Automated SLA Monitoring for Web Services

Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Aad van Moorsel, and Fabio Casati

HP Laboratories, 1501 Page Mill Road, Palo-Alto, CA 94304
{asahai, vijaym, msayal, aad, casati}@hpl.hp.com

Abstract. SLA monitoring is difficult to automate as it would need precise and unambiguous specification and a customizable engine that collects the right measurement, models the data and evaluates the SLA at certain times or when certain events happen. Also most of the SLA neglect client side measurement or restrict SLAs to measurements based only on server side. In a cross-enterprise scenario like web services it will be important to obtain measurements at multiple sites and to guarantee SLAs on them. In this article we propose an automated and distributed SLA monitoring engine.

1 Introduction

A web service can be described broadly as a service available via the web that conducts transactions. E-businesses set up Web Services for clients and other Web Services to access. They have a Uniform Resource Locator at which they can be accessed and have a set of Interfaces that can be utilized to access them. Web services that are capable of intelligent interaction would be able to discover and negotiate with each other, mediate on behalf of their users and compose themselves into more complex services. This composition could be static or dynamic. Emerging standards such as SOAP, UDDI, and WSDL[1] are steps in this direction. As these web services interact and delegate jobs to each other they would need to create and manage *Service Level Agreements* amongst each other. Service Level Agreements (SLA)s are signed between two parties for satisfying clients, managing expectations, regulating resources and controlling costs. SLA management involves the procedure of signing SLAs thus creating binding *contracts*, monitoring their compliance and taking control actions to enable compliance.

SLA monitoring is difficult to automate as it would need a precise and unambiguous definition of the SLA as well as a customizable engine that understands the specification, customizes instrumentation, collects the necessary data, models it in a logical manner and evaluates the SLA at the required times. Also most of the SLAs are about measurement located at a particular location. In a federated cross-enterprise scenario, as is the case in web services, measurements may often have to be collected at the client side as opposed to the server side to guarantee Quality of Experience (for e.g. guarantee on response time as measured on the client side when the number of con-

[1] SOAP: Simple Object Access Protocol (Microsoft, W3C); UDDI: Universal Discovery, Description, and Integration (Consortia, includes HP); WSDL: Web Services Description Language (IBM, Microsoft, W3C).

M. Feridun et al. (Eds.): DSOM 2002, LNCS 2506, pp. 28–41, 2002.

current users are 10,000 on the server side). Also as multiple web services will inter-operate, it may be necessary to guarantee SLA between two constructs not directly related to each other (for e.g. a guarantee may be given on the period between Service A receives an order message from a client and the time when a ReceivedGoods message is sent by the client to a shipment company that Service A uses to deliver goods physically to the client). In such cases unless the measurements are obtained from multiple locations and aggregated, SLA monitoring cannot be done. In this paper we propose an automated and distributed SLA monitoring engine that enables the above functionality.

2 Web Service Infrastructure

A web service infrastructure would comprise of large number of business processes. These business processes will usually comprise of set of activities. Each activity will be handled by either humans (as is in work-flow management systems), automated systems (based on legacy systems or state of the art application servers) or some times will be outsourced to external e-businesses. In Figure 1, a simple example of a web service infrastructure is shown. This particular business is set up by PCMaker.com that receives orders from companies/humans interested in buying PCs. It has internal business processes like user authentication, PC manufacturing, preparation of invoices etc. These business processes are defined in terms of WSFL/XLANG (or BPEL4WS). These activities are exposed through set of operations in their WSDL.

