation Event is used for the definition of an event that triggers the evaluation of an expression.

Type Definition

```
<xsd:simpleType name="EvaluationEventType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="NewValue"/>
  </xsd:restriction>
</xsd:simpleType>
```

There is only one `EvaluationEventType` defined in this specification:

- `NewValue` means evaluation must be performed whenever a new value is available.

Other `EvaluationEventType` can be added by extending this base type.

Example

```
 <ActionGuarantee name="g2">
          <Obliged>ms</Obliged>
   <Expression>
     <Predicate xsi:type="wsla:Violation">
              <ServiceLevelObjective>g1</ServiceLevelObjective>
     </Predicate>
   </Expression>
   <EvaluationEvent>NewValue</EvaluationEvent>
   <Action>
       ....
 </ActionGuarantee>
```

The evaluation event defines when the expression must be evaluated.

2.5.5. Execution Modality

The execution modality of an action defines an additional precondition of the execution of an action in an action guarantee, if the expression associated with the action guarantee was evaluated to true. The purpose is to control how often activities are executed.

Type Definition

```
<xsd:simpleType name="ExecutionModalityType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Always"/>
    <xsd:enumeration value="OnEnteringCondition"/>
    <xsd:enumeration value="OnEnteringAndOnLeavingCondition"/>
    <xsd:enumeration value="OnEveryEvaluation"/>
  </xsd:restriction>
</xsd:simpleType>
```

The different types carry the following semantics:

- `Always` means that an activity is executed each time its precondition is met.
- `OnEnteringCondition` defines that an activity is executed the first its precondition is met. When the precondition is not met for the first time after the execution of an activity, the state is reset and the activity is executed the next time the precondition holds.
- `OnEnteringAndOnLeavingCondition` defines that an activity is executed the first time a precondition is met and then again when the precondition is not met for the first time and so on, i.e. each time the trueness of the precondition changes.

- `OnEveryEvaluation` defines that an activity is executed even if the precondition fails. This is interesting, for example, if notifications must be sent regardless of the occurrence of a violation.

Example

```
<ActionGuarantee name="g2">
  <Obliged>ms</Obliged>
  <Expression>
    <Predicate xsi:type="wsla:Violation">
      <ServiceLevelObjective>g1</ServiceLevelObjective>
    </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
  <Action>
     <Party>customer</Party>
     <Action>Notification</Action>
  </Action>
  <ExecutionModality>Always</ExecutionModality>
</ActionGuarantee>
```

In the example case the action has to be executed each time the expression becomes true.

Remark: Other, more sophisticated models exist, e.g. in the area of network management [9]. Filters for alarms are defined using a rising and a falling threshold, thus being more "robust" than the simple model introduced here. These advanced models can be introduced by subclassing the `ExecutionModalityType`.

2.5.6. Operand

Defines the set of types that are operands to functions.

Type Definition

```
<xsd:complexType name="OperandType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
          <xsd:element name="LongScalar" type="xsd:long"/>
          <xsd:element name="Scalar" type="xsd:float"/>
          <xsd:element name="StringScalar" type="xsd:string"/>
          <xsd:element name="BoolScalar" type="xsd:string"/>
        </xsd:choice>
      </xsd:sequence>
</xsd:complexType>
```

- `Metric` is the name of a metric to which is referred. This is to be interpreted that the `Value` of the metric is used for computation.
- A `Function` can be an operand of another function.
- Operands can be just scalar values of particular type such as a `LongScalar`.
- `FloatScalar` is a simple value of type float.
- `StringScalar` is a simple value of type string.

Example

```
         <Function xsi:type="Minus" resultType="double">
              <Operand>
                <LongScalar>1</LongScalar>
```

```
            </Operand>
            <Operand>
              <Function xsi:type="Divide" resultType="long">
                <Operand>
                  <Function xsi:type="ValueOccurs" resultType="long">
                    <Metric>StatusTimeSeries</Metric>
                    <Value>
                      <LongScalar>0</LongScalar>
                    </Value>
                  </Function>
                </Operand>
                <Operand>
                  <LongScalar>1440</LongScalar>
                </Operand>
              </Function>
            </Operand>
          </Function>
```

This is equivalent to 1 - (ValueOccurs(StatusTimeSeries,0) / 1440), in a typical arithmetic representation.

2.5.7. Lists

A number of lists of basic types are available: StringList, IntegerList and FloatList.

Type Definition

```
<xsd:simpleType name="StringList">
  <xsd:list itemType="xsd:string"/>
</xsd:simpleType>
<xsd:simpleType name="IntegerList">
  <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
<xsd:simpleType name="FloatList">
  <xsd:list itemType="xsd:float"/>
</xsd:simpleType>
```

3. Standard Extensions

3.1. Extension Mechanism and Purpose of Standard Extensions

WSLA provides a mechanism to extend the agreement language beyond its core defined in section 2. This is important to adapt to different expressive needs, e.g. to include operation descriptions according to different description standards or to include domain specific vocabulary. WSLA relies on the extension mechanism of the XML Schema Definition language. This is useful not only to introduce new syntactical symbols of WSLA but also to define the semantics of the new symbols as derivations of standard abstract language elements.

A number of types in the core WSLA language in section 3 are defined as `abstract`. This means these types need to be extended to become useful for actually defining an SLA. All these abstract types are necessarily related to a particular technology or a particular expressive need. These abstract types are:

- **OperationDescription**. The behavioral properties of an operation largely depend on its particular implementation, e.g., a WSDL defined SOAP-encoded operation over http.

- **ActionDescription**. The relevant attributes to describe an action (a management activity) also mainly depend on the particular implementation details, similar to the OperationDescription.

- **MeasurementDirective**. Relevant properties of a MeasurementDirective depend on the particular system to be measured, the parameters that the system exposes and the method of measurement. In particular, Measurement Directives are the glue to access measurable properties exposed from the instrumentation of a managed resource. Since the ways of retrieving these properties may vary across different environments, different MeasurementDirective types will be used to accommodate these.

- **Function**. Functions are used to derive high-level metrics from simpler ones. The functions needed depend on the types of metrics that are relevant in each particular case. Functions can be either general mathematical functions or domain specific ones.

- **Predicate**. Similar to functions, predicates are applied to metrics. The particular predicates needed depend on the metrics that occur in a specific case. There will be frequently used common predicates such as greater, less and equal on numeric values but also domain specific predicates.

- **ActionInvocation.** For each type of action in the contract, e.g. Notification, ParameterUpdate, etc., we require a corresponding ActionInvocation type that allows the parties to describe in guarantees how to marshal a particular action. There is a standard set of actions that occur frequently, which can be extended by ActionInvocation subtypes for domain-specific actions.

Typically, a group of contracting organizations will agree on the relevant extensions to be used. However, the definition of these extensions is an extra effort that should be minimized. We want to provide the users of the WSLA language with a useful set of language elements that covers the most common needs of organization. Structurally, we could compare this to the C programming language. While the core C syntax is relatively compact, the user is supplied with a useful common set of functions in the libC library. In this section we introduce the WSLA language's standard extensions.

3.2. Operation Descriptions

3.2.1. WSDL SOAP Operation Description

WSDL SOAP operation description extends the OperationDescriptionType. It adds relevant parameters to refer to an operation defined in a WSDL file as part of a specific binding.

Type Definition

```
<xsd:complexType name="WSDLSOAPOperationDescriptionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:OperationDescriptionType">
      <xsd:sequence>
        <xsd:element name="WSDLFile" type="xsd:anyURI"/>
        <xsd:element name="SOAPBindingName" type="xsd:ID"/>
        <xsd:element name="SOAPOperationName" type="xsd:ID"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `WSDLFile` refers to the location of the file that contains the operation's definition.
- `SOAPBindingName` refers to the particular binding of the operation that is meant.
- `SOAPOperationName` is the name of the operation in its binding.

Example

```
 <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType"
    name="WSDLSOAPGetQuote">
   <SLAParameter name="AverageResponseTime"
                 type="float"
                 unit="seconds">
     <Metric>averageResponseTime</Metric>
   </SLAParameter>

   <Metric name="averageResponseTime" type="double" unit="seconds">
     <Source>support1</Source>
       <MetricURI>http://support1.com/getMetric?averageResponseTime
       </MetricURI>
       <Function xsi:type="wsla:TextExpression" resultType="double">
           (averageResponseTimeHost1,averageResponseTimeHost2)/2
       </Function>
   </Metric>
   <Metric name="averageResponseTimeHost1" type="double"
           unit="seconds">
       <Source>support1</Source>
       <MetricURI>
          http://support1.com/getMetric?averageResponseTimeHost1
       </MetricURI>
       <Function xsi:type="wsla:Average" resultType="double">
         <Metric>responseTimesHost1</Metric>
       </Function>
   </Metric>
   ....
   <!-- This is the specific part of the WSDLSOAPOperation -->
    <WSDLFile>StockQuoteService.wsdl</WSDLFile>
    <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
    <SOAPOperationName>getQuote</SOAPOperationName>
 </Operation>
```

The example shows an operation getQuote that has an SLA Parameter and its metrics defined. At the end of the definition we find the additional parameters.

3.3. Action Descriptions

3.3.1. WSDL SOAP Action Description

WSDL SOAP action description extends the ActionDescriptionType in a way similar to a WSDL SOAP operation description.

Type Definition

```
<xsd:complexType name="WSDLSOAPActionDescriptionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ActionDescriptionType">
      <xsd:sequence>
        <xsd:element name="WSDLFile" type="xsd:anyURI"/>
        <xsd:element name="SOAPBindingName" type="xsd:ID"/>
        <xsd:element name="SOAPOperationName" type="xsd:ID"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `WSDLFile` refers to the location of the file that contains the action's definition.
- `SOAPBindingName` refers to the particular binding of the action that is meant.
- `SOAPOperationName` is the name of the action in its binding.

Example

```
 <Action name="Notification" xsi:type="WSDLSOAPActionDescription">
   <WSDLFile>Notification.wsdl</WSDLFile>
   <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
   <SOAPOperationName>Notify</SOAPOperationName>
 </Action>
```

In this example, the action `"Notification"` is defined in the WSDL-file `"Notification.wsdl"`, the binding is the `"SOAPNotificationBinding"` and the SOAPOperation that implements the action is `"Notify"`.

3.4. Measurement Directives

3.4.1. Counter

This measurement directive describes the relevant information to retrieve a counter from the instrumentation of a service or managed resource. A counter is a non-negative integer that may be incremented, but not decremented. Possible maximum values are 2^{32} –1 (in case of a 32-bit integer) or 2^{64} –1 (if a 64-bit integer is used); when the counter reaches its maximum, it wraps around and starts increasing again from 0. Such counters, also known as rollover counters, are typically applied to count e.g., the number of requests issued, or the packets/octets sent or received. The only way for a measurement service or probe to keep track of wraparounds is to retrieve the value of the counter in appropriately intervals. Because 32-bit and 64-bit counters are widely used, this should not have to be done very often. Typically, the difference between two values of a counter is computed periodically and may be collected into a time series.

Type Definition

```
<xsd:complexType name="Counter">
  <xsd:complexContent>
```

```
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="MeasurementURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `MeasurementURI` is the URI to be used to send a probe to the system.

Example

```
    <MeasurementDirective xsi:type="wsla:Counter"
                          resultType="double">
       <MeasurementURI>http://support1.com/ipPacketsIn
       </MeasurementURI>
    </MeasurementDirective>
```

In the given example, the result type of the measurement directive is double. `resultType` is a common attribute of a measurement directive. The probes use the URI `http://support1.com/ipPacketsIn`.

3.4.2. Gauge

This measurement directive describes the relevant information to retrieve a gauge from the instrumentation of a service or managed resource. A gauge is a non-negative integer that may increase or decrease, with a maximum value of either 2^{32} –1 (in case of a 32-bit integer) or 2^{64} –1 (if a 64-bit integer is used). Typically, a gauge is used to measure the current value of some entity, such as the current number of requests stored in a queue. In contrast to a counter, a gauge does not roll over to zero once its maximum value is reached. Rather, if the gauge represents a value that increases beyond the maximum, the gauge remains stuck at the maximum value. If the represented value subsequently falls below the gauge maximum, the gauge is allowed to decrease to reflect the same value as the modeled value. Gauges are evaluated regularly and may be collected into a time series.

Type Definition

```
<xsd:complexType name="Gauge">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="MeasurementURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `MeasurementURI` is the URI to be used to send a probe to the system.

