Towards Automated SLA Management for Web Services

Akhil Sahai, Anna Durante, Vijay Machiraju
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-310 (R.1)
July 11th , 2002*

E-mail: {asahai, annad, vijaym}@hpl.hp.com

web services, service level agreements, SLA management, automation

In order to automate SLA management it is essential to specify SLAs in precise and unambiguous manner as well as keep the specification flexible. While precision will help automate the process of monitoring and metric collection, flexibility will enable extending it to unforeseen service level agreement specifications.

Towards Automated SLA Management for Web Services

Akhil Sahai, Anna Durante, Vijay Machiraju
HP Laboratories, 1501 Page Mill Road, Palo-Alto, CA 94034
{asahai, annad,vijaym}@hpl.hp.com

Keywords: web services, service level agreements, SLA management, and automation.

Abstract: In order to automate SLA management it is essential to specify SLAs in precise and unambiguous manner as well as keep the specification flexible. While precision will help automate the process of monitoring and metric collection, flexibility will enable extending it to unforeseen service level agreement specifications.

1 Introduction

Recent spurt in growth of e-commerce on Internet has led to conceptualization of web services. Increasingly businesses on the web are being turned into services that are accessible over the web. A web service is a service available via the Internet that completes tasks, solves problems, or conducts transactions. A web service drives new revenue streams or creates new efficiencies in the Internet economy. These services can be distinguished by the following characteristics:

- They are accessible on the web at a particular Uniform Resource Locator.
- They are composable in nature. A web service may depend on other web services. This composition may be static (at creation time) or dynamic (at run time).
- They are federated in nature. These web services would interact across enterprise boundaries.
- Their implementations are often different. They could be J2EE [11], .Net based [12].
- They agree upon document exchange protocols and web service language standards to communicate and interoperate with each other, like UDDI, XML, WSDL[14], SOAP.

If cross-enterprise dynamic composition of web services were to become a reality, a number of fundamental issues need to be addressed beyond agreement on document exchange formats. Some of these issues are (a) How will web services agree on what will be executed by each of the participants? (b) How will web services agree upon how well (performance, quality, etc) each of them will execute? (c) Who will be responsible for the overall execution or completion? (d) Who will be responsible if there is a failure in the overall execution or completion? (e) How will web services trust each other? **Service level agreements (SLAs)** are the corner stone for addressing these issues in web services. A service level agreement is an agreement between two web services regarding the guarantees that one of the web services (provider) offers to the other (consumer). The guarantees are about what transactions need to be executed and how well they should be executed. SLAs help in dividing up the responsibilities and risks among web services, thereby bringing more order to web services.

In a real-world scenario each web service interacts with many other web services, switching between the roles of being a provider in some interactions and consumer in others. Each of these interactions could potentially be governed by an SLA. Considering the number of interactions that a web service typically executes for completing one transaction, and the number of transactions that are executed within a day, the job of manually keeping track of which SLAs are violated, why they are violated, and how many times they are violated becomes an overwhelming task. Further, considering the legal and monetary implications in violating SLAs, it is in the best interest of a web service to predict and correct violations before they occur. On the other hand, if there is too much leeway in the specification of SLAs, a web service may not be able to fully capitalize on its capabilities.

Web services SLA management refers to automatic monitoring, enforcement, and optimization of SLAs between web services. One of the enablers for automated SLA management is a **flexible** but **precise** formalization of what an SLA is. The flexibility is needed since we neither completely understand nor can anticipate all possible SLAs for all the different types of web service providers. This will also help create a generic SLA management system for managing a range of different SLAs. The precision is essential so that an SLA management system can unambiguously interpret, monitor, enforce, and optimize SLAs.

In this paper, we discuss an SLA formalization that is flexible and precise. We expect this formalization will help us in creating an automated SLA management system (which is currently being worked on). The rest of this paper is organized as follows: Section 2 describes the state of the art in SLA specifications. In section 3, we explain the causes for ambiguity in SLAs with examples. This motivates our SLA formalization as explained in section 4. Section 5 describes three common SLAs that are useful for web services – performance, availability, and cost. We summarize our conclusions in section 6.

2 Background and Related Work

An SLA is typically signed between two parties, which have the role of provider and consumer respectively. A typical SLA [16] has the following components:

Purpose – describing the reasons behind the creation of the SLA

Parties – describes the parties involved in the SLA and their respective roles.

Validity Period- defines the period of time that the SLA will cover. This is delimited by start time and end time of the term.

Scope - defines the services covered in the agreement.

Restrictions - defines the necessary steps to be taken in order for the requested service levels to be provided.

Service-level objectives - are the levels of service that both the users and the service providers agree on, and usually include a set of *service level indicators*, like availability, performance and reliability. Each aspect of the service level, such as availability, will have a target level to achieve. Service Level objectives have day-time constraints associated with them, which delineate their validity.

Service-level indicators - the means by which these levels can be measured. Service Level Indicators (SLI) are the base level indicators. SLIs (that are relevant for web services) are mentioned in appendix A.

Penalties- spells out what happens in case the service provider under-performs and is unable to meet the objectives in the SLA. If the agreement is with an external service provider, the option of terminating the contract in light of unacceptable service levels should be built in.

Optional services - provides for any services that are not normally required by the user, but might be required as an exception.

Exclusions - specifies what is not covered in the SLA.

Administration - describes the processes created in the SLA to meet and measure its objectives and defines organizational responsibility for overseeing each of those processes

SLA specification has been researched earlier. Quality management Language (QML) [2] has been proposed for specifying Quality of Service for applications. In QML contracts are instances of contract types. A contract type defines the structure of its instances. In general a contract contains a list of constraints. A constraint consists of a name, an operator and a value. The name refers to a dimension and can also refer to a dimension aspect. A dimension aspect can be a percentile, mean, variance, and frequency. Usually SLAs have validity periods (start and end dates) and have SLOs that in turn have daytime constraints (e.g. an SLO may be valid only from Mon-Wed, between 6:00 PM-8:00 PM). It is quite essential to capture this kind of information in the contracts. This kind of information is however not captured in QML. In addition there should be a possibility of specifying a wide range of mathematical operations on the measured data. An example would be to guarantee *response time of 5 longest running transactions in last 24 hours < 25ms.* As it does not fall in the allowed dimension aspects it would not be possible to capture this in QML.

Bhoj et al. [3] describe a contract to be defined by a triple (P,M,A), where P is a set of properties, A is the set of assertions and M is the set of methods available on the contract. An *assertion* is an atomic group of statements agreed upon between the parties agreeing to the contract. Statements in an assertion are made up of logical predicates whose values can be uniquely determined. The logical predicates are composed using variables as well as logical operators, quantifiers, set operations and constraints on these variables. An example assertion may be *response time < 25 ms.* We believe however, that assertions have to be unambiguously specified, and the web services being autonomous should not need human intervention to understand the meaning of SLAs, and to monitor and measure their compliance. Also the web services will sign numerous SLAs with multiple parties over time and the SLA management process should be automated as much as possible. An assertion as mentioned above is quite ambiguous. This could mean an instance response time or average response time. Again if it is average response time that is being referred to, is it averaged over every 5 minutes, an hour or 24 hours? It is also necessary to indicate when the averages are calculated. Is it at 6PM everyday? Or at any time? The assertions thus have to be quite unambiguous. Also the assertions have to be captured in such a way that their meaning is clear to web services software that handle them, even if they are implicitly understood by humans. It is also necessary to identify components in assertions that are common across SLAs so that base modules can be identified and SLA management can be automated.

3 Rethinking SLA Specification

3.1 Precision

To understand the notion of precision in SLAs, consider the following example. Imagine a web service for ordering supplies on the Internet. Assume that "order-supplies" is an operation supported by this web service. In order to execute this operation, a client should send a "purchase-order" document and should obtain a "confirmation" document in response. Now, consider the following guarantee by the supplier:

95% of the time, the time for executing "order-supplies" will be less than 20 seconds.