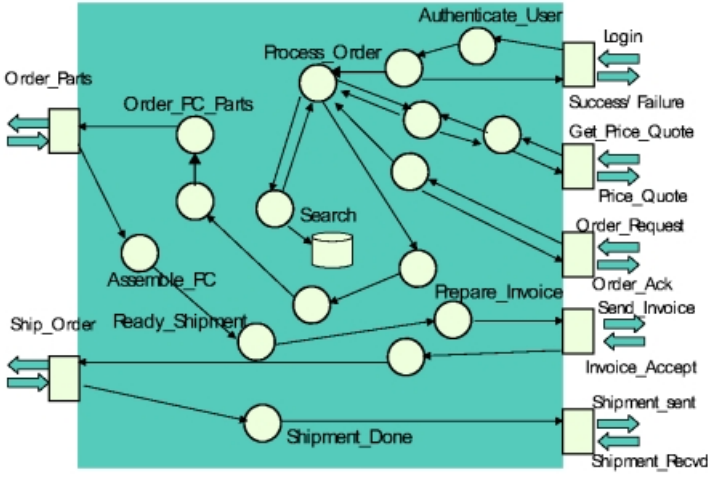

Fig. 1. A typical web service and business process infrastructure

3 Web Service SLA

Protocols like ebXML/BTP enable web service to web service interactions to be captured through a set of well-defined processes. These processes are distinct from

the internal business processes as mentioned above. Parts of these processes could be sub-processes defined by standards such as say RosettaNet PIPs. This enables the fact that web services can undertake business by executing an orchestration of business transactions amongst themselves. This involves definition of a combined process between the two partners, which in turn is bi-sected according to the roles undertaken by the partners (namely customer, provider). Each party executes the process belonging to their role. These processes involve a particular sequence of invocation of each other's operations through message exchanges between them. The operation and message exchange interfaces are already captured in WSDL descriptions as explained. These processes also interface with internal business processes that are defined in process definition languages like WSFL or XLANG.

While, WSDL introduces concepts such as messages, operations, ports, and end points – which are useful for describing the operations of any web service, WSFL introduces the notion of activities and process flows. 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, WSFL, XLANG or BTP/ebXML. Here are some examples that show how such association can be done.

- Response time of a web service operation.
- Average response time between two set of messages
- Response time of a process flow.
- Average response time of a set of process flows of a particular type
- Security of an operation.
- Number of times an activity is executed in a flow.
- Cost of executing an operation.
- Availability of an end point.
- Recoverability of an end-point

The concept of service level agreements and guarantees is missing as yet in the world of web services and business transactions. We introduce the concept of SLAs/contracts amongst web services in this article. An SLA has a set of Service Level Objectives (SLOs) as specified.

A typical SLA between a company manufacturing PCs (say PCMaker.com) and a company buying PCs (PCBuyer.com) for a period of 6 months will be as follows:

SLO1: PCMaker's e-procurement system *will be available to PCBuyer1, Monday to Friday from 9AM-5PM, 99.9 % of the time*
SLO2: PCMaker shall deliver the ordered goods *on an **average** within 10 days of the receipt of a purchase order*
SLO3: PCMaker shall invoice PCBuyer for any goods ordered *within 6 hours*
SLO4: Payment of goods by PCBuyer shall be done ***always** within forty-five days of the receipt of invoice from PCMaker.*

Each SLO has a functional part (that refers to a system, endpoint, a process, or a set of processes…) and a guarantee part (italicized) applied on the functional part. The guarantee is on a system, a particular instance of a construct (process/opera-

tion/message..) or on a set of such constructs. SLA monitoring involves monitoring whether these guarantees on the functional parts are being met.

In order to automate SLA monitoring, we propose a specification language that enables definition of precise and flexible SLAs, and is described in detail in section 3.1. Section 3.2 describes the instrumentation aspects that enable correlation of web service and business process data. The Web Service Management Network Agent (WSMN Agent) that automates and distributes the SLA monitoring process is described in section 3.3. In section 4, the implementation details of the WSMN Agent are described.

3.1 SLA Specification

The first enabler 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.

Examples of the lack of flexibility and precision in existing SLA formalizations are discussed in [1]. Detailed explanation of how we have addressed flexibility and precision in coming up with SLA formalization are also presented in [1]. Below is a summary of the formalization.