Example

```
    <MeasurementDirective xsi:type="wsla:Gauge"
                          resultType="double">
       <MeasurementURI>http://support1.com/numRequestsInQueue
       </MeasurementURI>
    </MeasurementDirective>
```

In the given example, the result type of the measurement directive is double. `resultType` is a common attribute of a measurement directive. The probes use the URI `http://support1.com/numRequestsInQueue` to measure the current number of queued requests.

3.4.3. Response Time

This measurement directive describes the relevant information to measure the response time. We assume that response time is measured by probing. Typically, response times are evaluated regularly and collected into a time series.

Type Definition

```
<xsd:complexType name="ResponseTime">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="MeasurementURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `MeasurementURI` is the URI to be used to send a probe to the system.

Example

```
    <MeasurementDirective xsi:type="wsla:ResponseTime"
                          resultType="double">
       <MeasurementURI>http://support1.com/testResponse
       </MeasurementURI>
    </MeasurementDirective>
```

In the given example, the result type of the measurement directive is double. `resultType` is a common attribute of a measurement directive. The probes use the URI `http://support1.com/testResponse`.

3.4.4. Invocation Count

This measurement directive describes the relevant information to read the invocation count of a system, i.e. the number of usages of a particular operation. This is typically used to derive the throughput of a system by computing the increment from one measurement to the next.

Type Definition

```
<xsd:complexType name="InvocationCount">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="CounterURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `CounterURI` is the URI where the counter can be read.

Example

```
    <MeasurementDirective xsi:type="wsla:InvocationCount"
                          resultType="integer">
       <CounterURI>http://party1.com/counter</CounterURI>
    </MeasurementDirective>
```

In the given example, the result type of the measurement directive is integer, the number of invocations of our stock quote service. The counter is read at URI `http://party1.com/counter`.

3.4.5. Status

This measurement directive describes the relevant information to measure the status of a service, i.e. whether it is alive or not. We assume that status is measured by probing.

Type Definition

```
<xsd:complexType name="Status">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="MeasurementURI" type="xsd:anyURI"/>
        <xsd:element name="TimeOut" type="wsla:Interval"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `MeasurementURI` is the URI to be used to send a probe to the system.
- `TimeOut` is the time waiting for a response. If the time out is exceeded it is assumed that the system is not alive.

Example

```
    <MeasurementDirective xsi:type="wsla:Status"
                          resultType="boolean">
       <MeasurementURI>http://support1.com/testStatus</MeasurementURI>
       <TimeOut>
         <Seconds>5<Seconds>
       </TimeOut>
    </MeasurementDirective>
```

In the given example, the probes use the URI `http://support1.com/testStatus`. The timeout value is 5 seconds.

3.4.6. Status Request

Request the status of service object from a measurement interface. The metric assumes the value 0 if the service object is down and 1 if it is up.

Type Definition

```
<xsd:complexType name="StatusRequest">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="RequestURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `RequestURI` is the URI to be used to send a probe to the system.

Example

```
<Metric name="MeasuredStatus" type="integer" unit="">
  <Source>YMeasurement</Source>
  <MeasurementDirective xsi:type="StatusRequest" resultType="integer">
    <RequestURI>http://ymeasurement.com/StatusRequest/GetQuote
    </RequestURI>
  </MeasurementDirective>
</Metric>
```

In the given example, the Metric `"MeasuredStatus"` is measured based on a measurement directive of type `StatusRequest`. The measurement is obtained at the URI `http://ymeasurement.com/StatusRequest/GetQuote`.

3.4.7. Downtime

The measurement directive `Downtime` refers to the metric that can be obtained in the form of a high-level reading from systems with advanced instrumentation. Rather than deriving downtime as a derived metrics of time series of a system's status, some systems offer this reading directly.

Type Definition

```
<xsd:complexType name="Downtime">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The measurement directive does not require additional parameters. For specific cases the Downtime measurement directive can be extended by required additional information.

Example

```
    <MeasurementDirective xsi:type="wsla:DownTime"
                          resultType="double">
    </MeasurementDirective>
```

In the given example, the result type of the measurement directive is double. `resultType` is a common attribute of a measurement directive.

3.5. Functions

3.5.1. Time Series Constructor

In many cases, metrics and SLAParameters are defined as function of the behavior of a simple metric over time, for example, the average response time in a given period. The basic element for the application of average, maximum and similar functions is an underlying time series of measured values. This function describes the construction of a time series.

Type Definition

```
<xsd:complexType name="TSConstructor">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:element name="Schedule" type="xsd:string"/>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Window" type="xsd:long" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Schedule` element is a reference to a schedule defined in the service description. It defines the time span when the time series is constructed and the intervals at which new values are inserted.

- The `Metric` is a reference to the basic metric over which the time series is constructed.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.
- The `Window` defines how many values of the time series are to be stored. The measurement component assumes that only the last n values actually need to be kept available for further computing. If the `Window` element is not used it is assumed that all measured values are stored.

Example

```
<Function xsi:type="wsla:TSConstructor" resultType="TS">
    <Schedule>hourlyschedule</Schedule>
    <Metric>SumResponseTime</Metric>
    <Window>2</Window>
</Function>
```

In this example the time series is constructed over the metric `"SumResponseTime"` according to the `"hourlyschedule"`, which refers to a schedule definition in the service definition part of the SLA. Only the last two values are stored.

3.5.2 Time Series Select

This function selects a particular index from the time series.

Type Definition

```
<xsd:complexType name="TSSelect">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:element name="Operand" type="wsla:OperandType"/>
        <xsd:element name="Element" type="xsd:long"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Operand` element contains either a reference to a metric or the definition of another function. Type is `OperandType`.
- `Element` is the selector of the time series element. 0 represents the most recent element. Negative numbers refer to elements further in the past, e.g. -1 is the one before last value.

Example

```
<Function xsi:type="wsla:TSSelect" result="long">
  <Operand>
    <Metric>SumResponseTimeTimeSeries</Metric>
  </Operand>
  <Element>-1</Element>
</Function>
```

In this example the next-to-last element of the time series `"SumResponseTimeTimeSeries"` is selected.

3.5.2. Queue Constructor

Queues are time-ordered collections of measurements. Typically, queue entries are not measured triggered by a schedule but are events that are pushed from the instrumentation to the measurement components.

Type Definition

```
<xsd:complexType name="QConstructor">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Window" type="xsd:long" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric over which the queue is constructed.
- Alternatively, a `Function` can be defined that results in a value that is to be represented in the queue.
- The `Window` defines how many values will be stored in the queue. This assumes that only the last n values actually need to be kept available for further computing. If the `Window` element is not used it is assumed the number of stored values is left to the implementation.

Example

```
<Metric name="SumTransactionSeries" type="Q" unit="transactions">
    <Source>ACMEProvider</Source>
    <Function xsi:type="QConstructor" resultType="Q">
        <Metric>SumTransactions</Metric>
        <Window>2</Window>
    </Function>
</Metric>
```

In this example queue is constructed over the metric `"SumTransactions"`. Only the last two values are stored.

3.5.3. Size

The `Size` function computes the number of elements of a time series or queue.

Type Definition

```
<xsd:complexType name="Size">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:element name="Operand" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Operand` must yield a time series or a queue.

Example

```
<Function xsi:type="Size" resultType="float">
  <Metric>StatusTimeSeries</Metric>
</Function>
```

In this example, the size of the metric `StatusTimeSeries` is computed.

3.5.4. Mean♠

The `Mean` function computes the arithmetic mean of the numeric values of a time series or queue. It is the sum of all values divided by the number of values. This is what is commonly called the "average."

Type Definition

```
<xsd:complexType name="Mean">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.

Example

```
  <Function xsi:type="wsla:Mean" resultType="double">
    <Metric>responseTimesHost1</Metric>
  </Function>
```

In this example the metric `responseTimesHost1` is the argument of the function and must be a time series.

NOTE: In previous versions of this specification, this function used to be called “Average” and can still be used.

3.5.5. Median ♠

The `Median` function computes the arithmetic median of the numeric values of a time series or queue. It is the middle value of a set of numbers when the values are arranged from lowest to highest. (NB: When a distribution is symmetric, mean = median.)

Type Definition

```
<xsd:complexType name="Median">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.

- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.

Example

```
  <Function xsi:type="wsla:Median" resultType="double">
    <Metric>responseTimesHost1</Metric>
  </Function>
```

In this example the metric `responseTimesHost1` is the argument of the function and must be a time series.

3.5.6. Mode ♠

The `Mode` function computes the value that occurs most often (i.e., with the highest frequency) within a set of measurements.

Type Definition

```
<xsd:complexType name="Mode">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.

- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.

Example

```
  <Function xsi:type="wsla:Mode" resultType="double">
    <Metric>responseTimesHost1</Metric>
  </Function>
```

In this example the metric `responseTimesHost1` is the argument of the function and must be a time series.

3.5.7. Round ♠

This function rounds a floating point number (or a number of type double) to the precision indicated by the `Digits` parameter.

Type Definition

```
<xsd:complexType name="Round">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Digits" type="wsla:OperandType"/>
      </xsd:sequence>
```

```
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type float or double.
- `Digits` indicate the precision of the round function: A value of "2" indicates that the number should be rounded to the second digit after the decimal point.

Example

```
  <Function xsi:type="wsla:Round"
            resultType="float">
    <Metric>responseTimeHost1</Metric>
    <Digits>
      2
    </Digits>
  </Function>
```

In this example the metric `responseTimesHost1` is the argument of the function and is supposed to be rounded to 2 digits after the decimal point.

NOTE: In analogy to `Round`, a function `Truncate` (which cuts off a float after a specified digit) may be specified as well.

3.5.8. Sum

The `Sum` function adds the numeric values of a time series or queue.

Type Definition

```
<xsd:complexType name="Sum">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.

(♠ In the current implementation of the WSLA Compliance Monitor, Sum applies to 2 metrics, like Plus.)

Example

```
  <Function xsi:type="wsla:Sum" resultType="double">
    <Metric>Failures</Metric>
  </Function>
```

In this example the sum of the time series metrics `Failures` is computed.

♠ Currently the same as "Plus".

3.5.9. Max

The `Max` function returns the highest value of a time series or queue of numeric metrics.

Type Definition

```
<xsd:complexType name="Max">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.

(♠ In the current implementation of the WSLA Compliance Monitor, Max applies to time series only.)

Example

```
  <Function xsi:type="wsla:Max" resultType="double">
    <Metric>responseTimesHost1</Metric>
  </Function>
```

In this example the maximum of the metrics `responseTimesHost1` is computed.

3.5.10. ValueOccurs

This function counts the number of occurrences of a particular value in a time series or queue. This is useful for computing overall occurrences of events such as service down times.

Type Definition

```
<xsd:complexType name="ValueOccurs">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:element name="Value" type="wsla:OperandType"/>
        <xsd:element name="Metric" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Function" type="wsla:FunctionType"
                     minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Value` that is the counted item.
- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.

Example

```
          <Function xsi:type="ValueOccurs" resultType="long">
            <Metric>StatusTimeSeries</Metric>
            <Value>
              <LongScalar>0</LongScalar>
            </Value>
          </Function>
```

In the example we assume a time series of boolean metrics `StatusTimeSeries` that represents whether a service was up (1) or down (0). This function computes the occurrences of the down state.

3.5.11. Arithmetic Functions

The usual arithmetic functions are available in their common meaning. The operands can be either metrics or again functions.