While this may seem like a reasonable specification that can be automatically managed on the first look, there are several ways to interpret this:

a. **When?** When should a web service check for the compliance of this SLA? Here are some examples when the evaluation of this SLA can be triggered: (i) whenever a new "order-supplies" execution is completed, (ii) after every 10 "order-supplies" executions, (iii) at the end of the day, or (iv) just before the termination of the SLA. It is trivial to note that if this guarantee holds true in one of these cases, it does not necessarily hold in the rest.

b. **Which?** Which inputs should be considered in evaluating the SLA? In order to check whether the above guarantee is upheld or not, one can consider (i) all executions of "order-supplies" since the formation of the SLA, (ii) all executions of "order-supplies" since the beginning of that day, (iii) all executions of "order-supplies" no matter who the client is, (iv) all executions of "order-supplies" initiated by the client with which the SLA is created, or (v) the last 100 executions of "order-supplies", no matter when they happened.

c. **Where?** Where is the timing of the execution monitored? It can either be monitored at the client issuing the request or at the service provider handling the request. Usually, clients are interested in SLAs that give guarantees from their perspective as opposed to the service provider's perspective. On the other hand, it is quite hard for the service provider to make client-side guarantees since that could be influenced by factors that are outside the control of the service provider (e.g., the documents between the client and service provider often flow through other service providers such as ISPs).

d. **What and How?** In the above example, the metric of interest is the response time from the when the purchase-order is sent out to when a confirmation is received. The guarantee is on the 95th percentile. One way to interpret this is that out of every 100 executions, 95 will have a response time of less than 20 s. Another way to interpret this is that 95 out of every 100 executions will have a *mean* response time of less than 20 s. Metrics that are not as well defined as the typically well known metrics like availability, reliability, cost, and quality will further complicate how an SLA should be evaluated.

For a system to automate the management of SLAs, all of the above ambiguities should be removed from the SLA specification.

3.2 Flexibility

There are many kinds of interactions between consumers and providers of web services – single request-reply interaction, multiple back-and-forth messages, short-lived interactions, or interactions that last for several hours or days. Web services themselves are quite diverse. The metrics that are of interest in an SLA are, many times, quite specific to the web service. For example, a bookseller would like to provide a guarantee on the number of days it takes to deliver a book. A credit card authorization service provider would like to provide a guarantee on the security of the transmitted information. The vast diversity in the kinds of interactions as well as metrics that are of interest necessitates the need for a *flexible* SLA formalization. One should be able to build an automated SLA management system (or at least a framework for SLA management) using the flexible formalization.

Emerging standards such as Web Services Definition Language (WSDL) [14] and Web Service Flow Language (WSFL) [15] are creating flexible and generic interaction models for web services. For example, WSDL introduces concepts such as messages, operations, ports, and end points – which are useful for describing the operations of any web service. Similarly, WSFL introduces the notion of activities and flows – which are useful for describing both local business process flows and global flow of messages between multiple web services. So, one way to create a flexible SLA formalization is to build upon these concepts. In other words, one can create a flexible SLA formalization by associating "quality metrics" to the formalizations that are already defined in WSDL and WSFL. Here are some examples that show how such association can be done.

- Response time of a web service operation.
- Response time of a flow.
- Security of an operation.
- Number of times an activity is executed in a flow.
- Cost of executing an operation.
- Availability of an end point.

Another way to guarantee flexibility is to identify generic components in typical SLA specifications, which in turn are extensible so that new components can be defined by extending these base components.

4 SLA specification

An SLA typically has a date constraint (start date, end date, nextevaldate) and a set of Service Level Objective (SLO). An SLO in turn has typically a day–time (Mo-We, 6:00PM-8:00 PM) constraint and a set of clauses that make up the SLO. A clause is based on measured data. This is referred to as a *measuredItem*. A measuredItem can contain one or more *items*. A measuredAt element relates to the "where" part of the specification, referring to the side the measurements are taken (provider or consumer). A clause evaluation is done either when an event happens, e.g. say a message arrives, an operation completes or at a fixed time, say at 6PM. We call this an *evalWhen* component. This refers to the "When" component of the specification. Once the evalWhen trigger arrives, a set of samples of measuredItem are obtained applying a sampling function. The *evalOn* component determines the set of samples over which a condition is applied. The sample set is a constrained set of measured data that is obtained by applying the evalOn component. Examples of evalOn components may be a number or a time period, e.g. the 5 longest running transactions, or all the samples for last 24 hours. The evalOn component refers to the "which" part of the specification. A condition is thereafter applied on the sample set so obtained. This component is the actual *evalFunc* which is applied on the sample set so obtained from the evalOn component. An example of evalFunc would be average response time function < 5 ms. The evalFunc must be a mathematical function that is expressible in terms of its inputs and logic. The evalFunc refers to the "What" and "How" part of the specification. A sample clause like *At 6 PM the Average response time for the 5 longest running bookbuy transactions measured on the client side should be < 5 ms* can be broken up into a measuredItem (item:bookbuy transaction measuredAt:consumer), evalWhen(at 6PM), evalOn function(set of 5 longest running transactions) and the evalFunc (average response time < 5 ms). The evalAction could be notification to the administrator in this case.

```
SLA  = Dateconstraint Parties SLO*

Dateconstraint = Startdate Enddate Nextevaldate

SLO = Daytimeconstraint Clause*

Dateconstraint = Day* Time*

Clause = MeasuredItem EvalWhen EvalOn EvalFunc EvalAction

MeasuredItem = Item*

Item  = MeasuredAt ConstructType ConstructRef
```

An SLA specification thus must contain components as described in section 2. These are also mentioned in Appendix B as a part of the full SLA specification. Here we will bring out the essential components of the SLA specification.

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl= http://schemas/xmlsoap.org/wsdl/
 xmlns:wsfl = http://schemas/xmlsoap.org/wsfl/
 xmlns:wsml= http://schemas/hp.com/wsml/>

<xsd:element name="SLA" type="SLAtype" minOccurs = "1" maxOccurs = "unbounded">

<xs:complexType  name="SLAtype"/>
   <attribute name="name" type="NCName" use="required"/>
   <attribute name="SLAId" type="xsd:integer" use="optional"/>
   <xsd:sequence>
      <xsd:element  name="startDate"  type="xsd:date" />
      <xsd:element  name="endDate"  type="xsd:date" />
      <xsd:element  name="nextEvalDate"  type="xsd:date" />
      <xsd:element  name="provider"  type="xsd:String" />
      <xsd:element  name="consumer"  type="xsd:String" />
      <xsd:element  name="SLO" type="SLOType" minOccurs ="1"
          maxOccurs="unbounded" />
   <xsd:sequence>
</xsd:complexType>
```

An SLA specification essentially consists of a set of service level objectives that have to be achieved.

4.1 SLO Schema

The Service Level Objective provides the specifics of how a service provider will perform. For the purpose of the SLO Schema, the SLO is broken down into two items: a time constraint and one or more clauses.

```
<xsd:complexType  name="SLOtype"/>
   <attribute name="id" type="xsd:ID" use="required"/>
   <xsd:sequence>
      <xsd:element  name="timeConstraint"  type="timeConstraintType" />
      <xsd:element name="clause" type="clauseType" minOccurs ="1"
          maxOccurs="unbounded"/>
   <xsd:sequence>
</xsd:complexType>
```

4.1.1 SLO Time Constraint

The time constraint essentially describes the days and times during the week when an SLO will be in effect. This can be a single day or a set of days. When there is more than one day defined, it is assumed that the SLO is defined as space delimited list of days (i.e. Mo Tu We). It also has time description for specifying fixed times of day when the SLA will be effective (e.g. 6PM –8PM).