An SLA is specified over a set of data that is measurable. An SLA typically has a date constraint (start date, end date, nextevaldate) and a set of Service Level Objectives (SLOs). 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 determines where the measurements are taken (provider, consumer side). A clause evaluation is triggered 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 of an SLO. Once the evalWhen trigger arrives, a set of samples of measuredItem are obtained applying a sampling function. The *evalOn* component determines how this sample is computed. The sample set is a constrained set of measured data that is constrained by 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. A function (*evalFunc*) is thereafter applied on the sample set so obtained. 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 evalAction specifies what action to perform after the evaluation is done. The following grammar shows a portion of this formalization.

As an example, a 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. The complete set of examples of how complex SLAs can be represented in it are presented in [1].

```
SLA  = dateconstraint SLO*

Dateconstraint =
startdate enddate nextevaldate

SLO = daytimeconstraint clause*

Daytimeconstraint = Day* time

Clause =
MeasuredItem evalWhen evalOn evalFunc evalAction

MeasuredItem = Item*

Item  =
measuredAt constructType constructRef
```

3.2 Instrumentation

In order to ensure that guaranteed SLAs can be evaluated and their compliance measured, it is necessary that raw measurement data be collected about the managed system. This managed data is obtained through instrumentation of processes, activities that are executed, and messages that go in and out of the e-business infrastructure.

Instrumenting the web service. It is necessary to observe the message exchanges among web services in order to collect information about the interactions with business partners. An acceptable solution should not impose any modifications or limitations on existing web services. Since SOAP is rapidly becoming the preferred standard for web service interactions, we assume SOAP messages are used among web services in order to submit request and response messages. We have implemented a small proxy component tries to capture incoming and outgoing messages, and records data about the message exchanges, then forwards the captured messages to the actual recipients. We have considered various alternatives for easily attaching a proxy component to existing web services in order to listen to incoming and outgoing messages: port sniffing, server-side filters (Microsoft's ISAPI, or Netscape's NSAPI), API provided by web services themselves, and modification of SOAP toolkit. Since SOAP is widely accepted for message exchange, port sniffing and server-side filters are not suitable, because the message contents are encrypted by SOAP toolkit. Most web services do not provide an API for controlling or querying about their activities due to security issues or simply because the web service developers did not feel any need for such interfaces. Consequently, we have chosen to keep track of message exchanges among web services by modifying SOAP toolkit by overwriting it. However the monitoring may not be restricted to SAP and additional modules can be defined say as part of the Apache AXIS architecture to undertake message payload monitoring.

In order to correlate individual message exchanges with each other, we use the notion of Global Flow (GF) as described within our assumptions above. The GUID is used for keeping track of a GF. Every time our proxy component catches a message that is exchanged between web services, it first checks whether a GUID exists. If a GUID does not exist in the message, the proxy inserts a GUID into SOAP header of the message. All web services and other software components propagate the GUID in their communications. Consequently, our proxy components that are attached to SOAP toolkits at business partner sites can easily figure out which SOAP message is sent in the context of which previous messages.

Instrumentation of business process. Since business processes at the back-end automate activities of web services, it is necessary to collect data from those software components in order to gather detailed information about internal activities of a business, and correlate those internal activities with external message exchanges. As we indicated among our assumptions, most business process management systems log data about internal business process executions into a raw log file or database. For example, HP Process Manager (HPPM) logs execution data into a raw file, which is then uploaded into database tables by a dedicated process. A proxy component can be configured in order to read logged data from proper database tables. This component can also correlate the message exchanges with internal process executions using the GUID that is passed through all web services and their back-end software components.

3.3 SLA Monitoring

As minimal human intervention is desirable in web services it is necessary to create monitoring engine that can take care of a variety of specifications and monitor the necessary management data. We believe that the SLA formalizations described above are precise enough to be able to create or customize an SLA monitoring engine on the fly. To simplify the discussion, we will describe the details of the engine as if it manages a single SLA between two services. Such an engine has then two components – one on the service provider side and one on the service consumer side. Extending our notion to a large number of SLAs requires that the engine keep track of the state of multiple SLAs simultaneously, and be able to relate each measurement to one or more affected SLAs. Extending our notion of two services to a large number of interacting services requires the engine's components to take the dual role of acting as both "service providers" in some SLAs and as "service consumers" in some SLAs.

