Automating WS-Agreement Monitoring

Ajaya Kumar Tripathy, Department of CSE, Silicon Institute of Technology, Bhubaneswar, India
Manas Ranjan Patra, Department of Computer Science, Berhampur University, Berhampur, India

An SLA (Service Level Agreement) is an agreement between the web service provider and web service consumer that specifies a mutually agreed level of service quality, functional properties of a web service and business assumptions. WS-Agreement is a standard form of SLA for specifying agreement between service provider and service consumer. In WS-Agreement specification, the level of service quality, functional properties of a web service and business assumptions are termed as guarantee terms. Service provider and consumer need to be assured of the guarantee terms specified in the SLA at the time of service provisioning which can be done by SLA monitoring. A non-intrusive and service composition platform independent SLA monitoring is a real challenge. This paper proposes a protocol for specifying SLA guarantee terms using XML schema. Further, a non-intrusive and SBS composition platform independent monitoring framework has been proposed using a first order temporal logic based language called MSL, in order to ensure the WS-Agreement guarantee terms and monitor both functional and non-functional properties at run-time.

Categories and Subject Descriptors: H.3.5 [Web-based services]: Web-based services
General Terms: Verification, Language, Design
Additional Key Words and Phrases: Web Services, Service Based System, Run-Time Monitoring, Service Level Agreement

1. INTRODUCTION

A web service is an application that exports a description of its functionality and makes it available using standard network technologies. These functionality can be accessed through standard XML messages over a network.

Research on Web Services spans over many interesting issues covering all the phases of the service life-cycle: e.g. service description, service discovery, selection and invocation, service composition, service advertisement, service negotiation, service security and reliability. In this article, we focus on yet another very interesting research topic: SLA specification and monitoring.

The ability to set up and monitor SLAs has been increasingly recognized as one of the essential preconditions for the deployment of web-services [Ludwig et al. 2003]. SLAs are set through collaboration between service consumers and service producers in order to specify the terms under which a service is to be offered and the quality aspects that it should satisfy under these terms. The ability to monitor the compliance of a set of agreed upon services at runtime with respect to a service level agreement is crucial both from the point of view of the service consumer and the service producer.

For service consumers, monitoring service level agreements is necessary in order to check if the terms of an agreement are satisfied, identify the consequences in the event of violation of certain terms in the agreement, and claim penalty for any violations. For service providers, monitoring of the provisions of a service against the terms specified in an agreement is necessary in order to gather evidence regarding the agreed upon service provisions in case of any dispute with a service consumer while availing a service. Besides this monitoring is necessary in order to identify problems with the delivery of certain services and take appropriate action before an agreement is violated.

Service providers can make use of the SLA technology to advertise and offer their service capabilities while consumers can formalize their service level objectives through SLAs. It is in the interest of both the parties to create and operate SLAs with minimum human interaction on one hand and to agree upon legally binding electronic contracts and monitor the contracts on the other hand [Hasselmeyer et al. 2007]. Several SLA monitoring frameworks have been proposed to cope with the SLA guarantee term specification and monitoring(see e.g. [Chau
et al. 2008; Mahbub and Spanoudakis 2007; Barbon et al. 2006; Keller and Ludwig 2003; Sahai et al. 2002; Tripathy and Patra 2010a; Michlmayr et al. 2010). A novel solution to the problem of SLA specification and automatic monitoring of SLA guarantee terms has been proposed in this paper.

This paper proposes an extension of WS-Agreement for SLA specification. It also describes a framework that we have developed to support the monitoring of guarantee terms (functional, quality of service requirements and business assumptions) which are specified as part of service level agreements. This framework can monitor the provision of services to web-service based systems (SBS) i.e, a system that uses one or more external web-services which are coordinated by a composition process. The proposed framework is event based and non-intrusive in the sense that events are collected during the operation of an SBS system without interfering in the composition process. The monitoring mechanism is also independent of service composition platform.