```
<xsd:complexType  name="timeConstraintType"/>
   <attribute name="id" type="xsd:ID" use="required"/>
   <xsd:all>
      <element name="daytimeConstraints" type="daytimeConstraintsType"/>
   </xsd:all>
</xsd:complexType>
```

```
<xs:complexType  name="daytimeConstraintType"/>
<xs:sequence>
<xsd:simpleType name="day">
<xsd:list itemType="dayType">
<xsd:restriction>
<xsd:minLength value = 1>
<xsd:maxLength value = 7>
</xsd:restriction>
</xsd:simpleType>
<xsd:element name="startTime" type="xs:time" />
<xsd:element name="endTime" type="xs:time" />
```

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


<xsd:simpleType name = ”dayType”>
<xsd:restriction base="xsd:NMTOKEN">
<xsd:enumeration value="Mo"/>
<xsd:enumeration value="Tu"/>
<xsd:enumeration value="We"/>
<xsd:enumeration value="Th"/>
<xsd:enumeration value="Fr"/>
<xsd:enumeration value = “Sa” />
<xsd:enumeration value = “Su” />
</xsd:restriction>
</xsd:simpleType>
```

4.1.2 SLO Clause

The clause provides the exact details on the expected performance. Five descriptors are used to capture these details namely *measuredItem*, *evalWhen, evalOn, evalFunction, evalAction.*

```
<xsd:complexType  name=”clauseType”/>
   <attribute name="id" type="xsd:ID" use="required"/>
   <xsd:sequence>
         <xsd:element  name=”measuredItem”  type=”measuredItemType”
            minOccurs="1" maxOccurs="1"/>
         <xsd:element name=”evalWhen” type=”evalWhenType”
            minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element  name=”evalOn”  type=”evalOnType”
            minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element  name=”evalFunc”  type=”evalFunctionType”
            minOccurs="1" maxOccurs="1"/>
         <xsd:element  name=”evalAction”  type=”evalActionType”
            minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType
```

The *measuredItem* gives a precise definition of what is being measured. The execution of an SLO is not always a single process, where the measurement is applied to every *measuredItem*. In some cases, measurements are applied across a set of *measuredItems* (e.g. over a set of operations). For SLA's that are based on WSDL and WSFL, the *measuredItem* may indicate which messages, operations, ports, activities or flows that are being measured.

The evalWhen element specifies the instant of evaluation of the SLA. An SLA evaluation can happen at a point of time say 6 PM everyday or it can happen when an event occurs, e.g. an operation or an activity completes.

Once the SLA evaluation is planned, it is necessary to obtain a set of data over which the evaluation takes place. This is described by the *evalOn* element . The evalOn element can refer to a set of samples, (e.g. over all instances in last 24 hours) or to a selected set sampled out of the available data (5 longest running transactions).

The evalFunc refers to the actual function that is applied on the set of data obtained by applying the evalOn component. The evalFunc component can be a function to ensure that the "average response time < 5 ms".

The evalAction refers to the action that needs to be activated. It could refer to a method call or a message exchange or any other type of action.

In order to ensure *flexibility* and extensibility each of the clause components are described in such a manner that they can be extended to express unforeseen service level specifications.

4.1.2.1 MeasuredItem

It is important to note that a *measuredItem* may be a collection of items. This enables the same measurement to be applied across one or more monitored items. For example, it allows a customer to specify a response time of 5 seconds across ALL of the service provider's operations. Additionally, it allows parts of a *flowModel* to be measured so that a subset of a flow (or set of activities) can be monitored.

```
<xsd:complexType  name="measuredItemType">
<attribute name="id" type="xsd:ID" use="required" default="NoName"/>
<xsd:sequence>
<xsd:element  name="item"  type="itemType" minOccurs = 1  maxOccurs =
"unbounded"/>
<xsd:sequence>
</xsd:complexType>

<xsd:complexType  name="itemType"/>
<xsd:sequence>
<xsd:element  name="measuredAt"  type="xsd:uri" minOccurrs = 0 maxOccurs =1/>
<xsd:element  name="constructType"  type="Name" />
<xsd:element name ="constructRef" type ="xsd:uri" />
<xsd:sequence>
</xsd:complexType>
```

The *measuredAt* element is required to clearly define where the measurements are being undertaken so that the SLA management engine can operate on the right set of management data. This could be a http URL , or a TCP address.

4.1.2.2 EvalFunc

An evalFunc when applied to a set of data always evaluates to a Boolean, i.e. The condition expressed through the evalFunction results in either TRUE or FALSE. For example, responseTime of a buyBook transaction < 5s will yield either TRUE or FALSE when evaluated on a particular buyBook transaction instance. A set of evaluation functions and corresponding terms will be defined in WSML (Web Services Management Language) namespace. The mathematical functions will be expressed using XML based languages to express mathematical logic, e.g. MathML[13]. As arbitrary evalFunc functions (in fact any function that is expressible in a mathematical logic that operates over a pool of data samples) can be described and used at will, this enables definition of a wide range of such functions ranging from simple to complex. This also enables the possibility of definition of evalFuncs in future that would be required in unforeseen service level specifications. This makes the SLA specification process flexible.

```
<xsd:complexType  name=”evalFunctType” abstract="true"/>
   <attribute name="id " type="xsd:ID" use="optional"/>
</xsd:complexType>
```

A few examples of these types of extensions are provided below. In addition, a library of the basic *evalFunc* measurement extensions may be defined that will be used by web services dynamically to define their typical service level agreements. Extensions for availability and response time (that would form part of the web service management language (WSML) namespace[1] are defined below,

```
<xsd:complexType name="evalFuncPercentageAvailabilityType"/>
<extension base="evalFuncType">
<documentation xml:lang="en">
“This function evaluates the availability of a measuredItem. If the
Availability percentage meets the specified value it returns true else
False”
</documentation>
<xsd:attribute  name=”percentage”  type=”xsd:int” />
</extension>
</xsd:complexType>


<xsd:complexType name="evalFuncResponseTimeType"/>
<extension base="evalFuncType">
<documentation xml:lang="en">
The response time function calculates the time a system takes to react to a
given input.  It then compares the response time to a threshold value.
The evaluation of the threshold is based on the type of operator.  The function
evaluates to "true" if it met the threshold requirements and "false" if it did
not".
</documentation>
<sequence>
<xsd:attribute  name=”operator”  type=”wsml:operatorType” />
<xsd:attribute  name=”threshold”  type=”xsd:duration”/>
</sequence>
</extension>
</xsd:complexType>


<xsd:complexType name="evalFuncResponseTimeOf5sType"/>
 <extension base="wsml:evalFuncResponseTime ">
<documentation xml:lang="en">
The responseTimeof5Sfunc checks to see if the response time was less than 5
seconds.
</documentation>
  <sequence>
   <xsd:attribute  name=”operator” type=”wsml:operatorType”  default =”LT”>
    <xsd:restriction base=”wsml:operatorType”>
      <xsd:enumeration  value=”LT”>
```

[1] WSML namespace is partially defined in appendix C

```
      </xsd:restriction>
      </xsd:attribute>

    <xsd:attribute  name=”threshold”  default=”P5S”>
      <xsd:restriction base=”xsd:duration”>
      <xsd:enumeration  value=”P5S”>
      </xsd:restriction>
      </xsd:attribute>
</extension>
</xsd:complexType>
```

4.1.2.3 EvalOn

The *evalOn* function is used for determining the samples over which the *evalFunc* is evaluated. The samples could be the longest lasting transactions, first 5 samples over an hour or the last 5 samples over an hour. If EvalOn is not provided, it is assumed that the *evalFunction* will be evaluated on each instance.

```
<xsd:complexType  name=”evalOnType” abstract="true"/>
   <attribute name="id" type="xsd:ID" use="optional" default="NoName"/>
</xsd:complexType>
```