The instance data so collected has to be modeled in the high performance database and a data warehouse so that service level agreements can be monitored on top of the modeled data. The high performance database is updated for every transaction instance data that is received. The data warehouse is updated at regular intervals of time for keeping the data for a longer period of time.

SLM Engine. The SLM Process Controller executes the management processes for the SLM engine. These management processes are distinct from the business processes that are executed in the web services infrastructure as discussed in section

2. These management process flows are created and managed for a variety of purpose. These flows are defined in WSFL and are exposed to other WSMN agents through WSDL specification of their own. These WSMN agents thus can initiate management related conversation with each other. The WSMN agent process controller executes the *SLA monitoring process flow* for undertaking SLA evaluation and reporting.

As the specification typically has startdate, enddate, daytimeconstraint, evalWhen, evalOn and evalFunc components to it, each of these constitutes a generic component that can be used by our SLA Management engine. In addition, we have also identified the most common variants of these generic components, which can be readily parameterized by the engine for a large number of possible combinations of SLAs. Using a new, evalWhen, evalOn, or evalFunc component in an SLA requires an administrator to first develop such a component within the framework of our engine and then to add it to the engine.

The *model generator* receives the WSDL/WSFL specifications and creates a model of the web service in the *model repository*. All the measurements collected from the web service (e.g., ongoing conversations, performance measurements, etc) are attached to this model. The instrumentation in the web service is responsible for collecting these measurements and passing them on to the *management handler* to be stored in the model repository. If the measurements are collected on the client side (as determined by the measuredAt components of the items in SLA clauses), then the *communicator* is responsible for receiving the measurements and storing them into the repository. SLM Engine process controller receives the SLA executes a monitoring process flow (as explained in subsequent section) and accordingly informs the SLA customizer which in turn customizes the alarms at the Alarm Manager (depending on the evalWhen and dateconstraint components). The Alarm Manager comprises of the SLO Validity Period Monitor, and triggers (time based and event based). The SLA customizer also creates an SLO object in the SLA/contract repository and registers it as the call back handler of the alarms. The SLO object maintains the state of the SLO (valid, active, invalid). If a registered alarm for start-date of an SLO arrives the state of the SLO is changed from init to valid. The SLO is invalidated when the end-date trigger arrives. In between as the evalWhen alarms are triggered (because of a time or an event happening) the SLO evaluator evaluates the SLO. The *SLO evaluator* obtains the required management information (based on evalOn, daytime Constraint and the evalFunc constituent of the specification) from the high performance database in memory. The SLO evaluator determines compliance/violations. The *SLA violation engine* maintains the record for violations, their timestamps, the levels of violation, and the clauses that are violated (both in memory and in log files). The management console can be used for looking and visual analysis of the current SLAs, SLOs, and their violation records. The violation records will also be used for triggering SLA *assurance* processes.

Management Information Modeling. The *model generator* component receives the WSDL/WSFL specifications and creates a model of the web service in the *model repository*. The instrumentation dictionary contains information about the instrumentation and thereby the metrics that are available for various components of the web service. It can then combine the service model with the metrics available at