Type Definition

```
<xsd:complexType name="Plus">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
          <xsd:element name="Operand" type="wsla:OperandType" minOccurs="2"
                                                              maxOccurs="2"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Minus">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
          <xsd:element name="Operand" type="wsla:OperandType" minOccurs="2"
                                                              maxOccurs="2"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Divide">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
          <xsd:element name="Operand" type="wsla:OperandType"
                      minOccurs="2"
                      maxOccurs="2"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Times">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
          <xsd:element name="Operand" type="wsla:OperandType"
                      minOccurs="2"
                      maxOccurs="2"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

(♠ In the current implementation of the WSLA Compliance Monitor, Plus is called Sum.)

3.5.12. PercentageGreaterThanThreshold ♠

This function returns the percentage of elements of a time series or queue whose value is greater than a given value.

Type Definition

```
<xsd:complexType name="PercentageGreaterThanThreshold">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.
- `Value` is the threshold.

Example

```
  <Function xsi:type="wsla:PercentageGreaterThanThreshold" resultType="float">
    <Metric>responseTimeHost1</Metric>
    <Value>
      <LongScalar>2000</LongScalar>
    </Value>
  </Function>
```

3.5.13. PercentageLessThanThreshold ♠

This function returns the percentage of elements of a time series or queue whose value is less than a given value.

Type Definition

```
<xsd:complexType name="PercentageLessThanThreshold">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.
- Alternatively, a `Function` can be defined that results in a value that is to be represented in the time series.

- `Value` is the threshold.

Example

```
  <Function xsi:type="wsla:PercentageLessThanThreshold" resultType="float">
    <Metric>responseTimeHost1</Metric>
    <Value>
      <LongScalar>2000</LongScalar>
    </Value>
  </Function>
```

3.5.14. NumberGreaterThanThreshold ♠

This function returns the number of elements of a time series or queue whose value is greater than a given value.

Type Definition

```
<xsd:complexType name="NumberGreaterThanThreshold">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.
- `Value` is the threshold.

Example

```
  <Function xsi:type="wsla:NumberGreaterThanThreshold"
            resultType="integer">
    <Metric>responseTimeHost1</Metric>
    <Value>
      <LongScalar>2000</LongScalar>
    </Value>
  </Function>
```

3.5.15. NumberLessThanThreshold ♠

This function returns the number of elements of a time series or queue whose value is less than a given value.

Type Definition

```
<xsd:complexType name="NumberLessThanThreshold">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
```

```
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue with numeric items.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.
- `Value` is the threshold.

Example

```
  <Function xsi:type="wsla:NumberLessThanThreshold"
            resultType="integer">
    <Metric>responseTimeHost1</Metric>
    <Value>
      <LongScalar>2000</LongScalar>
    </Value>
  </Function>
```

3.5.16. RateOfChange ♠

This function returns the rate of change of the values of a time series. The time series values must be numeric. The result is again a time series.

Type Definition

```
<xsd:complexType name="RateOfChange">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series with numeric items.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.

Example

```
  <Function xsi:type="wsla:RateOfChange" resultType="integer">
    <Metric>responseTimeTimeSeriesHost1</Metric>
    <Value>
      <LongScalar>2000</LongScalar>
    </Value>
  </Function>
```

3.5.17. Span

This function returns the number of sequential occurrences of a particular value in a time series or queue, backwards from the most recent entry. The result is of type long.

Type Definition

```
<xsd:complexType name="Span">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- The `Metric` is a reference to the basic metric on which the function is applied. The metric must be of type time series or queue.
- Alternatively, a `Function` can be defined that results in a value, which is to be represented in the time series.
- Value is the specific value whose sequential occurrence is counted.

Example

```
<Metric name="CurrentDownTime" type="long" unit="minutes">
    <Source>YMeasurement</Source>
    <Function xsi:type="Span" resultType="double">
        <Metric>StatusTimeSeries</Metric>
        <Value>
          <Scalar>0</Scalar>
        </Value>
    </Function>
</Metric>
```

This example illustrates the use of the `Span` function in a metric definition. It is applied on a time series defined in the metric `"StatusTimeSeries"` and counts how many times the scalar `"0"` (indicating the service is down) occurs in a row counting from the most recent entry. In this example, the value is `"1"` if the service is up, thus the result of the `Span` function is 0. If it is down for the most recent 3 entries, the `Span` function returns 3.

3.6. Predicates

3.6.1. Violation

The `Violation` predicate verifies whether a service level guarantee has been violated. This predicate is particularly important to define action guarantees that oblige to action upon occurrence of a service level guarantee violation.

This predicate is different from the following as it refers to another guarantee and is a kind of meta-predicate, as opposed to other predicates that refer primarily to SLA parameters.

Type Definition

```
<xsd:complexType name="Violation">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="ServiceLevelObjective" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
```

```
  </xsd:complexContent>
</xsd:complexType>
```

- `ServiceLevelObjective` is a reference to a service level guarantee.

Example

```
    <ActionGuarantee name="g2">
      <Obliged>provider</Obliged>
      <Expression>
        <Predicate xsi:type="wsla:Violation">
          <ServiceLevelObjective>g1</ServiceLevelObjective>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <Action>
        <Party>customer</Party>
        <Action>Notification</Action>
      </Action>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>
```

In the example the provider must send a notification to the customer if the service level guarantee `"g1"` is violated.

3.6.2. Greater

This derivation of the predicate type compares the value of an SLA Parameter with a given value.

Type Definition

```
<xsd:complexType name="Greater">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:IDREF"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `SLAParameter` is the parameter to which the predicate is applied.
- `Value` is a `double` to which the parameter's value is compared.

Example

```
   <Predicate xsi:type="wsla:Greater">
     <SLAParameter>Throughput</SLAParameter>
     <Value>27</Value>
   </Predicate>
```

The example requires the SLA Parameter `"Throughput"` to be `Greater` than 27.

3.6.3. Less

This derivation of the predicate type compares the value of an SLA Parameter with a given value.

Type Definition

```
<xsd:complexType name="Less">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
```

```
        <xsd:element name="SLAParameter" type="xsd:IDREF"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `SLAParameter` is the parameter to which the predicate is applied.
- `Value` is a `double` to which the parameter's value is compared.

3.6.4. Equal

This derivation of the predicate type compares the value of an SLAParameter with a given value.

Type Definition

```
<xsd:complexType name="Equal">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:IDREF"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `SLAParameter` is the parameter to which the predicate is applied.
- `Value` is a `double` to which the parameter's value is compared.

3.6.5. GreaterEqual

This derivation of the predicate type compares the value of an SLA Parameter with a given value.

Type Definition

```
<xsd:complexType name="GreaterEqual">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:IDREF"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `SLAParameter` is the parameter to which the predicate is applied.
- `Value` is a `double` to which the parameter's value is compared.

3.6.6. LessEqual

This derivation of the predicate type compares the value of an SLA Parameter with a given value.

Type Definition

```
<xsd:complexType name="LessEqual">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:IDREF"/>
```

```
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `SLAParameter` is the parameter to which the predicate is applied.
- `Value` is a `double` to which the parameter's value is compared.

3.6.7. True ♠

This predicate is always true.

Type Definition

```
<xsd:complexType name="True">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

3.6.8. False ♠

This predicate is always false.

Type Definition

```
<xsd:complexType name="False">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

3.7. Action Invocations

3.7.1. Notification

The `Notification` type of `ActionInvocation` defines the marshaling of the Notification operation. The signature of the Notification is defined in Appendix 4 as an interface in WSDL. Three parameters are to be set:

Type Definition

```
<xsd:complexType name="Notification">
  <xsd:complexContent>
    <xsd:extension base="wsla:ActionInvocationType">
      <xsd:sequence>
        <xsd:element name="NotificationType"
                     type="wsla:NotificationType"/>
        <xsd:element name="CausingGuarantee" type="xsd:string"/>
        <xsd:element name="SLAParameter" type="wsla:StringList"/>
             <!-- Convention: Can be either:
                   - "NONE"
                   - <enumerated list of SLAParameter IDs>
              -->
      </xsd:sequence>
```

```
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- `NotificationType` is either `"Violation"` for notifications sent on grounds of a violation of a service level guarantee, or `"Information"` for all other purposes.

- `CausingGuarantee` is the name of the obligation, either ServiceLevelObligation or ActionGuarantee on which grounds this notification is sent. The reference to the obligation enables the recipient to interpret the notification. The recipient must be aware of the different guarantees of an SLA

- `SLAParameter` is a list of names of SLA Parameters that are sent as name-value pairs. As a special information item, the XML specification of a ServiceLevelObjective expression can be included. It is referred to as “Expression:<SLOName>”.

Example

```
<Action actionName="notification" xsi:type="Notification">
   <NotificationType>Information</NotificationType>
   <CausingGuarantee>g3</CausingGuarantee>
   <SLAParameter>ResponseTimeThroughPutRatio TransactionRate
                 Expression:g1</SLAParameter>
</Action>
```

The example shows the use of the notification action. In this case, notification type is set to "Information", the causing guarantee is “`g3`” and the values of the SLA Parameters "`ResponseTimeThroughPutRatio`" and `"TransactionRate"` are to be included. Also, the expression of the SLO “`g1`” is passed along.

3.8. Additional Data Types of the Standard Extensions

3.8.1. NotificationType

The `NotificationType` distinguishes notifications on violation from other information.

Type Definition