The proposed framework uses an extension of WS-Agreement [Andrieux et al. 2004] which has been have been defined to monitor SLAs. This extension supports description of the functional and quality requirements as well as business assumptions for the services (i.e., the guarantee terms in the terminology of WS-Agreement). A language based on XML schema (XMSL) has been developed to specify service guarantee terms as events. In turn XMSL is based on a temporal logic based language called MSL [Tripathy and Patra 2010a]. Specification of service guarantee terms in XMSL can be developed independent of WS-Agreement. The events which are used in the specification of the service guarantee terms in an agreement can only be observed from the SBS environment (i.e, business layer, service layer and infrastructure layer of the SBS) during the execution of the composition process. The choice of MSL as the language for specifying service guarantee terms is due to its expressiveness as a formal language which allows specification of temporal constraints and the ability to monitor an agreement using well defined reasoning process in the form of inference rules written in first-order logic.

Further, our monitoring framework has been designed with the objective to support non-intrusive, service composition platform independent monitoring of service level agreements as well as SBS functional and non-functional properties. The framework that we discuss in this paper was originally developed to support the monitoring of functional and non-functional property requirements of web-service based systems and the main formal characteristics of the original form of the framework can be found in [Tripathy and Patra 2010a]. In this paper our focus is to show the usability of this framework in monitoring WS-Agreement and to introduce extensions to handle additional requirements of SBS.

The rest of the paper is structured as follows. Section 2, describes the state of the art in Web Service Technology and existing research approaches to monitoring of SLA for SBS. An example scenario is presented in Section 3. Section 4 introduces WS-Agreement. Section 5 gives a complete description of the SLA monitoring framework. In which, Section 5.2.2 describes the protocol designed for formal specification of SLA guarantee terms and Section 5.3 describes the run-time monitoring environment of the monitoring framework. Finally we conclude our work with a mention of our future research direction in Section 6.

2. STATE OF THE ART

Web Service Technologies

Web Services are platform-independent, self-contained, self-describing, modular components that can be published, located and invoked over the Web. In order to achieve interoperability in such an heterogeneous framework, standards are of vital importance [Leymann 2003]. A whole stack of different standards has already been proposed with the aim of supporting the description, discovery, and interoperability of distributed, heterogeneous applications as services.
The functional description of a Web Service is provided by the Web Services Description Language (WSDL) [Christensen et al. 2001]. WSDL describes a set of operations it offers, in-coming and out-going messages, and data types used by the Web Service (defined in terms of XML Schemas). Concrete protocol bindings and physical address port specifications complete a service description, providing a mechanism to locate a Web Service. WSDL defines what a Web Service does, not how it does; it characterizes the service only in terms of its interface, without providing any behavioral description. Such dynamic aspects are crucial for a complete understanding of a web service so that it can be recognized and used by autonomous applications.

**SLA Monitoring**

The necessity of specifying and monitoring different properties of composition assumptions as well as functional and non-functional requirements of SBS is widely recognized by industry and academia.

Jin et al. [Jin et al. 2002] propose a SLA modeling to understand the impact of the SLAs on the productivity of the customers. This model captures composition relationships between providers as well as the SLA between them. This model analyzes SLA based on simulation of the model and sensitivity analysis.

Lemana et al. [Lamanna et al. 2003] have proposed a SBS monitoring approach with the introduction of the language SLAng. This language is an extension of the existing business process languages. In this language properties are defined as a list of Quality of Service (QoS) parameters. At the implementation stage QoS parameters are assigned to the target business process, this leads to an intrusive approach. The target servers are required to support these QoS parameters. This approach becomes less extensible and flexible. The approach described in [Sahai et al. 2002] creates monitoring agents to monitor the business process. These agents monitor the business process by gathering the network usage information. Another process evaluates the properties for any change in the process. This approach requires the business process to update constantly in order to adopt to new property requirements.