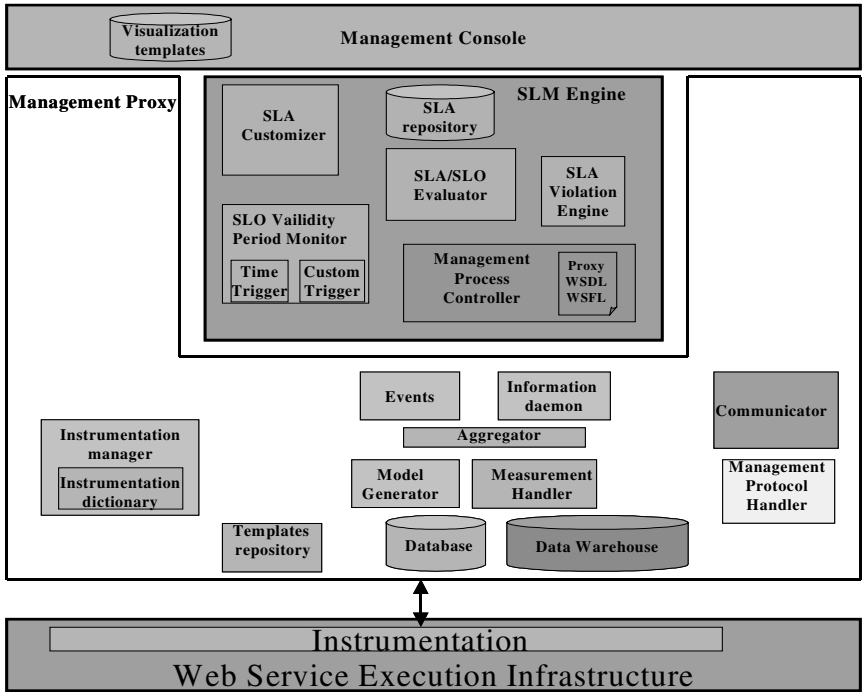

Fig. 2. The WSMN Agent

each of the web service model component. This combined model is created in the repository. Subsequently when the actual measured data are stored by the measurement handler, the management data is stored according to the combined model[11].

All the measurements collected from the web service (e.g., ongoing interactions, performance measurements, etc) are attached to this combined model. The instrumentation in the web service is responsible for collecting these measurements and passing them on to the *management information handler* to be stored in the model repository. If the measurements are collected on the client side (since the measuredAt component says so in an SLA), then the *communicator* is responsible for receiving the measurements and storing them into the repository.

In the managed object model used by the SLM engine, the basic web service and business process constructs are viewed as derived from a base class. We term the base class as the *managed object*. Every managed object has a set of *attributes*. An attribute is defined in the *attribute definition*. The attribute definition comprises of the *identifier*, *name*, *datatype*, *calculable*, *units* of the attribute. The identifier uniquely refers to an attribute definition while the name provides a label for it. The managed object has the *base attributes* of id, context, status, userId. All the other constructs, like operations, activity, processFlow, globalFlow, .. etc extend managed object. All

the constructs thus have id, attribute, context, status, userId and other attributes that are specific to them.

The basic managed object model is extensible. At each of the constructs new attributes conforming to the data types mentioned above can be defined through new attribute definitions. This will allow for management systems that are capable of collecting additional information about the constructs. Also derived attributes can be defined that manipulate the base attributes.

In addition, metrics can be defined on top of the managed object model as defined in the previous section. A management system may create a metric object for modeling a (set of) managed object(s). The ITU-T model is quite applicable in our case of managed systems modeled through web service and business process abstractions [5]. The ITU-T metric object model for example provides for definition of mean monitor, moving average mean monitor. Mean and variance monitor, mean and percentile, mean and min max monitor.

The management data is thus collected and modeled in the databases in the SLM engines on both sides. If all the measuredItems are local then the SLA can be evaluated on the local data. However, if the measuredItems refer to attributes on web services on either side the data so collected needs to be exchanged between the SLM engines.

The data is continuously measured, modeled and stored in the database (and consequently in the data warehouse at regular intervals of time) as shown in Figure 2. The WSMN Agent Process Controller receives the SLA specification either by snooping on the web service to web service communication or directly through the management console. Once the SLA is received the Service Level Monitoring process flow is executed on both the provider-side and the customer-side.