```
<xsd:simpleType name="NotificationType">
  <xsd:restriction base="xsd:string">
     <xsd:enumeration value="Violation"/>
     <xsd:enumeration value="Information"/>
  </xsd:restriction>
</xsd:simpleType>
```

The type is a simple restriction of the string type. Further types can be added in an extension if finer granularity is needed.

Acknowledgments

We are grateful for the input and support of the following contributors (in alphabetical order): Daniel Dias, Joe Hellerstein, Gautam Kar, Robert Kearney, Martin Sachs.

References

[1] W3C: XML Schema Definition Language. http://www.w3.org/XML/Schema
[2] IBM, Microsoft, Ariba (E. Christensen, F. Curbera, G. Meredith, S. Weerawarana): Web Services Description Language. http://www.w3.org/TR/wsdl
[3] IETF, IETF RFC 3060: Policy Framework. http://www.ietf.org
[4] ebXML: http://www.ebxml.org
[5] W3C: Web Service Description Language (WSDL). W3C Note, 15 March 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.
[6] A. Dan, D. M. Dias, R. Kearney, T. C. Lau, T. N. Nguyen, F. N. Parr, M. W. Sachs, and H. Shaikh, "Business-to-business integration with tpaML and a business-to-business protocol framework", IBM Systems Journal, Volume 40, Number 1, 2001, pp. 68-90.
[7] Oasis: http://www.oasis-open.org
[8] W. Stallings: SNMPv3 and RMON 1 and 2. Third Edition. Addison Wesley, 1999.
[9] W. Bumpus, J. Sweitzer, P. Thompson, A. Westerinen, R. Williams: Common Information Model - Implementing the Object Model for Enterprise Management. Wiley, 2000.
[10] IETF Network Working Group (B. Moore, E. Ellesson, J. Strassner, A. Westerinen. RFC 3060 - Policy Core Information Model. February 2001.
[11] Keller, A., Ludwig, H., The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, *Journal of Network and Systems Management, Special Issue on "E-Business Management"*, Volume 11, Number 1, Plenum Publishing Corporation, March, 2003. Also available as IBM Research Report RC 22456.

Appendix 1 - WSLA Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:wsla="http://www.ibm.com/wsla"
            targetNamespace="http://www.ibm.com/wsla"
            elementFormDefault="qualified">


<xsd:annotation>
  <xsd:documentation>
   Schema for the WSLA Language.

      Version: wsla-20030127-1
      Revision: 1.00
   Authors:  Heiko Ludwig (Contact), hludwig@us.ibm.com
                 Alexander Keller
                 Asit Dan
                 Richard P. King
                 Richard Franck

   Copyright 2001, 2002, 2003 IBM Corp. all rights reserved.

   Revision Comment:
                This revision 1.0 is not backward compatible to early versions.
  </xsd:documentation>
</xsd:annotation>

<!--                        -->
<!-- Global WSLA structure -->
<!--                        -->

<xsd:complexType name="WSLAType">
  <xsd:sequence>
    <xsd:element ref="wsla:Parties"/>
    <xsd:element name="ServiceDefinition" type="wsla:ServiceDefinitionType"
                                          maxOccurs="unbounded"/>
    <xsd:element name="Obligations" type="wsla:ObligationsType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="SLA" type="wsla:WSLAType">

</xsd:element>


<!--                    -->
<!-- Party Definitions -->
<!--                    -->

<xsd:complexType name="ContactInformationType">
  <xsd:sequence>
    <xsd:element name="Street" type="xsd:string"/>
    <xsd:element name="City" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="RoleType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="MeasurementService"/>
```

```
    <xsd:enumeration value="ManagementService"/>
    <xsd:enumeration value="ConditionEvaluationService"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="PartyType" abstract="true">
  <xsd:sequence>
    <xsd:element name="Contact" type="wsla:ContactInformationType"
                 minOccurs="0" />
    <xsd:element name="Action" type="wsla:ActionDescriptionType"
                 minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="SignatoryPartyType">
  <xsd:complexContent>
    <xsd:extension base="wsla:PartyType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SupportingPartyType">
  <xsd:complexContent>
    <xsd:extension base="wsla:PartyType">
      <xsd:sequence>
        <xsd:element name="Sponsor" type="xsd:string" maxOccurs="2"/>
        <xsd:element name="Role" type="wsla:RoleType" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="PartiesType">
  <xsd:sequence>
    <xsd:element name="ServiceProvider" type="wsla:SignatoryPartyType"/>
    <xsd:element name="ServiceConsumer" type="wsla:SignatoryPartyType"/>
    <xsd:element name="SupportingParty" type="wsla:SupportingPartyType"
                                        minOccurs="0"
                     maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="Parties" type="wsla:PartiesType"/>

<!-- Actions -->


<xsd:complexType name="ActionDescriptionType" abstract="true">
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="partyName" type="xsd:string"/>
</xsd:complexType>


<!--                       -->
<!-- Service Definitions -->
<!--                       -->

<xsd:complexType name="ServiceDefinitionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ServiceObjectType">
      <xsd:sequence>
        <xsd:element ref="wsla:Operation" minOccurs="0" maxOccurs="unbounded"/>
```

```
        <xsd:element ref="wsla:OperationGroup" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element ref="wsla:WebHosting" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- Service Objects -->

<xsd:complexType name="ServiceObjectType" abstract="true">
  <xsd:sequence>
    <xsd:element ref="wsla:Schedule" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="wsla:Trigger" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="wsla:Constant" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="wsla:MetricMacroDefinition" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element ref="wsla:MetricMacroExpansion" minOccurs="0"
maxOccurs="unbounded"/>
    <xsd:element ref="wsla:SLAParameter" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="wsla:Metric" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="Schedule" type="wsla:ScheduleType"/>

<xsd:complexType name="ScheduleType">
  <xsd:sequence>
    <xsd:element name="Period" type="wsla:PeriodType"/>
    <xsd:element name="Interval" type="wsla:IntervalType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>


<xsd:element name="OperationGroup" type="wsla:OperationGroupType"/>

<xsd:complexType name="OperationGroupType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ServiceObjectType">
      <xsd:sequence>
        <xsd:element ref="wsla:Operation" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Operation" type="wsla:OperationDescriptionType"/>

<xsd:complexType name="OperationDescriptionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ServiceObjectType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="WebHosting" type="wsla:WebHostingType"/>

<xsd:complexType name="WebHostingType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ServiceObjectType">
      <xsd:sequence>
```

```
        <xsd:element name="Coverage" type="xsd:anyURI" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- Parameters and Metrics -->

<xsd:element name="SLAParameter" type="wsla:SLAParameterType"/>

<xsd:complexType name="SLAParameterType">
  <xsd:sequence>
      <xsd:element name="Metric" type="xsd:string"/>
      <xsd:element name="Communication"
          type="wsla:SLAParameterCommunicationType"
          minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="type" type="wsla:Type"/>
  <xsd:attribute name="unit" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="SLAParameterCommunicationType">
  <xsd:sequence>
      <xsd:element name="Source" type="xsd:string"/>
      <xsd:element name="Pull" type="wsla:StringList" minOccurs="0"/>
      <xsd:element name="Push" type="wsla:StringList" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>


<xsd:complexType name="MetricType">
  <xsd:sequence>
      <xsd:element name="Source" type="xsd:string"/>
      <xsd:element name="MetricURI" type="xsd:anyURI" minOccurs="0"/>
      <xsd:choice>
        <xsd:element name="MeasurementDirective"
                     type="wsla:MeasurementDirectiveType"/>
        <xsd:element name="Function"
                     type="wsla:FunctionType"/>
        <xsd:element name="MeasurementDirectiveVariable"
                     type="wsla:MDVariableType"/>
      </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="type" type="wsla:Type"/>
  <xsd:attribute name="unit" type="xsd:string"/>
  <xsd:attribute name="counter" type="xsd:boolean" use="optional"/>
</xsd:complexType>

<xsd:element name="Metric" type="wsla:MetricType"/>

<!-- Measurement directives -->

<xsd:complexType name="MeasurementDirectiveType" abstract="true">
  <xsd:sequence>
      <xsd:element name="ReadingSchedule" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="resultType" type="wsla:Type"/>
</xsd:complexType>

<xsd:complexType name="MDVariableType">
</xsd:complexType>
```

```
<!-- Functions to compute derived metrics -->

<xsd:complexType name="FunctionType" abstract="true">
  <xsd:attribute name="resultType" type="wsla:Type"/>
</xsd:complexType>


<!-- Metric Macros -->

<xsd:complexType name="MetricMacroDefinitionType">
  <xsd:sequence>
      <xsd:element name="Metric"
          type="wsla:MetricType"
          maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="MetricMacroDefinition"
type="wsla:MetricMacroDefinitionType"/>

<xsd:complexType name="MDAssignmentType">
  <xsd:sequence>
      <xsd:element name="MeasurementDirective"
          type="wsla:MeasurementDirectiveType"/>
  </xsd:sequence>
  <xsd:attribute name="metricName" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="MetricMacroExpansionType">
  <xsd:sequence>
      <xsd:element name="MeasurementDirectiveAssignment"
          type="wsla:MDAssignmentType"
          minOccurs="0"
          maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="macroName" type="xsd:string"/>
  <xsd:attribute name="expansionName" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="MetricMacroExpansion" type="wsla:MetricMacroExpansionType"/>


<!-- Support Types -->

<xsd:element name="Constant" type="wsla:ConstantType"/>

<xsd:complexType name="ConstantType">
      <xsd:choice>
        <xsd:element name="String" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Integer" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="Float" type="xsd:float" minOccurs="0"/>
      </xsd:choice>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="Trigger" type="wsla:TriggerType"/>

<xsd:complexType name="TriggerType">
      <xsd:choice>
        <xsd:element name="Time" type="xsd:dateTime"/>
```

```
      </xsd:choice>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:simpleType name="Type">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Q"/>
    <xsd:enumeration value="TS"/>
    <xsd:enumeration value="integer"/>
    <xsd:enumeration value="float"/>
    <xsd:enumeration value="double"/>
    <xsd:enumeration value="long"/>
    <xsd:enumeration value="byte"/>
    <xsd:enumeration value="boolean"/>
    <xsd:enumeration value="string"/>
    <xsd:enumeration value="time"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="StringList">
  <xsd:list itemType="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="IntegerList">
  <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="FloatList">
  <xsd:list itemType="xsd:float"/>
</xsd:simpleType>

<xsd:complexType name="IntervalType">
      <xsd:sequence>
        <xsd:element name="Years" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="Months" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="Days" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="Hours" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="Minutes" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="Seconds" type="xsd:integer" minOccurs="0"/>
        <xsd:element name="MilliSeconds" type="xsd:integer" minOccurs="0"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PeriodType">
  <xsd:choice>
  <xsd:sequence>
    <xsd:element name="Start" type="xsd:dateTime"/>
    <xsd:element name="End" type="xsd:dateTime"/>
  </xsd:sequence>
  <xsd:sequence>
    <xsd:element name="ConditionTime" type="xsd:string"/>
    <xsd:element name="ConditionMonthOfYearMask" type="xsd:string"/>
    <xsd:element name="ConditionDayOfMonthMask" type="xsd:string"/>
    <xsd:element name="ConditionDayOfWeekMask" type="xsd:string"/>
    <xsd:element name="ConditionTimeOfDayMask" type="xsd:string"/>
    <xsd:element name="ConditionTimeZone" type="xsd:string"/>
    <xsd:element name="ConditionLocalOrUtcTime" type="xsd:string"/>
  </xsd:sequence>
  </xsd:choice>
</xsd:complexType>

<xsd:simpleType name="EvaluationEventType">
  <xsd:restriction base="xsd:string">
```

```
    <xsd:enumeration value="NewValue"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ExecutionModalityType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Always"/>
    <xsd:enumeration value="OnEnteringCondition"/>
    <xsd:enumeration value="OnEnteringAndOnLeavingCondition"/>
    <xsd:enumeration value="OnEveryEvaluation"/>
  </xsd:restriction>
</xsd:simpleType>


<!--            -->
<!-- Guarantees -->
<!--            -->


<xsd:complexType name="ObligationObjectType">
  <xsd:sequence>
    <xsd:element ref="wsla:Constant"
                 minOccurs="0"
                 maxOccurs="unbounded"/>
    <xsd:element name="ServiceLevelObjective"
                 type="wsla:ServiceLevelObjectiveType"
                 minOccurs="0"
                 maxOccurs="unbounded"/>
    <xsd:element name="ActionGuarantee"
                 type="wsla:ActionGuaranteeType"
                 minOccurs="0"
                 maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="ObligationsType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ObligationObjectType">
      <xsd:sequence>
        <xsd:element ref="wsla:ObligationGroup" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="ObligationGroup" type="wsla:ObligationGroupType"/>

<xsd:complexType name="ObligationGroupType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ObligationObjectType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>


<xsd:complexType name="ActionGuaranteeType">
  <xsd:sequence>
    <xsd:element name="Obliged" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="Expression" type="wsla:LogicExpressionType"/>
    <xsd:choice>
      <xsd:element name="EvaluationEvent" type="wsla:EvaluationEventType"/>
```

```
      <xsd:element name="Schedule" type="xsd:string"/>
    </xsd:choice>
    <xsd:element ref="wsla:QualifiedAction" maxOccurs="unbounded"/>
    <xsd:element name="ExecutionModality" type="wsla:ExecutionModalityType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>


<xsd:complexType name="ServiceLevelObjectiveType">
  <xsd:sequence>
    <xsd:element name="Obliged" type="xsd:string" />
    <xsd:element name="Validity" type="wsla:PeriodType" maxOccurs="unbounded"/>
    <xsd:element name="Expression" type="wsla:LogicExpressionType"/>
    <xsd:choice>
      <xsd:element name="EvaluationEvent" type="wsla:EvaluationEventType"/>
      <xsd:element name="Schedule" type="xsd:string"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="QualifiedAction" type="wsla:QualifiedActionType">
</xsd:element>

<xsd:complexType name="QualifiedActionType">
  <xsd:sequence>
    <xsd:element name="Party" type="xsd:string"/>
    <xsd:element name="Action" type="wsla:ActionInvocationType"/>

  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PredicateType" abstract="true"/>

<xsd:complexType name="LogicExpressionType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:element name="Predicate" type="wsla:PredicateType"/>
      <xsd:element name="And" type="wsla:BinaryLogicOperatorType"/>
      <xsd:element name="Or" type="wsla:BinaryLogicOperatorType"/>
      <xsd:element name="Not" type="wsla:UnaryLogicOperatorType"/>
      <xsd:element name="Implies" type="wsla:BinaryLogicOperatorType"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="BinaryLogicOperatorType">
  <xsd:sequence>
      <xsd:element name="Expression" type="wsla:LogicExpressionType"
                   minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="UnaryLogicOperatorType">
  <xsd:sequence>
      <xsd:element name="Expresssion" type="wsla:LogicExpressionType"/>
  </xsd:sequence>
</xsd:complexType>



<!--                      -->
```

```
<!-- Standard Extensions -->
<!--                     -->

<!-- Standard Action Types -->

<xsd:complexType name="WSDLSOAPActionDescriptionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ActionDescriptionType">
      <xsd:sequence>
        <xsd:element name="WSDLFile" type="xsd:anyURI"/>
        <xsd:element name="ServiceName" type="xsd:string" minOccurs="0"/>
        <xsd:element name="SOAPBindingName" type="xsd:string"/>
        <xsd:element name="SOAPOperationName" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="WSDLGetPostActionDescriptionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:ActionDescriptionType">
      <xsd:sequence>
        <xsd:element name="ServiceName" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Address" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- Standard Service Object Description Types -->

<xsd:complexType name="WSDLSOAPOperationDescriptionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:OperationDescriptionType">
      <xsd:sequence>
        <xsd:element name="WSDLFile" type="xsd:anyURI"/>
        <xsd:element name="ServiceName" type="xsd:string" minOccurs="0"/>
        <xsd:element name="SOAPBindingName" type="xsd:string"/>
        <xsd:element name="SOAPOperationName" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="WSDLSOAPOperation" substitutionGroup="wsla:Operation"
        type="wsla:WSDLSOAPOperationDescriptionType"/>

<xsd:complexType name="StringOperationDescriptionType">
  <xsd:complexContent>
    <xsd:extension base="wsla:OperationDescriptionType">
      <xsd:sequence>
        <xsd:element name="Description" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>


<!-- Standard Measurement Directive Types -->

<xsd:complexType name="Counter">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
```

```
      <xsd:sequence>
        <xsd:element name="MeasurementURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Gauge">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="MeasurementURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ResponseTime">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="MeasurementURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SumResponseTime">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Status">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="MeasurementURI" type="xsd:anyURI"/>
        <xsd:element name="TimeOut" type="wsla:IntervalType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="InvocationCount">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="CounterURI" type="xsd:anyURI" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="StatusRequest">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
        <xsd:element name="RequestURI" type="xsd:anyURI"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
```

```
</xsd:complexType>

<xsd:complexType name="Downtime">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="MaintenancePeriodQuery">
  <xsd:complexContent>
    <xsd:extension base="wsla:MeasurementDirectiveType">
      <xsd:sequence>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>


<!-- Standard Functions -->

<xsd:complexType name="QConstructor">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Window" type="xsd:long" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="TSConstructor">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:element name="Schedule" type="xsd:string"/>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Window" type="xsd:long" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Size">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:element name="Operand" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="TSSelect">
```

```
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:element name="Operand" type="wsla:OperandType"/>
        <xsd:element name="Element" type="xsd:long"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Plus">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
          <xsd:element name="Operand" type="wsla:OperandType" minOccurs="2"
                                                              maxOccurs="2"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Minus">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
          <xsd:element name="Operand" type="wsla:OperandType" minOccurs="2"
                                                              maxOccurs="2"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Divide">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
          <xsd:element name="Operand" type="wsla:OperandType" minOccurs="2"
                                                              maxOccurs="2"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Times">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
          <xsd:element name="Operand" type="wsla:OperandType" minOccurs="2"
                                                              maxOccurs="2"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- For backward compatibility - to be replace by Mean -->
<xsd:complexType name="Average">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
```

```
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Mean">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Median">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Mode">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Round">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Digits" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Max">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
```

```
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Sum">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ValueOccurs">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="PercentageGreaterThanThreshold">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="PercentageLessThanThreshold">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:complexType name="NumberGreaterThanThreshold">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="NumberLessThanThreshold">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="RateOfChange">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Interval" type="wsla:IntervalType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Span">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
        </xsd:choice>
        <xsd:element name="Value" type="wsla:OperandType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="QualifyMeasurementTimeSeries">
  <xsd:complexContent>
    <xsd:extension base="wsla:FunctionType">
      <xsd:sequence>
     <xsd:element name="BaseMetric" type="xsd:string"/>
          <xsd:element name="ValidationMetric" type="xsd:string"/>
        <xsd:element name="Value" type="wsla:OperandType"/>
```

```
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>



<xsd:complexType name="OperandType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Metric" type="xsd:string"/>
          <xsd:element name="Function" type="wsla:FunctionType"/>
          <xsd:element name="LongScalar" type="xsd:long"/>
          <xsd:element name="FloatScalar" type="xsd:float"/>
          <xsd:element name="StringScalar" type="xsd:string"/>
          <xsd:element name="Constant" type="xsd:string"/>
        </xsd:choice>
      </xsd:sequence>
</xsd:complexType>


<!-- Standard Predicate Types -->

<xsd:complexType name="NewValue">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:choice>
        <xsd:element name="SLAParameter" type="xsd:string"/>
        <xsd:element name="SLAParameterList" type="wsla:StringList"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Violation">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="ServiceLevelObjective" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Greater">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:string"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Less">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:string"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
```

```
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Equal">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:string"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="GreaterEqual">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:string"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>


<xsd:complexType name="LessEqual">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence>
        <xsd:element name="SLAParameter" type="xsd:string"/>
        <xsd:element name="Value" type="xsd:double"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="True">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="False">
  <xsd:complexContent>
    <xsd:extension base="wsla:PredicateType">
      <xsd:sequence/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>


<!-- Action Invocation Types -->

<xsd:complexType name="ActionInvocationType">
  <xsd:attribute name="actionName" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="Notification">
  <xsd:complexContent>
    <xsd:extension base="wsla:ActionInvocationType">
```

```
      <xsd:sequence>
        <xsd:element name="NotificationType" type="wsla:NotificationType"/>
        <xsd:element name="CausingGuarantee" type="xsd:string"/>
        <xsd:element name="SLAParameter" type="wsla:StringList"/>
          <!-- Convention: Can be either:
               - "NONE"
               - <enumerated list of SLAParameter IDs>
              It is understood that only the relevant
              subset of parameters are actually sent.
          -->
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>


<xsd:simpleType name="NotificationType">
  <xsd:restriction base="xsd:string">
     <xsd:enumeration value="Violation"/>
     <xsd:enumeration value="Information"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

Appendix 2 - Example 1

This is a simple example illustrating the use of metrics, functions and measurement directives aswell as guarantees.

The Scenario: This WSLA contract refers to a Stock Quote Service as defined in the file stockquote.wsdl. It is assumed that a service provider provides a simple function to obtain a stock quote.

Two hosts are assigned by the provider to serve requests, host1 and host2. Both hosts expose an SLAParameter "ResponseTime" that is measured at a given sampling interval to gain metrics such as average response time etc. The measurement is performed by a third party measurement service ms, which works on behalf of the service provider.

The service provider guarantees to the customer an average (mean) response time over all requestes and host systems of 5 seconds. In the case of the average response time reaching 4 seconds, an indicator for upcoming critical condition, provider and customer want to be notified by ms. Equally, if the average response time is again out of the critical zone, both parties expect notification. Finally, notification is agreed upon in the case of violation of the service level.