Barsi et al [Baresi et al. 2004] have proposed an approach for monitoring dynamic service composition with respect to guarantee terms expressed via assertions on services. This approach assumes composition process specified in BPEL. A guarantee term is verified by a call to an external service and the execution of the composition process waits until the monitor returns the result of the check. The composition process may continue or abort with an exception notification on whether the guarantee term is violated. The monitoring that it performs may effect the performance of the monitored system. This approach is intrusive to the normal operation of an SBS.

Another monitoring approach is presented by Baresi et al. [Baresi and Guinea 2008]. This approach monitors both functional correctness of BPEL orchestration and quality of service agreements set between the service provider and the service consumer. They provide a language called WSCoL (Web Service Constraint Language) [Baresi and Guinea 2005] which allows designers to specify constraints on BPEL orchestration. Appropriate external services called Monitoring managers are responsible for analyzing WSCoL constraints. The business logic is unaffected by monitor specification. Therefore, we can say the approach is non-intrusive at the specification time. But to allow the process to interact with the external monitors, additional BPEL code is added to the process at deployment time, this leads to an intrusive approach.

Lazovik et al. [Lazovik et al. 2004] presents a framework in which service requests are presented in a high-level language called XSRL (XML Service Request Language). The framework monitors the execution of the request services. Designers can define three kinds of properties: (1) goals that must be true before transiting to the next state (2) goals that must be true for the entire process execution, and (3) goals that must be true for the process execution and evolution sequence. The framework loops between execution and planning.
The latter activity is achieved by taking into account context and properties specified for the state-transition system. This makes it possible to discover whether a property has been violated by the previous execution.

Barbon et al. [Barbon et al. 2006] present a monitoring approach extending the open-source Active BPEL engine. This approach defines an executable monitoring language RTML (Run-Time Monitor specification language) to specify properties of SBS to monitor, which is based on events and combines them exploiting past-time temporal logics and statistical functionalities. Monitors run parallel to BPEL (Business Process Execution Language for Web Services) [Andrews et al. 2003] process as independent software modules that verify the guarantee terms by intercepting the input or output messages that are received or sent by the process. The framework supports automatic generation and deployment of monitors using guarantee terms specified in RTML. This is a nice approach but works only at service level for the BPEL processes.

Mahbub et al. [Mahbub and Spanoudakis 2004; Spanoudakis and Mahbub 2006; Mahbub and Spanoudakis 2007] present an approach extending the WS-Agreement [Andrieux et al. 2004]. This approach supports monitoring of quality and functional properties. It introduces a new language to specify service guarantee terms in terms of: (1) events signifying invocation of operations of a service by the composition process of an SBS system and return from these executions, (2) events signifying calls of operation of the composition process of an SBS system by external services and return from those executions, (3) the effect that events of either of the above kind have on the state of the SBS system or the service that it deploys. This language has been defined by a separate XML schema and is called EC-Assertion, which is based on Event Calculus (EC) [Shanahan 1999] which is a first order temporal logic language. It is a nice approach but limited to only service level BPEL processes.

Leitner et al. [Leitner et al. 2010; 2010; Wetzstein et al. 2009] propose an approach for monitoring SLA violation at run-time, which uses measured and estimated facts (instance data of the composition or QOS of used services) as input for a prediction model. The prediction model is based on machine learning regression techniques and trained using historical process instances.

Tripathy et al. [Tripathy and Patra 2010b; 2010a] present a non intrusive and SBS composition platform independent monitoring approach. This approach defines an executable monitoring language called MSL (Monitor specification language) to specify properties of SBS to monitor. Monitors verify the guarantee terms by intercepting the input or output messages that are received or sent by the process and interested events from different layer of SBS. The framework supports automatic generation and deployment of monitors using guarantee terms specified in MSL. This is a nice approach but only can monitor instance level properties.