Service Level Monitoring Process Flow. The process consists of the following steps:

(a) The process (SLM process) is initiated as soon as an SLA is received as input.
(b) Decide where the measurements are to be carried out. This is marked on every measured item in the SLA using *measuredAt*.
(c) Decide where the evaluation of the SLA is to be done. The SLA evaluation is carried out at the customer side, if the SLA has items that are all measured at the customer side. Similarly, if all the measured items are measured at the provider side, the SLA evaluation is carried out at the provider side. At the end of evaluation the SLM engines exchange violation report through SLA Violation Report Exchange protocol.
(d) If however, some of the items are measured at the customer side, and some of them are measured at the provider side, then the evaluation is carried out at the provider side. This last case, however requires that the customer-side measurements are transferred to the provider-side.
(e) If some of the measurements have to be transferred from customer side to provider side, initiate *measurement exchange protocol*. The measurement exchange protocol takes care of transferring measurements at the right fre-

quency and right level of aggregation. This is described in detail in the next section.

(f) If the engine is responsible for the SLA evaluation, it sends the SLA to its SLA customizer that in turn creates the SLO, stores it in the SLA repository, customizes the alarms in the Alarm Manager and registers the SLO object as the call back handler for them. Once configured, the components of the SLA monitoring engine described above automatically trigger the evaluation of the SLA.

Measurement Exchange Protocol (MEP). When the evaluation of an SLA depends on measurements from both the customer-side and provider-side, a measurement protocol is needed for transferring the measurements from the former to the latter. Such a protocol should be designed with the following objectives in mind: (a) minimize the amount of data that is transmitted between the two sides, and (b) transfer the data in time for the evaluation of SLA to take place when triggered.

To fulfill these two objectives, the SLA monitoring engines on both sides should agree on (a) what measurements need to be transferred and at what level of aggregation, and (b) how frequently they should be transferred. The type and level of aggregation of the measurements depends on both evalFunc and measuredAt. To specify the level of aggregation, we use typical sampling functions such as count (t), totaled, averaged, movingAvg(lastN), minN, maxN, threshold. In the case when the sampling function cannot be determined from the evalFunc, we ship all the measurements from the customer-side to the provider-side. The reporting frequency depends on evalWhen.

The measurement protocol handles both the agreement on level of aggregation and frequency, as well as the transfer of agreed measurements from customer-side to provider-side. There are in essence 5 different types of messages that form the protocol.

- Init: sent by the consumer to the provider for clauses whose measurement data need to be exchanged. The init message carries possible choices of sampling function, interval, duration and reporting interval details that the consumer supports.
- Request: The provider decides the exact measurement specification (sampling function, sampling params and reporting params) that it chooses and specifies it in its request message.
- Agreement: The consumer sends this message if it agrees to the request
- Start: message from provider to commence the reporting.
- Report: actual measurement report messages
- Close: message to terminate the reporting.

Violation Engine. Once the SLOs are invoked by the Alarm Manager, the SLO evaluator evaluates the function (evalFunc) of the SLO. The query that is created uses daytime constraint, evalOn and evalFunc components of the SLA specification. The results of these evaluations are compared against thresholds and the details of the evaluation are maintained as a Violation Record in the violation engine. It is also appended to the log File. Business managers can use the violation records for

controlling the web service and business process infrastructure for contract assurance purpose and for visual analysis.

4 Implementation

A WSMN Agent was implemented (in Java). The Agent uses Apache SOAP toolkit to exchange messages with each other. They execute management processes on HP Process Manager. A sample web services scenario as described earlier and shown in Figure 3 was implemented and the messages, business processes involved were instrumented. For the web service scenario the actual business processes were also created on HPPM. HPPM provides a Java API to control process executions by other software components. A proxy component uses this Java API to feed in the GUID into HPPM process instances and retrieve it when necessary. The web services also use Apache SOAP toolkit for exchanging messages with each other. The SOAP toolkit was modified to collect the message correlation and instrumentation data. The measured data was stored and modeled in mySql database and Oracl9i data warehouse.