```
<?xml version="1.0"?>
<!--
 Simple SLA for the Stock Quote Service.
 Author: Heiko Ludwig, hludwig@us.ibm.com
 Date: October 24, 2001
 -->
<SLA    xmlns="http://www.ibm.com/wsla"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.ibm.com/wsla c:\Projects\WSLA\wsla.xsd"
 name="StockquoteServiceAgreement12345" >

<Parties>
 <ServiceProvider
  name="provider">
  <Contact>
   <POBox>P.O.Box 218</POBox>
   <City>Yorktown, NY 10598, USA</City>
  </Contact>
   <Action xsi:type="WSDLSOAPActionDescriptionType"
         name="Notification"
            partyName="provider">
       <WSDLFile>Notification.wsdl</WSDLFile>
       <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
       <SOAPOperationName>Notify</SOAPOperationName>
    </Action>
 </ServiceProvider>
 <ServiceCustomer
  name="customer">
  <Contact>
   <Street>30 Saw Mill River RD</Street>
   <City>Hawthorne, NY 10532, USA</City>
  </Contact>
   <Action xsi:type="WSDLSOAPActionDescriptionType"
         name="Notification"
            partyName="customer">
       <WSDLFile>Notification.wsdl</WSDLFile>
       <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
       <SOAPOperationName>Notify</SOAPOperationName>
    </Action>
 </ServiceCustomer>
```

```
 <SupportingParty
  name="ms"
  role="MeasurementServiceProvider"
  sponsor="provider">
  <Contact>
   <Street>Saeumerstrasse 4</Street>
   <City>CH-8803 Ruschlikon, Switzerland</City>
  </Contact>
 </SupportingParty>
</Parties>
<ServiceDefinition
 name="StockQuoteservice">
 <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType"
  name="WSDLSOAPGetQuote">
   <Schedule name="MainSchedule">
     <Period>
       <Start>2001-11-30T14:00:00.000-05:00</Start>
       <End>2001-12-31T14:00:00.000-05:00</End>
    </Period>
     <Interval>
       <Minutes>2</Minutes>
       <Seconds>30</Seconds>
     </Interval>
   </Schedule>
   <SLAParameter name="AverageResponseTime"
                        type="float"
                        unit="seconds">
     <Metric>averageResponseTime</Metric>
   </SLAParameter>
   <Metric name="averageResponseTime" type="double" unit="seconds">
       <Source>ms</Source>
       <Function xsi:type="wsla:Divide" resultType="double">
                <Operand>
                  <Function xsi:type="wsla:Plus" resultType="double">
                    <Operand>
                      <Metric>averageResponseTimeHost1</Metric>
                    </Operand>
                    <Operand>
                      <Metric>averageResponseTimeHost2</Metric>
                    </Operand>
           </Function>
                </Operand>
               <Operand>
                  <LongScalar>2</LongScalar>
                </Operand>
       </Function>
   </Metric>
      <Metric name="averageResponseTimeHost1" type="double" unit="seconds">
       <Source>ms</Source>
       <Function xsi:type="wsla:Mean" resultType="double">
    <Metric>responseTimesHost1</Metric>
       </Function>
   </Metric>
      <Metric name="responseTimesHost1" type="TS" unit="seconds">
       <Source>ms</Source>
       <Function xsi:type="wsla:TSConstructor" resultType="TS">
           <Schedule>MainSchedule</Schedule>
           <Metric>responseTimeHost1</Metric>
       </Function>
   </Metric>
      <Metric name="responseTimeHost1" type="double" unit="seconds">
       <Source>ms</Source>
       <MeasurementDirective xsi:type="wsla:ResponseTime"
```

```
                              resultType="double">
          <MeasurementURI>http://ms.com/testResponse</MeasurementURI>
       </MeasurementDirective>
   </Metric>
      <Metric name="averageResponseTimeHost2" type="double" unit="seconds">
       <Source>ms</Source>
       <Function xsi:type="wsla:Mean" resultType="double">
         <Metric>responseTimesHost2</Metric>
       </Function>
   </Metric>
      <Metric name="responseTimesHost2" type="TS" unit="seconds">
       <Source>ms</Source>
       <Function xsi:type="wsla:TSConstructor" resultType="TS">
                <Schedule>MainSchedule</Schedule>
         <Metric>ResponseTimeHost2</Metric>
       </Function>
   </Metric>
      <Metric name="responseTimeHost2" type="double" unit="seconds">
       <Source>ms</Source>
       <MeasurementDirective xsi:type="wsla:ResponseTime"
                              resultType="double">
         <MeasurementURI>http://ms.com/testResponse</MeasurementURI>
       </MeasurementDirective>
   </Metric>
      <WSDLFile>StockQuoteService.wsdl</WSDLFile>
      <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
      <SOAPOperationName>getQuote</SOAPOperationName>
  </Operation>
</ServiceDefinition>
<Obligations>
 <ServiceLevelObjective name="g1" serviceObject="WSDLSOAPGetQuote">
   <Obliged>provider</Obliged>
   <Validity>
    <StartDate>2001-08-15:1400</StartDate>
    <EndDate>2001-09-15:1400</EndDate>
   </Validity>
   <Expression>
     <Predicate xsi:type="wsla:Less">
       <SLAParameter>AverageResponseTime</SLAParameter>
       <Value>5</Value>
     </Predicate>
   </Expression>
   <EvaluationEvent>NewValue</EvaluationEvent>
 </ServiceLevelObjective>
 <ActionGuarantee name="g2">
   <Obliged>ms</Obliged>
   <Expression>
     <Predicate xsi:type="wsla:Violation">
        <ServiceLevelObjective>g1</ServiceLevelObjective>
     </Predicate>
   </Expression>
   <EvaluationEvent>NewValue</EvaluationEvent>
   <QualifiedAction>
      <Party>customer</Party>
      <Action actionName="notification" xsi:type="Notification">
        <NotificationType>Violation</NotificationType>
        <CausingGuarantee>g1</CausingGuarantee>
        <SLAParameter>AverageResponseTime</SLAParameter>
      </Action>
   </QualifiedAction>
   <ExecutionModality>Always</ExecutionModality>
 </ActionGuarantee>
```