Similar to *evalFunc*, the abstract *evalOnType* must be extended in order to provide meaning. The following are some examples of extension.

```
<xsd:complexType  name=”evalOnLastNtimePeriodsType” abstract="true"/>
<extension base="evalOnType">
<documentation>
Applies the evalFunc on the last N number of time periods (e.g. measured items
gathered over the last 5 hours)
</documentation>
<xsd:attribute  name=”number”  type=”xsd:int” minOccurs="1" maxOccurs="1"/>
<xsd:attribute  name=”timePeriod”  type=”xsd:duration”
minOccurs="1" maxOccurs="1"/>
</extension>
</xsd:complexType>

<xsd:complexType  name=”evalOnLastNItemsType” abstract="true"/>
<extension base="evalOnType">
<documentation> Applies the evalFunc on the last N items
</documentation>
<xsd:attribute  name=”number”  type=”xsd:int” minOccurs="1" maxOccurs="1"/>
</extension>
</xsd:complexType>

<xsd:complexType name="evalOnLastNhoursType"/>
<extension base="evalOnLastNtimePeriodsType"/>
<documentation> Applies the evalFunc on the measured items collected over the
past N hours.
</documentation>
<xsd:attribute  name=”timePeriod”  default=”P1H”>
<xsd:restriction base=”xsd:duration”>
<xsd:enumeration  value=”P1H”>
</xsd:restriction>
</xsd:attribute>
```

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


<xsd:complexType name="evalOnLast5hoursType"/>
<extension base="evalOnLastNtimePeriodsType"/>
 <documentation>
 Applies the evalFunc on the measured items collected over the past 5 hours.
 </documentation>
      <xsd:attribute  name="number"  default="5">
      <xsd:restriction base="xsd:int">
      <xsd:enumeration  value="5">
      </xsd:restriction>
      </xsd:attribute>

      <xsd:attribute  name="timePeriod"  default="P1H">
      <xsd:restriction base="wsml:duration">
      <xsd:enumeration  value="P1H">
      </xsd:restriction>
      </xsd:attribute>
   </extension>
</xsd:complexType>
```

4.1.2.4 EvalWhen

The *evalWhen* element describes when the evaluation of the SLO happens. Examples of the evalWhen can be an instant of time at which the SLO is evaluated, say at 6 PM or when an attribute of measuredItem data changes, say an activity completion will change its status to completed. An absence of evalWhen element means the SLO is evaluated everytime an instance of management data is generated.

```
<xsd:complexType  name="evalWhenType"/>
<attribute name="id" type="xsd:ID" use="required"/>
</xsd:complexType>
```

The following are three basic extensions to the *evalWhenType*:

```
<xsd:complexType name="evalWhenTimeOccursType" abstract="true"/>
<extension base="evalWhenType/>
<documentation>Execute the evalFunc when a specific time occurs (e.g. at 6pm).
</documentation>
<xsd:element  name="time"  type="xsd:time" minOccurs = 1  maxOccurs = "1"/>
</extension>
</xsd:complexType>


<xsd:complexType name="evalWhenTimePeriodOccursType" abstract="true"/>
<extension base="evalWhenType/>
<documentation>Execute the evalFunc when a specific time period has passed
(e.g. evaluate every 24 hours)
</documentation>
<xsd:element  name="timePeriod"  type="xsd:duration" minOccurs = 1   maxOccurs
= "1"/>
</extension>
```

```
</xsd:complexType>



<xsd:complexType name="evalWhenEventOccursType" abstract="true"/>
<extension base="evalWhenType/>
<documentation>Execute the evalFunc when a specific event has occurred (e.g.
message is received).
</documentation>
<xsd:element  name="measuredItemAttribute"
type="measuredItemAttributeType " minOccurs = 1  maxOccurs = "1"/>
</extension>
</xsd:complexType>

<xsd:complexType  name="measuredItemAttributeType"/>
<attribute name="id" type="xsd:ID" use="optional"/>
<attribute  name="ref" type ="xsd:uri" minOccurs ="0" maxOccurs = "1"
use="required"/>
<attribute  name="value" type ="xsd:string" minOccurs ="0" maxOccurs = "1"  use
="required"/>
</xsd:complexType>
```

Below is an example of an evalWhen instance that leverages one of the above extensions:

```
<evalWhenTimeOccurs   time="6pm">
   <documentation>Evaluate evalFunc at 6pm
   </documentation>
</evalWhenTimeOccurs>
```

4.1.2.5 Examples

The *SLO Clause* is clearly the most complex part of the SLA. For this reason, we provide this section to illustrate how a number of the traditional service level objectives would be specified using the schema defined above.

All examples will use the same measuredItem and evalFunctionType. The measuredItem used in the examples is referred to as the *stationaryPurchase.* It can be defined in two different ways as follows:

```
   <xsd:element name="stationaryPurchase" type ="stationaryPurchaseMIType">

   <xsd:complexType name ="stationaryPurchaseMIType">
   <attribute name="id" type="xsd:ID" use="required" use="optional"/>
   <xsd:restriction base="measuredItemType">
   <xsd:element  name="item"  type="stationaryItemType" minOccurs = 1
   maxOccurs = "1"/>
   </xsd:restriction>
   </xsd:complexType>

   <xsd:complexType  name="stationaryItemType"/>
   <xsd:restriction base="itemType">
   <xsd:sequence>
   <xsd:element  name="constructType"  type="wsfl:flowModel"/>
   <xsd:element name ="constructRef" type="xsd:uri" default=
   "www.stationary.com/flowModel">
```

```
</xsd:sequence>
</xsd:restriction>
</xsd:complexType>
```

OR

```
<measuredItem id = "stationaryPurchase">
<item>
<constructType>flowModel</constructType>
<constructRef>www.stationary.com/flowModel</constructRef>
</item>
</measuredItem>
```

The evalFunc used is the avgResponseTimeOf5S in all the following examples. Few examples of some commonplace clauses are explained as follows:

1. **Average Response time for every stationary purchase transaction should be less than 5s.**

   This clause can be specified in the following manner:

   ```
   <clause id="avgRTforEachTransaction"
      <measuredItem idref = "stationaryPurchase" >
      <evalFuncResponseTime idref = "avgResponseTimeOf5S">
   </clause>
   ```

   or

   ```
   <clause id = "avgRTforEachTransaction">
      <stationaryPurchase></stationaryPurchase>
      <avgResponseTimeOf5s></avgResponseTimeOf5s>
   </clause>
   ```

   Note: Since this measurement is applied to every instance, there is no EvalWhen or EvalOn components in the specification. No EvalAction is specified.

2. **Average response time of last 5 transactions at any given instance of time should be less than 5 seconds.**

   ```
   <clause id="AvgRTofLast5Transactions"
      <measuredItem idref = "stationaryPurchase" >
      <evalFuncResponseTime idref = "avgResponseTimeOf5S">
      <evalOnLastNitems  N = "5">
   </clause>
   ```

   or

   ```
   <clause id="AvgRTofLast5Transactions"
      <stationeryPurchase></stationaryPurchase>
      <avgResponseTimeOf5s></avgResponseTimeOf5s>
   ```

```
      <evalOnLast5Items> </evalOnLast5Items>
   </clause>
```

Note: EvalWhen is "per instance" which is default.

3. **Average response time of all transactions thus far is less than 5 seconds.**

```
<clause name="AvgRTofAllTransactions"
   <stationeryPurchase></stationaryPurchase>
   <avgResponseTimeOf5s></avgResponseTimeOf5s>
</clause>
```

Note: EvalWhen and EvalOn are default. No EvalAction.

4. **Average response time of all transactions in the last 2 minutes at any given time thus far is less than 5 seconds.**