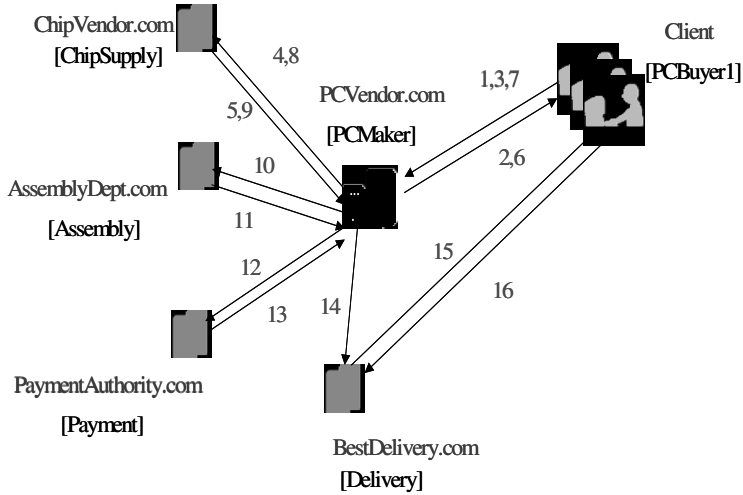

Fig. 3. The web services scenario that was implemented

The example scenario as shown in Figure 3 was implemented. The implemented scenario has two SLAs between PCMaker.com and its customers namely PCBuyer1.com. The SLAs were namely SLA1 and SLA2. Each SLA has a single Service Level Objective. The first SLA is with PCBuyer1.com. It guarantees that between the dates of 0.2/15/02 and 07/15/02 all the invoice processes from 9-5 and on weekdays will be undertaken in 6 hours. The evaluation will be done every day at 6 PM.

#	MSG_TYPE	SENDER	RECEIVER
1	SubmitLoginmsg	PCBuyer1	PCMaker
2	ConfirmLoginmsg	PCMaker	PCBuyer1
3	SubmitQuoteRequestmsg	PCBuyer1	PCMaker
4	RequestChipQuotemsg	PCMaker	ChipSupply
5	SendChipQuotemsg	ChipSupply	PCMaker
6	SendQuotemsg	PCMaker	PCBuyer1
7	SubmitPORequestmsg	PCBuyer1	PCMaker
8	SendChipPOmsg	PCMaker	ChipSupply
9	RespondChipPOmsg	ChipSupply	PCMaker
10	SendAssemblyPOmsg	PCMaker	Assembly
11	RespondAssemblyPOmsg	Assembly	PCMaker
12	SendPaymentPOmsg	PCMaker	Payment
13	RespondPaymentPOmsg	Payment	PCMaker
14	SendDeliveryPOmsg	PCMaker	Delivery
15	SendDeliveryNotificationmsg	Delivery	PCBuyer1
16	sendReceiptNotificationmsg	PCBuyer1	Delivery

<sla>
<slaId>1</slaId>
<partnerName>PcBuyer1.com</partnerName>
<startDate>Fri Feb 15 00:00:00 PST 2002</startDate>
<endDate>Mon Jul 15 00:00:00 PDT 2002</endDate>
<slo>
<sloId>1</sloId>
<dayTimeConstraint>Mon-Fri: 9-17</dayTimeConstraint>
<measuredItem>
<item>
<constructType>process</constructType>
<constructRef>PcMaker.com/Invoice</constructRef>
<measuredAt>PcMaker.com</measuredAt>
</item>
</measuredItem>
<evalWhen>6PM</evalWhen>
<evalOn>all</evalOn>
<evalFunc name =”averageResponseTime” operatior =”LT” Threshold =”6” unit =”hours”></evalFunc>
</slo>
</sla>

SLA2 has an SLO that is based on two messages from two different end-points. This particular SLO has two items. The management proxy at PCBuyer1.com and PCMaker.com utilize the measurement exchange protocol to agree on sending measurements for sendReceiptNotificationmsg everyday just before 6 PM and keep sending the reports from startDate to endDate.

```
<sla>
<slaId>2</slaId>
<startDate>Fri Feb 15 00:00:00 PST 2002</startDate>
<endDate>Mon Jul 15 00:00:00 PDT 2002</endDate>
<slo>
<sloId>1</sloId>
<dayTimeConstraint>Mon-Fri: 9-17</dayTimeConstraint>
<measuredItem>
<item>
<constructType>message</constructType>
<constructRef>PcMaker.com/submitPORequestmsg</constructRef>
<measuredAt>PcMaker.com</measuredAt>
</item>
<item>
<constructType>message</constructType>
<constructRef>PcBuyer1.com/sendReceiptNotificationmsg</constructRef>
<measuredAt>PcBuyer1.com</measuredAt>
</item>
</measuredItem>
<evalWhen>6PM</evalWhen>
<evalOn>all</evalOn>
<evalFunc name ="averageResponseTime"  operatior ="LT" Threshold ="2" unit
="days"></evalFunc>
</slo>
</sla>
```