```
 <ActionGuarantee name="g3">
          <Obliged>ms</Obliged>
   <Expression>
     <Predicate xsi:type="wsla:Less">
       <SLAParameter>AverageResponseTime</SLAParameter>
       <Value>4</Value>
     </Predicate>
   </Expression>
   <EvaluationEvent>NewValue</EvaluationEvent>
   <QualifiedAction>
      <Party>customer</Party>
      <Action actionName="notification" xsi:type="Notification">
        <NotificationType>Violation</NotificationType>
        <CausingGuarantee>g1</CausingGuarantee>
        <SLAParameter>AverageResponseTime</SLAParameter>
      </Action>
   </QualifiedAction>
   <QualifiedAction>
      <Party>provider</Party>
      <Action actionName="notification" xsi:type="Notification">
        <NotificationType>Violation</NotificationType>
        <CausingGuarantee>g1</CausingGuarantee>
        <SLAParameter>AverageResponseTime</SLAParameter>
      </Action>
   </QualifiedAction>
   <ExecutionModality>OnEnteringAndOnLeavingCondition</ExecutionModality>
 </ActionGuarantee>
</Obligations>
</SLA>
```

Appendix 3 - Example 2

This is the full SLA of the running example of this document, involving supporting parties and defining guarantees related to performance as well as availability. Although the SLA defines various SLA parameters, we will focus on the following parts and describe them in more detail:

For the operation `getQuote` of a Web Service, two SLA parameters `TransactionRate` and `OverLoadPercentage` (percentage of time the service provider's system experiences a workload that is above the agreed-upon threshold) are defined.
The rationale for choosing these two parameters is as follows: SLAs are defined under the assumption that the ranges of SLA parameters defined for a service reflect typical workloads. In practice, a service provider has authority over some environmental factors while others are beyond his control. Thus, an SLA needs to take into account under which conditions the obligations are valid. Assigning simply a threshold to an SLA parameter is not helpful without considering the variations of workload to which a service provider's system may be exposed, because sudden load surges may increase the workload on the system by several multiples. An increase of the workload by, e.g., a factor of 5 or more makes it impossible for a service provider to meet fixed response time or throughput targets. Thus, in our example, `OverLoadPercentage` will serve as a precondition to constrain under which circumstances the service provider needs to guarantee a given `TransactionRate`.

Since SLA parameters are surfaced by a Measurement Service to a Condition Evaluation Service, it is important to define which party is supposed to provide the value (`Source`) and which parties can receive it, either event-driven (`Push`) or through polling (`Pull`). In our example, we define the SLA parameter `OverloadPercentage` accordingly: It is assigned the metric `OverloadPercentageMetric`, which is defined independently of the SLA parameter for being used potentially multiple times. YMeasurement promises to send (`Push`) new values to ZAuditing, which is also allowed to retrieve new values on its own initiative (`Pull`).

`OverloadPercentageMetric` and `UtilizationTimeSeries` are two sample composite metrics having the datatypes `float` and `TS`, a WSLA type to represent time series. YMeasurement is in charge of computing the values of both metrics. `UtilizationTimeSeries` is of type TS and has no unit. The example illustrates the concept of a function: Every 5 minutes (cf. the schedule `5minuteschedule`), a new value of the metric `ProbedUtilization` is placed by the function of type `wsla:TSConstructor` into a time series for further processing. The second Metric `OverloadPercentageMetric` is used to determine the amount of time when a system is overloaded and expresses this as a percentage. In our example, we consider a system utilization of less than 80% as a safe operating region; above this value, the system is considered overloaded. Specific functions, such as *Minu*s, *Average* or, here, *PercentageGreaterThanThreshold* (yielding the percentage of values over a threshold in a time series, in our example 0.8 or 80%) are extensions of the common function type.

The example below further details a service level objective `ConditionalSLOForTransactionRate` given by ACMEProvider and valid for a full month in the year 2001. It guarantees that the SLA parameter `TransactionRate` must be greater than 1000 if the SLA parameter `OverloadPercentage` is less than 0.3, i.e., the service provider must make sure his system is able to handle at least 1000 transactions per second under the condition that his system is operating under normal load conditions for 70% of the time. If the service provider experiences an overload condition for 30% of the time (due, e.g., to an excessive amount of incoming requests), he is not obliged to fulfill the `TransactionRate` requirement. Note that in our example, overload is defined as a system utilization of at least 80% for a period of one hour (see the definition of the metric `PercentOverUtilized`). This condition should be evaluated each time a new value for the SLA parameter is available. The example shows how the `Implies` element can be used for defining preconditions in WSLA.

In the ActionGuarantee example `ConstrainedTARateGuarantee`, `ZAuditing` is obliged to invoke the notification action of the service customer `XInc` if a violation of the service level objective `ConditionalSLOForTransactionRate` occurs. The precondition should be evaluated every time the evaluation of the SLO returns a new value. Further details are given in [11].

```
<?xml version="1.0"?>

<!--
    Sample SLA involving third parties for measurement and
    condition evaluation.

    Authors: Heiko Ludwig, hludwig@us.ibm.com
             Richard P. King, rpk@us.ibm.com
             Alexander Keller, alexk@us.ibm.com
    Date: November 11, 2002

-->

<SLA xmlns="http://www.ibm.com/wsla"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.ibm.com/wsla WSLA.xsd"
     name="DemoSLA123" >

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

<Parties>
    <ServiceProvider
        name="ACMEProvider">
      <Contact>
        <Street>PO BOX 218</Street>
        <City>Yorktown, NY 10598, USA</City>
      </Contact>
   <Action xsi:type="WSDLSOAPOperationDescriptionType"
             name="notification"
           partyName="ZAuditing">
     <WSDLFile>Notification.wsdl</WSDLFile>
        <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
        <SOAPOperationName>Notify</SOAPOperationName>
      </Action>
    </ServiceProvider>

    <ServiceConsumer
      name="XInc">
      <Contact>
        <Street>19 Skyline Drive</Street>
        <City>Hawthorne, NY 10532, USA</City>
      </Contact>
   <Action xsi:type="WSDLSOAPOperationDescriptionType"
             name="notification"
           partyName="ZAuditing">
     <WSDLFile>Notification.wsdl</WSDLFile>
        <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
        <SOAPOperationName>Notify</SOAPOperationName>
      </Action>
    </ServiceConsumer>

    <SupportingParty
      name="YMeasurements"
      role="MeasurementService">
        <Contact>
            <Street>Saeumerstrasse 4</Street>
```

```
            <City>CH-8803 Rueschlikon, Switzerland</City>
          </Contact>
    <Sponsor>ACMEProvider</Sponsor>
     </SupportingParty>

     <SupportingParty
       name="ZAuditing"
    role="ConditionEvaluationService">
       <Contact>
         <Street>Hursley Park</Street>
         <City>Winchester, England, SO21 2JN</City>
       </Contact>
    <Action xsi:type="WSDLSOAPOperationDescriptionType"
              name="parameterUpdate"
            partyName="ZAuditing">
      <WSDLFile>ParameterUpdate.wsdl</WSDLFile>
         <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
         <SOAPOperationName>parameterUpdate</SOAPOperationName>
       </Action>
    <Sponsor>ACMEProvider</Sponsor>
    <Sponsor>XInc</Sponsor>
     </SupportingParty>

</Parties>


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

<ServiceDefinition
     name="DemoService">

       <Schedule name="businessdayschedule">
         <Period>
           <Start>2001-11-30T14:00:00.000-05:00</Start>
           <End>2001-12-31T14:00:00.000-05:00</End>
         </Period>
         <Interval>
           <Minutes>1440</Minutes>
         </Interval>
     </Schedule>

       <Schedule name="hourlyschedule">
         <Period>
           <Start>2001-11-30T14:00:00.000-05:00</Start>
           <End>2001-12-31T14:00:00.000-05:00</End>
         </Period>
         <Interval>
           <Minutes>60</Minutes>
         </Interval>
       </Schedule>

       <Schedule name="5minuteschedule">
         <Period>
           <Start>2001-11-30T14:00:00.000-05:00</Start>
           <End>2001-12-31T14:00:00.000-05:00</End>
         </Period>
         <Interval>
           <Minutes>5</Minutes>
         </Interval>
     </Schedule>

       <Schedule name="availabilityschedule">
```

```
      <Period>
        <Start>2001-11-30T14:00:00.000-05:00</Start>
        <End>2001-12-31T14:00:00.000-05:00</End>
      </Period>
      <Interval>
        <Minutes>1</Minutes>
      </Interval>
 </Schedule>

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

   <SLAParameter name="OverloadPercentage"
                 type="float"
                 unit="Percentage">
     <Metric>OverloadPercentageMetric</Metric>
<Communication>
  <Source>ACMEProvider</ Source >
  <Pull>ZAuditing</Pull>
  <Push>ZAuditing</Push>
</Communication>
   </SLAParameter>

   <SLAParameter name="TransactionRate"
                 type="float"
                 unit="transactions / hour">
    <Metric>Transactions</Metric>
    <Communication>
      <Source>ACMEProvider</Source>
      <Pull>ZAuditing</Pull>
      <Push>ZAuditing</Push>
    </Communication>
   </SLAParameter>

   <SLAParameter name="Availability_CurrentDownTime"
                 type="long"
                 unit="minutes">
     <Metric>CurrentDownTime</Metric>
       <Communication>
         <Source>YMeasurement</Source>
         <Push>ZAuditing</Push>
       </Communication>
   </SLAParameter>

   <SLAParameter name="Availability_UpTimeRatio"
                 type="float"
                 unit="">
     <Metric>UpTimeRatio</Metric>
     <Communication>
       <Source>YMeasurement</Source>
       <Push>ZAuditing</Push>
     </Communication>
   </SLAParameter>

   <Metric name="OverloadPercentageMetric" type="float" unit="Percentage">
     <Source>YMeasurement</Source>
     <Function xsi:type="PercentageGreaterThanThreshold" resultType="float">
        <Schedule>businessdayschedule</Schedule>
        <Metric>UtilizationTimeSeries</Metric>
           <Value>
              <LongScalar>0.8</LongScalar> <!-- 80% -->
           </Value>
     </Function>
   </Metric>
```

```
<Metric name="UtilizationTimeSeries" type="TS" unit="">
  <Source>YMeasurement</Source>
  <Function xsi:type="TSConstructor" resultType="float">
      <Schedule>5minuteschedule</Schedule>
      <Metric>ProbedUtilization</Metric>
      <Window>12</Window>
   </Function>
</Metric>

<Metric name="ProbedUtilization" type="float" unit="">
  <Source>YMeasurement</Source>
  <MeasurementDirective xsi:type="Gauge" resultType="float">
      <RequestURL>http://acme.com/SystemUtil</RequestURL>
  </MeasurementDirective>
</Metric>