```
<clause name="AvgRTofTransactionsEvery2min"
   <stationeryPurchase></stationaryPurchase>
   <avgResponseTimeOf5s></avgResponseTimeOf5s>
   <evalOnLastNtimePeriods number =”2” timePeriod =”P1M” >
  </evalOnLastNtimePeriods>
</clause>
```

Note: EvalWhen is default because it is on every instance. No EvalAction is specified.

5. **Average response time of 5 longest-lasting transactions in the last 24 hours should be less than 5 seconds.**

```
<clause name="AvgRTofLast5LongestTransIn24Hrs"
<stationeryPurchase></stationaryPurchase>
<avgResponseTimeOf5s></avgResponseTimeOf5s>
<evalOnLastNtimePeriods number =”24” timeperiod =”P1H”>
</evalOnLastNtimePeriods>
<evalOnNlongestTransactions number = “5”>
</evalOnNlongestTransactions>
</clause>
```

Note: EvalWhen is "per instance". At any given instance, we should be able to look at the 5 longest-lasting transactions over a period of 24 hours.

6. **Average response time of all transactions, averaged at the end of the day, should be less than 5 seconds.**

```
<clause name="AvgDailyResponseTime"
<stationeryPurchase></stationaryPurchase>
<avgResponseTimeOf5s></avgResponseTimeOf5s>
<evalOnLastNtimePeriods number =”24” timeperiod =”P1H” >
</evalOnLastNtimePeriods>
<evalWhenTimeOccurs time =”11:59PM”>
</evalWhenTimeOccurs>
</clause>
```

```
Note: The evaluation is specifically done at the end of the day. EvalWhen
indicates when the measurement should be taken.
```

7. **Average response time of all transactions in the last 10 minutes averaged at 5 minute intervals, should be less than 5 seconds.**

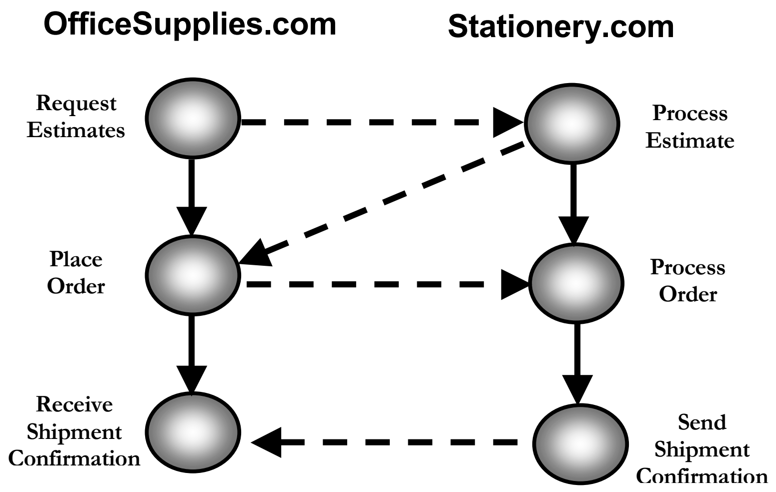
```
<clause name="AvgRTOver10minutes"

<stationeryPurchase></stationaryPurchase>
<avgResponseTimeOf5s></avgResponseTimeOf5s>
<evalOnLastNtimePeriods number =”10” timeperiod =”P1M” >
</evalOnLastNtimePeriods>
<evalWhenTimePeriodOccurs  timeperiod =”P5M”>
</evalWhenTimePeriodOccurs>
</clause>
```

5 Specifying SLA for Web Services: An Example

The following is an example of an SLA that is based on a web service’s WSDL and WSFL descriptions. It focuses on the relationship between an office supplies broker and one if its suppliers. The WSDL and WSFL for these services are kept simple, so that the focus is placed on the supporting SLAs.

Diagram 1 gives an overview of the flow between the service provider (stationery.com) and the customer (officesupplies.com).



OfficeSupplies.com has established a service level agreement with Stationery.com, based on the WSDL and WSFL defined by these two services. The focus of this SLA is on **four** key performance agreements:

1. Stationery.com will respond to estimate requests in less than 5 minutes.
2. Stationery.com will provide 100% availability to all "estimate" requests, Monday through Friday.
3. Stationery.com will provide 90% availability to all "process order" requests, Monday through Friday.
4. Stationery.com will process and ship orders in less than 24 hours of receiving the "process order" request.

The WSDL and WSFL for these services are defined in Appendix D. Section 5.3 presents the SLA between OfficeSupplies.com and Stationery.com, based on the constructs presented in this paper.

Assumptions:

- Process is significantly oversimplified in order to make example easy to follow.

- Port bindings are ignored in order to simplify the example.

All items are shipped at the same time. Realistically, back-ordered items would ship at a later date. For the purpose of keeping this example simple, we require that all items ship at the same time. TheWSDL and WSFL definitions are mentioned in Appendix C.

Service Level Agreement for web service OfficeSupplies.com

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

  targetNamespace =
   http://www.OfficeSupplies.com/WebServices/Messages/SLA
   xmlns:ssla="http://www.OfficeSupplies.com/WebServices/Messages/SLA"
   xmlns:osn= http://www.officesupplies.com/WebServices/Messages/Procurement"

 <import namespace=
   "http://www.officesupplies.com/WebServices/Messages/Procurement"
   location="http://www.officesupplies.com/WebServices/Messages/Procurement" />


<daytimeConstraint id="alldayConstraints">
```

```
<day> MO Tu We Th Fr Sa Su </day>
<startTime>12AM</startTime>
<endTime>12 PM</endTime>
</daytimeConstraint>

<daytimeConstraint id="workdayConstraints">
<day> MO Tu We Th Fr </day>
<startTime>12AM</startTime>
<endTime>12 PM</endTime>
</daytimeConstraint>

<daytimeConstraint id="weekendConstraints">
<day>Sa Su</day>
<startTime>12AM</startTime>
<endTime> 12PM </endTime>
</daytimeConstraint>


<SLA  id = “stationary.com/SLA1”>
<startDate>01-01-01</startDate>
<endDate>01-01-02</endDate>
<nextEvalDate>01-01-02</nextEvalDate>
<provider>Stationery.com</provider>
<consumer>OfficeSupplies.com</consumer>

<SLO id = “SLO1”>
<daytimeConstraint idref=”alldayConstraints”>
<clause id = ”SLO1Clause1”>
<measuredItem id =”estimateMeasuredItem”>
<item>
<constructType>wsdl:operation</constructType>
<constructRef>osn:processEstimate</constructRef>
</item>
</measuredItem>
<avgResponseTimeOf5s></avgResponseTimeOf5s>
</clause>
</SLO>

<SLO id = “SLO2”>
<daytimeConstraint idref=”workdayConstraints”>
<clause id = ”SLO2Clause1”>
<measuredItem idref =”estimateMeasuredItem”>
<evalFuncPercentageAvailabilty  percentage= “100”>
</evalFuncPercentageAvailability>
</clause>
</SLO>

<SLO  id = “SLO3”>
<daytimeConstraint idref=”workdayConstraints”>
<clause id = ”SLO3Clause1”>
<measuredItem id =”processOrderMeasuredItem”>
<item>
<constructType>wsdl:operation</constructType>
<constructRef>osn:processOrder</constructRef>
</item>
</measuredItem>
```

```
<evalFuncPercentageAvailabilty  percentage= "90">
</evalFuncPercentageAvailability>
</SLO>

<SLO id = "SLO4">
<daytimeConstraints idref="alldayConstraints">
<clause id ="SLO4Clause1">
<measuredItem idref ="processOrderMeasuredItem">
<evalFuncResponseTime operator ="LT" threshold="P24H" >
</evalFuncResponseTime>
</clause>
</SLO>
```

6 Conclusion

We have identified the base components in typical SLA specifications, so as to define SLAs in a precise manner. A clear enunciation of the base components of a typical SLA specification will help automate their monitoring and compliance. We have also made sure that these base components are flexible and extensible, so that unforeseen SLAs can be specified. The proposed specification has been used to specify SLAs between typical web services defined in standard Web Service Languages.