The WSMN Agent corresponding to PCMaker.com is loaded with the SLAs as mentioned above. These SLAs are passed as input to the WSMN Agent Process controller that in turn determine that these SLAs are either all locally measured as in SLA1 or need measurements (necessitating MEP) from other sites as in SLA2. They are then passed to the SLA customizer. The SLA customizer creates the SLO objects and customizes the Alarm Managers. The evaluations are done as these alarms arrive.

5 Related Work

SLAs as means of defining Quality of Services have been researched earlier [6,7,8]. In [2] SLA management in a federated environment has been researched. The SLA management engine needs a service model that determines the services offered in the domain as well as dependencies between the service components. It also needs the measurements available from them at each level to be specified in the model. A systems dictionary is required that specifies which plugins to use to gather which information. This requires human intervention for service model and system dictionary definition. As the contracts defined in contract definition language are mapped to measurements from the systems dictionary and the process in not totally automated the specification may lead to ambiguities.

Details of schema for specification of SLAs and a corresponding conceptual architecture are presented in [10]. Our paper also presents an SLA specification language

[1] and an automated and customizable SLA monitoring engine based on the specification language. In order to achieve its objective, it describes how to measure and correlate messages and processes in their overall global context, as captured by the concept of global flow id, and to model them logically so that SLA monitoring can be facilitated. This model as presented in section 3 is distinct from the SLA specification schema. The SLM engine also is capable of dealing with SLAs based on measurements from multiple sites by using MEP which is not the case in the above mentioned paper.

Inter-domain communication has been handled in telecommunication networks [3][4]. However, unlike the Internet their networks are regulated and typically designed to offer a single type of service.

6 Conclusion

Service Level Agreements are difficult to specify in a clear and unambiguous manner. It is equally difficult to automate the monitoring of these SLAs. In addition, most of the SLAs deal with provider side guarantees and neglect client side measurements. In this article, we have proposed an automated and distributed SLA monitoring engine that monitors an SLA specified in our language.

References

1. Sahai A, Durante A, Machiraju V. : Towards Automated SLA Management. For Web Services. HPL-2001-310. (2001)
2. Bhoj P., Singhal S., Chutani S.: SLA Management in a federated Environment. HPL-1998-203. (1998)
3. Lewis et al. Experiences in Integrated Multi-Domain Management. IFIP/IEEE International Conference on Management of Multi-Media Networks and Services, Montreal, Canada, (1997)
4. Hall J (editor). Management of Telecommunication systems and Services: Modelling and Implementing TMN based Multi-Domain Management, Lecture Notes in Computer Science 1116, Springer-Verlag, ISBN 3-540-61578-4, (1996)
5. Telecommunication Management Network (TMN) at ITU-T.: Formerly CCITT. http://www.itu.int
6. Long T. P., Jong W. B., Woon H.J. : 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. Tele Management Forum SLA Management Handbook, GB917, public evaluation version 1.5 , June. (2001).
9. Sahai A, Ouyang J, Machiraju V, Wurster K. : Message Tracking in SOAP-Based Web Services. In the proceedings of NOMS 2002 33-51., Italy (2002)
10. Keller A et al. : Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture. In the proceedings of NOMS 2002. 513-528 , Italy (2002).
11. Sahai A, Machiraju V, Casati F. An Adaptive and Extensible Model for Management of Web Services and Business Processes. Openview University Association (2002).