<Metric name="Transactions" type="long" unit="transactions">
    <Source>ACMEProvider</Source>
    <Function xsi:type="Minus" resultType="double">
      <Operand>
          <Function xsi:type="TSSelect" resultType="long">
            <Operand>
              <Metric>SumTransactionTimeSeries</Metric>
            </Operand>
            <Element>0</Element>
          </Function>
      </Operand>
      <Operand>
          <Function xsi:type="TSSelect" resultType="long">
            <Operand>
              <Metric>SumTransactionTimeSeries</Metric>
            </Operand>
            <Element>-1</Element>
          </Function>
      </Operand>
    </Function>
</Metric>

<Metric name="SumTransactionTimeSeries" type="TS" unit="transactions">
    <Source>ACMEProvider</Source>
    <Function xsi:type="TSConstructor" resultType="TS">
        <Schedule>hourlyschedule</Schedule>
        <Metric>SumTransactions</Metric>
        <Window>2</Window>
    </Function>
</Metric>

 <Metric name="ResponseTime" type="long" unit="milliseconds">
    <Source>ACMEProvider</Source>
    <Function xsi:type="Minus" resultType="double">
        <Operand>
          <Function xsi:type="TSSelect" resultType="long">
            <Operand>
              <Metric>SumResponseTimeTimeSeries</Metric>
            </Operand>
            <Element>0</Element>
          </Function>
        </Operand>
        <Operand>
          <Function xsi:type="TSSelect" resultType="long">
            <Operand>
              <Metric>SumResponseTimeTimeSeries</Metric>
```

```
                </Operand>
                <Element>-1</Element>
              </Function>
            </Operand>
        </Function>
    </Metric>

     <Metric name="SumResponseTimeTimeSeries" type="TS" unit="milliseconds">
        <Source>ACMEProvider</Source>
        <Function xsi:type="TSConstructor" resultType="TS">
            <Schedule>hourlyschedule</Schedule>
            <Metric>SumResponseTime</Metric>
            <Window>2</Window>
        </Function>
    </Metric>

    <Metric name="SumTransactions" type="long" unit="tansactions">
        <Source>ACMEProvider</Source>
        <MeasurementDirective xsi:type="InvocationCount" resultType="long"/>
    </Metric>

     <Metric name="SumResponseTime" type="long" unit="milliseconds">
        <Source>ACMEProvider</Source>
        <MeasurementDirective xsi:type="SumResponseTime" resultType="long"/>
    </Metric>


    <Metric name="CurrentDownTime" type="long" unit="minutes">
        <Source>YMeasurement</Source>
        <Function xsi:type="Span" resultType="double">
            <Metric>StatusTimeSeries</Metric>
            <Value>
              <LongScalar>0</LongScalar>
            </Value>
        </Function>
    </Metric>

    <Metric name="UpTimeRatio" type="long" unit="">
        <Source>YMeasurement</Source>
        <Function xsi:type="Minus" resultType="double">
            <Operand>
              <LongScalar>1</LongScalar>
            </Operand>
            <Operand>
              <Function xsi:type="Divide" resultType="long">
                <Operand>
                  <Function xsi:type="ValueOccurs" resultType="long">
                    <Metric>StatusTimeSeries</Metric>
                    <Value>
                      <LongScalar>0</LongScalar>
                    </Value>
                  </Function>
                </Operand>
                <Operand>
                  <LongScalar>1440</LongScalar>
                </Operand>
              </Function>
            </Operand>
        </Function>
    </Metric>

    <Metric name="StatusTimeSeries" type="TS" unit="">
        <Source>YMeasurement</Source>
```

```
          <Function xsi:type="TSConstructor" resultType="TS">
              <Schedule>availabilityschedule</Schedule>
              <Metric>MeasuredStatus</Metric>
              <Window>1440</Window>
          </Function>
      </Metric>

      <Metric name="MeasuredStatus" type="integer" unit="">
          <Source>YMeasurement</Source>
          <MeasurementDirective xsi:type="StatusRequest" resultType="integer">

<RequestURI>http://ymeasurement.com/StatusRequest/GetQuote</RequestURI>
          </MeasurementDirective>
      </Metric>

      <WSDLFile>DemoService.wsdl</WSDLFile>
      <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
      <SOAPOperationName>getQuote</SOAPOperationName>
  </Operation>

</ServiceDefinition>


<!-- The obligations of the parties, referring to parameters defined above.
-->


  <Obligations>

    <ServiceLevelObjective name="ConditionalSLOForTransactionRate">
      <Obliged>ACMEProvider</Obliged>
      <Validity>
         <Start>2001-11-30T14:00:00.000-05:00</Start>
         <End>2001-12-31T14:00:00.000-05:00</End>
      </Validity>
      <Expression>
        <Implies>
          <Expression>
            <Predicate xsi:type="Less">
              <SLAParameter>OverloadPercentage</SLAParameter>
              <Value>0.3</Value>      <!-- 30% -->
            </Predicate>
          </Expression>
          <Expression>
            <Predicate xsi:type="Greater">
              <SLAParameter>TransactionRate</SLAParameter>
              <Value>1000</Value>
            </Predicate>
           </Expression>
         </Implies>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
    </ServiceLevelObjective>

    <ServiceLevelObjective name="UpTimeSLO">
      <Obliged>ACMEProvider</Obliged>
      <Validity>
          <Start>2001-11-30T14:00:00.000-05:00</Start>
          <End>2001-12-31T14:00:00.000-05:00</End>
      </Validity>
      <Expression>
        <Predicate xsi:type="Greater">
          <SLAParameter>Availability_UpTimeRatio</SLAParameter>
```

```
          <Value>0.97</Value>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
    </ServiceLevelObjective>

    <ServiceLevelObjective name="ContinuousDowntimeSLO">
      <Obliged>ACMEProvider</Obliged>
      <Validity>
          <Start>2001-11-30T14:00:00.000-05:00</Start>
          <End>2001-12-31T14:00:00.000-05:00</End>
      </Validity>
      <Expression>
        <Predicate xsi:type="Less">
          <SLAParameter>Availability_CurrentDownTime</SLAParameter>
          <Value>10</Value>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
    </ServiceLevelObjective>

    <ActionGuarantee name="UpTimeNotificationGuarantee">
      <Obliged>ZAuditing</Obliged>
      <Expression>
        <Predicate xsi:type="Violation">
          <ServiceLevelObjective>UpTimeSLO</ServiceLevelObjective>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>XInc</Party>
        <Action actionName="notification" xsi:type="Notification">
     <NotificationType>Violation</NotificationType>
     <CausingGuarantee>UpTimeNotificationGuarantee 3</CausingGuarantee>
     <SLAParameter>Availability_UpTimeRatio</SLAParameter>
   </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>

    <ActionGuarantee name="ContinuousDowntimeNotificationGuarantee">
      <Obliged>ZAuditing</Obliged>
      <Expression>
        <Predicate xsi:type="Violation">
          <ServiceLevelObjective>ContinuousDowntimeSLO</ServiceLevelObjective>
        </Predicate>      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>XInc</Party>
        <Action actionName="notification" xsi:type="Notification">
     <NotificationType>Violation</NotificationType>
     <CausingGuarantee>ContinuousDowntimeNotificationGuarantee
4</CausingGuarantee>
      <SLAParameter>Availability_CurrentDownTime</SLAParameter>
   </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>

    <ActionGuarantee name="ConstrainedTARateGuarantee">
      <Obliged>ZAuditing</Obliged>
      <Expression>
        <Predicate xsi:type="Violation">
```

```
<ServiceLevelObjective>ConditionalSLOForTransactionRate</ServiceLevelObjective>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>XInc</Party>
        <Action actionName="notification" xsi:type="Notification">
     <NotificationType>Violation</NotificationType>
     <CausingGuarantee> ConstrainedTARateGuarantee</CausingGuarantee>
     <SLAParameter>OverloadPercentage TransactionRate</SLAParameter>
   </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>


  </Obligations>

</SLA>
```

Appendix 4 - WSLA Service Deployment Information (WSLA SDI) Language

The WSLA Service Deployment Information (WSLA SDI) language is derived from the WSLA language to represent information that is sent from a service provider or customer to third party services that are sponsored by the respective party. Signatory parties must supply their sponsored parties with the information that is necessary for them to fulfill the role in a particular SLA, but not more information than required. (See 1.4.5 - deployment process.) Making the entire SLA available to a third party would release information beyond what is required for a third party service to fulfil its role. For this purpose we need a means to express suitable substructures of the WSLA that allow to represent the information necessary for a particular type of service.

The WSLA SDI language represents a "repackaging" of the elements of the WSLA language to suit these purposes. The WSLA SDI language includes the complete WSLA language and adds a set of new top-level types and elements that group information content of an SLA for the intended sponsored service.

The examples used in this Appendix are derived from the SLA example in Appendix 3.

Appendix 4.1 Measurement Service SDI

A measurement service is commissioned to read resource metrics according to measurement directives, aggregate them into higher-level metrics and finally send them to a condition evaluation service (either implemented by another third party or by a signatory party) in the form of SLAParameters. For this purpose it primarily needs to know the exact WSDL definitions, interface and implementation, of the `ParameterUpdate` operations of the condition evaluation services that it is corresponding with, the SLA Parameters that it should send out, the metrics that it needs to compute for this purpose or that it can obtain from another measurement service and the schedule definitions required for the metric computations.

Type Definition

```
<xsd:complexType name="SDIMeasurementServiceType">
  <xsd:sequence>
    <xsd:element name="Party" type="wsla:PartyType">
    <xsd:element name="Action" type="wsla:ActionDescriptionType"
                                        maxOccurs="unbounded"/>
    <xsd:element name="Schedule" type="wsla:ScheduleType"
                                        minOccurs="0"
                                        maxOccurs="unbounded"/>
    <xsd:element name="ServiceObject" type="wsla:ServiceObjectType"
                                        maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="SDIMeasurementService"
             type="wsla:SDIMeasurementServiceType"/>
```

The following elements are contained in the SDI definition, all being based on types of the WSLA language and XML schema:

- There can be any number of `Action` definitions of `parameterUpdate` operations, usually corresponding to the number of condition evaluation services that need to be updated.
- The `Schedules` needed to compute the relevant `Metrics`.
- The `SLAParameters` whose value updates should be sent to a condition evaluation service.
- The `Metric` definitions that are necessary to compute the `SLAParameters`.

Example

```
<?xml version="1.0"?>
<!--
    Service Deployment Information for a Measurement Service
    Authors: Heiko Ludwig, hludwig@us.ibm.com
             Richard P. King, rpk@us.ibm.com
    Date: December 7, 2001
-->
<SDIMeasurementService
     xmlns="http://www.ibm.com/wsla"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.ibm.com/wsla WSLASDI.xsd"
     name="DemoSLA123" >
<!-- Definition of the Involved Parties, the signatory parties as well as
    the supporting ones -->
<Party name="YMeasurement" xsi:type="SupportingParty"
        role=”MeasurementService”>
    <Action name="notification" partyName="YMeasurement"
                  xsi:type="wsla:WSDLGetPostActionDescriptionType">
      <Address>http://localhost:8080/wsla/
                      servlet/Condlet?com.ibm.wsla.cm.WSNotifier</Address>
    </Action>
<Party>
<Action name="parameterUpdate"
        partyName="ZAuditing"
        xsi:type="WSDLSOAPActionDescriptionType">
           <WSDLFile>parameterUpdate.wsdl</WSDLFile>
           <SOAPBindingName>soapnotification</SOAPBindingName>
           <SOAPOperationName>parameterUpdate</SOAPOperationName>
</Action>
<Schedule name="availabilityschedule">
  <Period>
    <Start>2001-11-30T14:00:00.000-05:00</Start>
    <End>2001-12-31T14:00:00.000-05:00</End>
  </Period>
  <Interval>
    <Minutes>1</Minutes>
  </Interval>
</Schedule>