Acknowledgements

We would like to thank Klaus Wurster and Aad van Moorsel for their feedback and inputs on this paper.

References

1. Campbell A, Aurrecoechea C., Hauw L .QoS review Architectures, Proceedings of the 4th International Workshop on Quality of Service (IWQoS), 1996

2. Svend Frolund, Jari Koistinen. Quality-of-Service Specification in Distributed Object Systems. HPL-98-159.

3. Bhoj P, Singhal S, Chutani S. SLA Management in Federated Environments. HPL-98-203

4. Dirk Thiβen and Helmut Neukirchen. Internet Trading and Load Balancing for Efficient Management of Services in Distributed Systems. Third International IFIP/GI Working Conference, USM 2000. Munich, Germany, September 12-14, 2000. In Proceedings Lecture Notes in Computer Science 1890 titled "Trends in Distributed Systems: Towards a Universal Service Market".

5. Jerome Daniel, Bruno Traverson, and Sylvie Vignes. A QoS Meta Model to Define a Generic Environment for QoS Management. Third International IFIP/GI Working Conference, USM 2000. Munich, Germany, September 12-14, 2000. In Proceedings Lecture Notes in Computer Science 1890 titled "Trends in Distributed Systems: Towards a Universal Service Market". Springer Verlag.

6. Long T P, Jong W B, Woon HJ. Management of service level agreements for multimedia Internet service using a utility model. IEEE communications Managezine Vol 39, no.5, May 2001

7. Forbath T. Why and how of SLAs [service level agreements]. Business Communications Review, Vol 28. No. 2, Feb 1998

8. Chatterjee BS, Sydir M, Lawrence T. Taxonomy for QoS specifications. In the proceedings of WORDS'97, February, 1997

9. Lewis L, Ray P. Service Level Management: Definition, Architecture, and Research Challenges. In the proceedings of IEEE GlobeCom'99.

10. Katcgabaw M, Lutfiyya H, and Bauer M. Driving Resource Management with Application-Level Quality of Service Specifications. In the proceedings of ICE 98, USA.

11. Java 2 Enterprise Edition (J2EE) . http://java.sun.com/j2ee/

12. .Net at Microsoft http://www.microsoft.com/net/

13. Mathematical Markup Language (MathML) specification. http://www.w3.org/TR/REC-MathML/

14. Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl

15. Web Services Flow Language (WSFL). www.ibm.com/software/webservices

16. R. Sturm, W. Morris, M. Jander. Foundations of Service Level Management. (SAMS, 4/00).

17. Tele Management Forum SLA Management Handbook. http://www.tmfcentral.com/kc/repository/documents/GB917v1.5.pdf. GB917, public evaluation version 1.5, June 2001.

Appendix A Service Level Indicators

Metrics	Definition	Applicable for following web service constructs	Target specification
Availability	Availability of an entity minus the impact time from any events (scheduled or unexpected) other than loss of network or system availability	Port Port Type ServiceProvider Endpoint Flow	Expressed as percentage or in time
Response Time	The time taken for an entity to complete a client request and return a response	Operation Activity Flow	normally specified as service to complete X% of transactions of type Y to be completed within Z seconds
Throughput	Number in unit of time	Operation Activity Flow Endpoint Port	Expressed as tps
Security	The security at different levels need to be agreed upon in the SLAs	Service Provider Port Operations Activity Flow Endpoint	Level supported 1. High 2. Medium 3. None
Payment Rate	Rate at which the service/transactions are charged	Operations Activity Port Service Provider Endpoint Flow	The payment rate can be a subscription based model or can be expressed per transaction type or instance level.
Problem Response	The time required for a client to receive a response after reporting a problem	Service Provider Port Endpoint	1-High Priority[md] expressed in target time 2-Medium Priority[md] expressed in target time 3-Low Priority[md] expressed in target time
Problem Circumvention or Resolution Time	The time required for a client to receive a circumvention or a solution after reporting a problem	Service Provider Port EndPoint	Expressed as times for various categories of problems. High category problems have to be resolved faster than low category problems.
Repeat Trouble Rate	Number of times the trouble is repeated before it is escalated.	Service Provider Port Endpoint	This is usually expressed as a rate
Account set up time	Time taken to create and set up new accounts on the system	Service Provider EndPoint Port	Expressed as unit of time

Appendix B: Web Service SLA template

The schema for the web service template SLA can be described as follows:

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
<xsd:annotation>
<xsd:documentation  xml:lang ="en">
      SLA template schema for web services
</xsd:documentation>
</xsd:annotation>

<xsd:element  name="purpose"    type="xsd:string" />
<xsd:element  name="parties"    type="partyType" />
<xsd:element  name= "validityPeriod" type= "xsd:decimal" />
<xsd:element  name = "scope"  type="scopeType"/>
<xsd:element  name = "restrictions"  type="restrictionsType"/>
<xsd:element  name = "penalties"  type="penaltiesType"/>
<xsd:element  name = "optionalServices"  type="optionalServicesType"/>
<xsd:element  name = "exclusions"  type="exclusionsType"/>
<xsd:element  name = "administration"  type="adminType"/>
<xsd:element   name  =  "SLO"   type="SLOType"  minOccurs  =  "1"  maxOccurs  =
"unbounded"/>

<xsd:complexType  name ="scopeType">
<xsd:sequence>
<xsd:element name ="scopedesc" type = "xsd:string"  minOccurs ="1" />
<xsd:element name ="servicehours" type = "xsd:string"  minOccurs ="1" />
<xsd:element name ="responsibilities" type = "xsd:string"  minOccurs ="1" />
<xsd:element name ="supportHours" type = "xsd:string"  minOccurs ="1" />
</xsd:sequence>
</xsd:complexType>

.
.
.
</xsd:schema>
```

Appendix C: Web Services Management Language NameSpace Definitions

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema”
 xmlns:wsdl= http://schemas/xmlsoap.org/wsdl/
 xmlns:wsfl = http://schemas/xmlsoap.org/wsfl/
 targetNameSpace= http://www.hp.com/wsml>

<xsd:complexType name="webServiceItemType">
<complexContent>
<xsd:extension base="itemType">
<xsd:sequence>
<xsd:restriction  base ="constructType">
<xsd:enumeration  value ="wsdl:message">
<xsd:enumeration  value ="wsdl:portType">
<xsd:enumeration  value ="wsdl:port">
<xsd:enumeration  value ="wsdl:binding">
<xsd:enumeration  value ="wsdl:service">
<xsd:enumeration  value ="wsfl:serviceProviderType">
<xsd:enumeration  value ="wsfl:serviceProvider">
<xsd:enumeration  value ="wsfl:flowModelType">
<xsd:enumeration  value ="wsfl:flowModel">
<xsd:enumeration  value ="wsfl:activityType">
<xsd:enumeration  value ="wsfl:dataLinkType">
<xsd:enumeration  value ="wsfl:controlLinkType">
<xsd:enumeration  value ="wsfl:plugLinkType">
<xsd:enumeration  value ="wsfl:globalModel">
<xsd:enumeration   value="wsfl:globalModelType">
<xsd:enumeration   value="wsfl:spawnResult">
</xsd:restriction>
<xsd:sequence>
</complexContent>
</xsd:complexType>


<xsd:element name=”operator” type =”operatorType”>
<xsd:element name=”timeThreshold” type =”timeThresholdType”>

<xsd:complexType  name  = ”operatorType”/>
<xsd:restriction  base  = ”xsd:NMTOKEN”>
<xsd:enumeration  value = ”LT”/>
<xsd:enumeration  value = ”GT“/>
<xsd:enumeration  value = ”EQ“/>
<xsd:enumeration  value = ”GE”/>
<xsd:enumeration  value = ”LE"/>
<xsd:sequence>
</xsd:complexType>

<xsd:complexType name = “timeThresholdType”>
<attribute  name = “number”  type =”xsd:integer”>
<attribute name = “timeUnit”     type = timeUnitType>
</xsd:complexType>

<xsd:complexType  name  = ”timeUnitType”/>
```

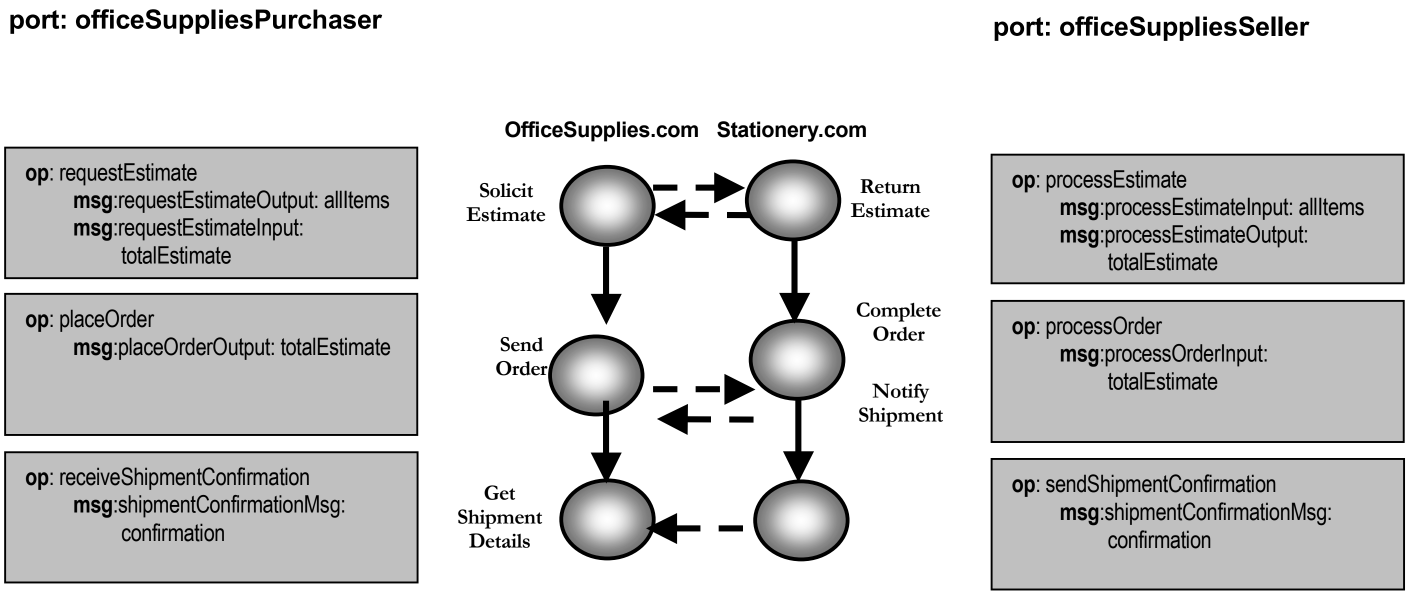
```
<xsd:restriction  base  = ”xsd:NMTOKEN”>
<xsd:enumeration  value = ”ms”/>
<xsd:enumeration  value = ”s“/>
<xsd:enumeration  value = ”min“/>
<xsd:enumeration  value = ”hr“/>
<xsd:enumeration  value = ”day”/>
<xsd:enumeration  value = ”month”/>
<xsd:enumeration  value = “year”/>
<xsd:sequence>
</xsd:complexType>
```

Appendix D: WSDL,WSFL definitions of the web service example

6.1 WSDL Definitions

The following diagram provides an overview of the ports that are externalized by the two flow models. WSDL definitions of the ports and messages are provided below.   Key terms are highlighted in bold.

Note that the element **`totalEstimate`** plays an important role in the overall flow. This element carries a description of all ordered items, along with their estimated costs and the **`flowInstanceId`** that is generated by the Stationery.com flow. In essence, it provides the unique identifier for the entire transaction.


port: officeSuppliesPurchaser
port: officeSuppliesSeller
OfficeSupplies.com
Stationery.com
op: requestEstimate
msg:requestEstimateOutput: allItems
msg:requestEstimateInput: totalEstimate
op: placeOrder
msg:placeOrderOutput: totalEstimate
op: receiveShipmentConfirmation
msg:shipmentConfirmationMsg: confirmation
Solicit Estimate
Send Order
Get Shipment Details
Return Estimate
Complete Order
Notify Shipment
op: processEstimate
msg:processEstimateInput: allItems
msg:processEstimateOutput: totalEstimate
op: processOrder
msg:processOrderInput: totalEstimate
op: sendShipmentConfirmation
msg:shipmentConfirmationMsg: confirmation

NAME SPACE DEFINED:

```
<?xml version="1.0"?>
<definitions name="officesupplies"
targetNamespace=
   "http://www.OfficeSupplies.com/WebServices/Messages/Procurement"
   xmlns:osn=
      http://www.OfficeSupplies.com/WebServices/Messages/Procurement/
   xmlns="">
```

TYPES DEFINED:

```
<types>
<schema
   xmlns= "http://www.w3.org/2000/10/XMLSchema"
   targetNamespace =
      "http://www.OfficeSupplies.com/WebServices/Messages/Procurement"
   xmlns:osn=
      "http://www.OfficeSupplies.com/WebServices/Messages/Procurement">

<complexType name="item">
      <all>
         <element name="Manufacturer" type="string"/>
         <element name="Product" type="string"/>
         <element name="ModelNumber" type="string"/>
         <element name="Color" type="string"/>
      </all>
</complexType>

<element name="allItems" type="osn:item"
   minOccurs="1" maxOccurs="unbounded"/>

<complexType="itemCost">
   <all>
      <element ref="osn:item"/>
      <element name="cost" type="float"/>
   </all>
</complexType>

<element name="allItemsAndCosts">
   <complexType>
      <all>
         <element name="allItemCosts" type="osn:itemEstimate"
            minOccurs="1" maxOccurs="unbounded"/>
         <element name="total" type="float"/>
      </all>
   </complexType>
</element>

<element name="totalEstimate">
   <complexType>
      <all>
         <element name="estimatedTotalCost" type="osn:allItemsAndCosts"/>
         <element name="supplierFlowInstanceId" type="wsfl:FlowInstanceId"/>
         <element name="estimatedShipmentDate" type="date"/>
         <element name="estimatedShipmentTime" type="time"/>
      </all>
   </complexType>
</element>

<element name="order">
   <complexType>
      <all>
         <element name="orderedItems" type="osn:allItems"/>
         <element name="supplierFlowInstanceId" type="wsfl:FlowInstanceId"/>
      <all>
   <complexType>
</element>

<element name="orderConfirmation">
   <complexType>
      <all>
         <element name="totalCost" type="osn:allItemsAndCosts"/>
         <element name="supplierFlowInstanceId" type="wsfl:FlowInstanceId"/>
      <all>
   <complexType>
</element>
```

Note: FlowInstanceId of supplier's flow is returned with estimate.

```
<element name="shipmentConfirmation"
   <complexType>
      <all>
         <element name="actualShipmentDate" type="date"/>
         <element name="acutalShipmentTime" type="time"/>
         <element name="supplierFlowInstanceId" type="wsfl:FlowInstanceId"/>
      </all>
   </complexType>
</element>

</schema>
</types>
```

MESSAGES DEFINED:

Stationery.com

```
<message name="processEstimateInput" >
   <part name= "estimate" element="osn:allItems"/>
</message>

<message name="processEstimateOutput" >
   <part name= "estimate" element="osn:totalEstimate"/>
</message>

<message name="processOrderInput" >
   <part name= "order" element="osn:order"/>
</message>

<message name="processOrderOutput" >
   <part name= "orderConfirmation" element="osn:orderConfirmation"/>
</message>

<message name="sendShipmentConfirmationOutput">
   <part name="confirmation" element="osn:shipmentConfirmation"/>
</message>
```