<SLAParameter name="Availability_CurrentDownTime"
                  type="long"
                  unit="minutes">
        <Metric>CurrentDownTime</Metric>
 <Communication>
   <Service>YMeasurement</Service>
   <Push>ZAuditing</Push>
 </Communication>
</SLAParameter>
<SLAParameter name="Availability_UpTimeRatio"
                  type="float"
                  unit="">
        <Metric>UpTimeRatio</Metric>
 <Communication>
   <Service>YMeasurement</Service>
   <Push>ZAuditing</Push>
 </Communication>
</SLAParameter>
<Metric name="CurrentDownTime" type="long" unit="minutes">
    <Source>YMeasurement</Source>
    <Function xsi:type="Span" resultType="double">
        <Metric>StatusTimeSeries</Metric>
```

```
        <Value>
          <LongScalar>0</LongScalar>
        </Value>
    </Function>
</Metric>
<Metric name="UpTimeRatio" type="long" unit="">
    <Source>YMeasurement</Source>
    <Function xsi:type="Minus" resultType="double">
        <Operand>
          <LongScalar>1</LongScalar>
        </Operand>
        <Operand>
          <Function xsi:type="Divide" resultType="long">
            <Operand>
              <Function xsi:type="ValueOccurs" resultType="long">
                <Metric>StatusTimeSeries</Metric>
                <Value>
                  <LongScalar>0</LongScalar>
                </Value>
              </Function>
            </Operand>
            <Operand>
              <LongScalar>1440</LongScalar>
            </Operand>
          </Function>
        </Operand>
    </Function>
</Metric>
<Metric name="StatusTimeSeries" type="TS" unit="">
    <Source>YMeasurement</Source>
    <Function xsi:type="TSConstructor" resultType="TS">
        <Schedule>availabilityschedule</Schedule>
        <Metric>MeasuredStatus</Metric>
        <Window>1440</Window>
    </Function>
</Metric>
<Metric name="MeasuredStatus" type="integer" unit="">
    <Source>YMeasurement</Source>
    <MeasurementDirective xsi:type="StatusRequest" resultType="integer">

<RequestURI>http://ymeasurement.com/StatusRequest/GetQuote</RequestURI>
    </MeasurementDirective>
</Metric>
</SDIMeasurementService>
```

Appendix 4.2 Condition Evaluation Service SDI

A condition evaluation service supervises service level guarantees defined in the WSLA and triggers actions at management services or signatory parties when particular conditons hold (e.g. a service level guarantee is violated or a critical condition, but not yet a violation, is reached). A condition evaluation service receives SLA Parameters by one or more measurment services or the equivalent function of a signatory party and triggers actions at any number of management services and signatory parties.

To fulfill its role, the condition evaluation service needs the definition of actions that it needs to trigger at various parties, the SLA Parameters that it is supposed to receive and the obligations of the contract that it is charged of dealing with. This comprises the service level guarantees to be supervised as well as action guarantees that need to be implemented.

Type Definition

```
<xsd:complexType name="SDIConditionEvaluationServiceType">
```

```
  <xsd:sequence>
    <xsd:element name="Party" type="wsla:PartyType">
    <xsd:element name="Action" type="wsla:ActionDescriptionType"
                                        minOccurs="0"
                                        maxOccurs="unbounded"/>
    <xsd:element name="ServiceObject" type="wsla:ServiceObjectType"
                                        minOccurs="0"
                                        maxOccurs="unbounded"/>
    <xsd:element name="Obligations" type="wsla:ObligationsType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="SDIConditionEvaluationService"
             type="wsla:SDIConditionEvaluationServiceType"/>
```

This SDI contains the following elements, all of which based on the respective types in the WSLA language and schema:

- There can be any number of `Action` definitions for all actions of all parties that need to be triggered by the condition evaluation service.
- There can be any numbers of `SLAParameters` whose values are expected to be received (or actively obtained) from a measurement service or a signatory party's equivalent.
- The set of `Obligations` that need to be supervised and implemented by the condition evaluation service comprises service level guarantees as well as action guarantees.

Example

```
<?xml version="1.0"?>
<!--
    Service Deployment Information for a Condition Evaluation Service
    Authors: Heiko Ludwig, hludwig@us.ibm.com
             Richard P. King, rpk@us.ibm.com
    Date: December 6, 2001
-->
<SDIConditionEvaluationService
     xmlns="http://www.ibm.com/wsla"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.ibm.com/wsla WSLASDI.xsd"
     name="DemoSLA123" >
<!-- Definition of the Involved Parties, the signatory parties as well as
    the supporting ones -->
 <Party name="ZAuditing" xsi:type="SupportingParty"
        role=”ConditionEvaluationService”>
    <Action name="notification" partyName="YMeasurement"
                  xsi:type="wsla:WSDLGetPostActionDescriptionType">
      <Address>http://localhost:8080/wsla/
                     servlet/Condlet?com.ibm.wsla.cm.WSNotifier</Address>
    </Action>
 <Party>
 <Action name="notification"
                partyName="ZAuditing"
                xsi:type="WSDLSOAPActionDescriptionType">
           <WSDLFile>notification.wsdl</WSDLFile>
           <SOAPBindingName>soapnotification</SOAPBindingName>
           <SOAPOperationName>notification</SOAPOperationName>
        </Action>
 <Action name="notification"
                partyName="ZAuditing"
                xsi:type="WSDLSOAPActionDescriptionType">
           <WSDLFile>notification.wsdl</WSDLFile>
```

```
           <SOAPBindingName>soapnotification</SOAPBindingName>
           <SOAPOperationName>notification</SOAPOperationName>
        </Action>
<SLAParameter name="ResponseTimeThroughPutRatio"
                        type="float"
                        unit="milisconds / transactions">
        <Metric>ResponseTimeThroughPutRatioMetric</Metric>
</SLAParameter>
<SLAParameter name="TransactionRate"
                        type="float"
                        unit="transactions / hour">
        <Metric>Transactions</Metric>
</SLAParameter>
<SLAParameter name="Availability_CurrentDownTime"
                        type="long"
                        unit="minutes">
        <Metric>CurrentDownTime</Metric>
        <Communication>
          <Service>YMeasurement</Service>
          <Push>ZAuditing</Push>
        </Communication>
</SLAParameter>
<SLAParameter name="Availability_UpTimeRatio"
                        type="float"
                        unit="">
        <Metric>UpTimeRatio</Metric>
        <Communication>
          <Service>YMeasurement</Service>
          <Push>ZAuditing</Push>
        </Communication>
</SLAParameter>

<Obligations>
    <ServiceLevelObjective name="g1">
      <Obliged>ACMEProvider</Obliged>
      <Validity>
          <Start>2001-11-30T14:00:00.000-05:00</Start>
          <End>2001-12-31T14:00:00.000-05:00</End>
      </Validity>
      <Expression>
        <Or>
          <Expression>
            <Predicate xsi:type="Less">
              <SLAParameter>ResponseTimeThroughPutRatio</SLAParameter>
              <Value>0.5</Value>
            </Predicate>
          </Expression>
          <Expression>
            <Predicate xsi:type="Greater">
              <SLAParameter>TransactionRate</SLAParameter>
              <Value>10000</Value>
            </Predicate>
          </Expression>
        </Or>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
    </ServiceLevelObjective>
    <ServiceLevelObjective name="ga1">
      <Obliged>ACMEProvider</Obliged>
      <Validity>
          <Start>2001-11-30T14:00:00.000-05:00</Start>
          <End>2001-12-31T14:00:00.000-05:00</End>
      </Validity>
```

```
      <Expression>
        <Predicate xsi:type="Greater">
          <SLAParameter>Availability_UpTimeRatio</SLAParameter>
          <Value>0.97</Value>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
    </ServiceLevelObjective>
    <ServiceLevelObjective name="ga2">
      <Obliged>ACMEProvider</Obliged>
      <Validity>
          <Start>2001-11-30T14:00:00.000-05:00</Start>
          <End>2001-12-31T14:00:00.000-05:00</End>
      </Validity>
      <Expression>
        <Predicate xsi:type="Less">
          <SLAParameter>Availability_CurrentDownTime</SLAParameter>
          <Value>10</Value>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
    </ServiceLevelObjective>
    <ActionGuarantee name="ga3">
      <Obliged>ZAuditing</Obliged>
      <Expression>
        <Predicate xsi:type="Violation">
          <ServiceLevelObjective>ga1</ServiceLevelObjective>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>XInc</Party>
        <Action actionName="notification" xsi:type="Notification">
          <NotificationType>Violation</NotificationType>
          <CausingGuarantee>ga1</CausingGuarantee>
          <SLAParameter>Availability_UpTimeRatio</SLAParameter>
        </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>
    <ActionGuarantee name="ga4">
      <Obliged>ZAuditing</Obliged>
      <Expression>
        <Predicate xsi:type="Violation">
          <ServiceLevelObjective>ga2</ServiceLevelObjective>
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>XInc</Party>
        <Action actionName="notification" xsi:type="Notification">
          <NotificationType>Violation</NotificationType>
          <CausingGuarantee>ga2</CausingGuarantee>
          <SLAParameter>Availability_CurrentDownTime</SLAParameter>
        </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>
    <ActionGuarantee name="g2">
      <Obliged>YAuditing</Obliged>
      <Expression>
        <Predicate xsi:type="Violation">
          <ServiceLevelObjective>g1</ServiceLevelObjective>
```

```
        </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>XInc</Party>
        <Action actionName="notification" xsi:type="Notification">
          <NotificationType>Violation</NotificationType>
          <CausingGuarantee>g2</CausingGuarantee>
          <SLAParameter>ResponseTimeThroughPutRatio
TransactionRate</SLAParameter>
        </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>
    <ActionGuarantee name="g3">
      <Obliged>YAuditing</Obliged>
      <Expression>
          <Predicate xsi:type="Greater">
            <SLAParameter>ResponseTimeThroughPutRatio</SLAParameter>
            <Value>0.4</Value>
          </Predicate>
      </Expression>
      <EvaluationEvent>NewValue</EvaluationEvent>
      <QualifiedAction>
        <Party>XInc</Party>
        <Action actionName="notification" xsi:type="Notification">
          <NotificationType>Information</NotificationType>
          <CausingGuarantee>g3</CausingGuarantee>
          <SLAParameter>ResponseTimeThroughPutRatio</SLAParameter>
        </Action>
      </QualifiedAction>
      <ExecutionModality>Always</ExecutionModality>
    </ActionGuarantee>
</Obligations>
</SDIConditionEvaluationService>
```

Appendix 4.4 WSLA SDI Schema

The schema of the WSLA SDI language uses the same wsla namespace as the WSLA schema and includes the WSLA schema entirely.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:wsla="http://www.ibm.com/wsla"
            targetNamespace="http://www.ibm.com/wsla"
            elementFormDefault="qualified">
<!-- include main WSLA schema -->
<xsd:include schemaLocation="WSLA.xsd"/>
<!--                                        -->
<!-- Service Deployment Information for a -->
<!-- Condition Evaluation Service         -->
<!--                                        -->
<xsd:complexType name="SDIConditionEvaluationServiceType">
  <xsd:sequence>
    <xsd:element name="Action" type="wsla:ActionDescriptionType"
                                        minOccurs="0"
                                        maxOccurs="unbounded"/>
    <xsd:element name="SLAParameter" type="wsla:SLAParameterType"
                                        minOccurs="0"
                                        maxOccurs="unbounded"/>
    <xsd:element name="Obligations" type="wsla:ObligationsType"/>
  </xsd:sequence>
```

```
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="SDIConditionEvaluationService"
             type="wsla:SDIConditionEvaluationServiceType"/>

<!--                                          -->
<!-- Service Deployment Information for a -->
<!-- Measurement Service                      -->
<!--                                          -->
<xsd:complexType name="SDIMeasurementServiceType">
  <xsd:sequence>
    <xsd:element name="Action" type="wsla:ActionDescriptionType"
                                          maxOccurs="unbounded"/>
    <xsd:element name="Schedule" type="wsla:ScheduleType"
                                          minOccurs="0"
                                          maxOccurs="unbounded"/>
    <xsd:element name="SLAParameter" type="wsla:SLAParameterType"
                                          maxOccurs="unbounded"/>
    <xsd:element name="Metric" type="wsla:MetricType"
                                          maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="SDIMeasurementService"
             type="wsla:SDIMeasurementServiceType"/>
</xsd:schema>
```