OfficeSupplies.com

```
<message name="requestEstimateOutput" >
   <part name= "estimate" element="osn:allItems"/>
</message>

<message name="requestEstimateInput" >
   <part name= "estimate" element="osn:totalEstimate"/>
</message>

<message name="placeOrderOutput" >
   <part name= "order" element="osn:order"/>
</message>

<message name="placeOrderInput" >
   <part name= "orderConfirmation" element="osn:orderConfirmation"/>
</message>

<message name="getShipmentConfirmationInput">
   <part name="confirmation" element="osn:shipmentConfirmation"/>
</message>
```

INTERNAL MESSAGES DEFINED:

Stationery.com

```
<message name="requestedEstimate">
   <part name="requestedItems" element="osn:allItems"/>
   <part name="estimatedItems" element="totalEstimate"/>
</message>

<message name="processedOrder">
   <part name="requestedOrder" element="osn:order" />
   <part name="confirmedOrder" element="osn:orderConfirmation"/>
</message>

<message name="shipmentDetails">
   <part name="confirmationDetails" element="osn:shipmentConfirmation"/>
</message>
```

OfficeSupplies.com

```
<message name="request">
   <part name="requestedItems" element="osn:allItems"/>
   <part name="estimatedItems" element="totalEstimate"/>
</message>

<message name="order">
   <part name="orderedItems" element="osn:order" />
   <part name="orderConfirmed" element="osn:orderConfirmation"/>
</message>

<message name="shipmentInfo">
   <part name="shippingInfo" element ="osn:shipmentConfirmation"/>
</message>
```

PORTS DEFINED:

```
<definitions name="officeSuppliesPortTypes"
   targetNamespace =
      "http://www.OfficeSupplies.com/WebServices/Messages/Procurement"
   xmlns:osn= http://www.OfficeSupplies.com/WebServices/Messages/Procurement">
```

Stationery.com

```
<portType name="officeSuppliesSeller">
   <operation name="processEstimate">
      <input message="osn:processEstimateInput"/>
      <output message="osn:processEstimateOutput"/>
   </operation>

   <operation name="processOrder">
      <input message="osn:processOrderInput"/>
      <output message="osn:processOrderOutput"/>
   </operation>

   <operation name="sendShipmentConfirmation">
      <output message="osn:sendShipmentConfirmationOutput"/>
   </operation>
</portType>
```

OfficeSupplies.com

```
<portType name="officeSuppliesPurchaser">
   <operation name="requestEstimate">
      <output message="osn:requestEstimateOutput"/>
      <input message="osn:requestEstimateInput"/>
   </operation>

   <operation name="placeOrder">
      <output message="osn:placeOrderOutput"/>
      <input message="osn:placeOrderInput"/>
   </operation>

   <operation name="getShipmentConfirmation">
      <input message="osn:getShipmentConfirmationInput"/>
   </operation>
</portType>

</definitions>
```

6.2 Flow Model Definitions

6.2.1 Service Provider Type Definitions

```
<definitions name="officeSuppliesPortTypes"
   targetNamespace =
      "http://www.OfficeSupplies.com/WebServices/Messages/Procurement"
   xmlns:osn=
      "http://www.OfficeSupplies.com/WebServices/Messages/Procurement">

<serviceProviderType name="supplierFlow">
   <portType name="osn:processEstimate"/>
   <portType name="osn:processOrder"/>
   <portType name="osn:sendShipmentConfirmation"/>
</serviceProviderType>

<serviceProviderType name="purchaserFlow">
   <portType name="osn:requestEstimate"/>
   <portType name="osn:placeOrder"/>
   <portType name="osn:getShipmentConfirmation"/>
</serviceProviderType>
```

6.2.2 Stationery.com Flow Model

```
<flowModel name="purchaseSupplies" serviceProviderType="supplierFlow">

<flowSource name="purchaserFlowSource">
   <output name="processInstanceData" message="osn:requestedEstimate"/>
</flowSource>

<serviceProvider name="purchaser" type="purchaserFlow"/>
<serviceProvider name="supplier" type="supplierFlow"/>

<export lifecycleAction="spawn">
   <target portType="osn:officeSuppliesSeller" operation="processEstimate">
      <map sourceMessage="processEstimateInput"
         targetMessage="processInstanceData"
         targetPart="requestedItems"/>
      <map sourceMessage="processInstanceData"
         sourcePart="estimatedItems"
         targetMessage="processEstimateOutput"
         targetPart="estimate"/>
   </target>
</export>

<activity name="completeOrder">
   <input name="completeOrderInput" message="osn:processedOrder"/>
   <output name="completeOrderOutput" message="osn:shipmentDetails"/>
   <performedBy serviceProvider="purchaser"/>
   <implement>
      <export portType="officeSuppliesSeller" operation="processOrder">
         <map sourceMessage="processOrderInput"
            sourcePart="order"
            targetMessage="completeOrderInput"
            targetPart="requestedOrder"/>
         <map sourceMessage="completeOrderInput"
            sourcePart="confirmedOrder"
            targetMessage="processOrderOutput"
            targetPart="orderConfirmation"/>
      </export>
   </implement>
</activity>

<activity name="notifyShipment" >
   <input name="shippingDetailsInput" message="osn:shipmentDetails"/>
   <performedBy serviceProvider="purchaser"/>
   <implement>
      <export portType="osn:officeSuppliesSeller"
         operation="sendShipmentConfirmation"
         <map sourceMessage="shippingDetailsInput"
            sourcePart=" confirmationDetails "
            targetMessage="sendShipmentConfirmationOutput"
            targetPart="confirmation"/>
</activity>

</flowmodel>
```

6.3 OfficeSupplies.com Flow Model

```
<flowModel name="purchaseSupplies" serviceProviderType="supplierFlow">

<flowSource name="purchaserFlowSource">
   <output name="processInstanceData" message="osn:request"/>
</flowSource>

<serviceProvider name="purchaser" type="purchaserFlow"/>
<serviceProvider name="supplier" type="supplierFlow"/>

<export lifecycleAction="spawn">
   <target portType="osn:officeSuppliesPurchaser"
      operation="requestEstimate">
      <map sourceMessage="processInstanceData"
         sourcePart="requestedItems"
         targetMessage="requestEstimateOutput"
         targetPart="estimate"/>
      <map sourceMessage="requestEstimateInput"
         targetMessage="processInstanceData"
         targetPart="estimatedItems"/>
   </target>
</export>
```

```
<activity name="sendOrder">
   <input name="sendOrderInput" message="osn:order"/>
   <output name="sendOrderOutput" message="osn:order"/>
   <performedBy serviceProvider="supplier"/>
   <implement>
      <export portType="officeSuppliesPurchaser"
         operation="placeOrderOutput">
         <map sourceMessage="sendOrderInput"
            sourcePart="orderedItems"
            targetMessage="placeOrderOutput"
            targetPart="order"/>
         <map sourceMessage="placeOrderInput"
            sourcePart="orderConfirmation"
            targetMessage="sendOrderInput"
            targetPart="orderConfirmed"/>
      </export>
   </implement>
</activity>

<activity name="getShipmentDetails" >
   <input name="getShipmentDetailsInput" message="osn:shipmentInfo"/>
   <performedBy serviceProvider="supplier"/>
   <implement>
      <export portType="osn:officeSuppliesPurchaser"
         operation="getShipmentConfirmation"
         <map sourceMessage="getShipmentConfirmationInput"
            sourcePart="confirmation"
            targetMessage="getShipmentDetailsInput"
            targetPart="shippingInfo"/>
</activity>

</flowmodel>
```