3. AN EXAMPLE SCENARIO

Here, we present an example scenario called Intra City Motorbike Rental Service (MRS) (see Figure 1), which is referred in the rest of the article to explain different concepts developed in the monitoring of an SBS. Essentially an MRS acts as a broker enabling its customers to avail bikes on rent which are normally provided by different bike rental companies. In order to use a bike in a particular area of a city a customer can book online and pick the bike from a bike parking place nearer to the requested location. The MRS can be viewed as a service based system which involves a service composition process that interacts with the following web services:

— Motorbike information services (MIS) which are provided by different motorbike rental companies. It maintains a registry of bikes, checks for the availability of bike when requested and issues an available bike identification number, if a requested bike is available.
— Identification Sensor Services (ISS) which are provided by the motorbike parks to sense motorbike identification as well as customer identification as they drive in or out of motorbike park area and accordingly informs the MRS.
— Motorbike Rent Payment Services (RPS) which are provided by different banks at different motorbike parks to receive the rent amount from MRS clients after use of motorbikes.
— Client Services (CS) that provide MRS with a user-friendly front-end for customers to interact.

In this explanatory example scenario of MRS messages flow as follows: CS activates the MRS with a motorbike rent request message mentioning the location at which the motorbike is required and his/her personal identification i.e, rentRequest(UID, location). Then MRS checks the availability of motorbike at the requested location by invoking MIS (by sending isAvailable(location) message). MIS responds with the availability status: saying no by sending notAvailable message / saying yes by sending an available message with a available motorbike identification i.e, available(mBikeID). If MRS gets an available response then acknowledge CS by sending a price and condition offer message i.e, offer(price, condition) otherwise acknowledge CS as motorbike is not available by sending notAvailable message. If CS accepts the offer of MRS, then it sends startPayment(accInfo) message to MRS, otherwise it rejects the offer by sending offerReject message. Then MRS sends a new offer to CS. In case of offer acceptance, MRS invokes RPS to get the payment by sending makePayment(accInfo, cost). RPS acknowledges MRS about payment success/fail by sending paymentSucc/paymentFail. After receiving paymentSucc, MRS acknowledges MIS that the bike is booked (by sending bikeBooked(mBikeID)). Then the MIS updates its database accordingly and sends a bike identification number with unlock key to CS (by sending bikeKey(mBikeID, unlockKey) message). When bike enter/exit to/from a bike park, ISS informs MRS by sending bikeIn(mBikeID, UID)/bikeOut(mBikeID, UID), subsequently inform MIS to update its database by sending a message bikeIn(mBikeID, UID)/bikeOut(mBikeID, UID). Figure 2 depicts the message flow of MRS as explained above.
Despite the simplicity of the domain, because of business requirements, the MRS provider and/or consumer may want to include some of the following properties of the SBS as guarantee terms in the SLA. Additionally, the service provider would like to monitor some of the following properties in order to provide better service.

*The class-I of guarantee terms are those that constrain the correct behaviors of the composition. For example:* Property 1: A bike should not enter to a bike park unless it is dispatched from any bike park. (Violation of this boolean property indicates ISS is malfunctioning in some bike park.)

Property 2: Allow the client to pay only if there is a bike available in a bike park nearer to the requested location by the client. (Non-violation of this boolean property ensures that MRS accept rent if and only if there is at least one free (not booked) bike available at the requested location.)

Property 3: A bike should not go out of a bike park if a key for that bike is not issued by the MRS prior to it. (Violation of this boolean property indicates MRS is malfunctioning.)

Property 4: A person can take a bike out of the bike park if and only if a bike key is issued to him/her earlier and he/she has already paid the rent. (Non-violation of this boolean property ensures that MRS accepts rent and issued a bike key to the person who is actually riding the bike out of the bike park.)

*The MRS provider may also be interested in counting number of times a given event occurs in the execution of MRS. For example:* Property 5: Count the number of offers offered to the user before the user accepts. (This information may be helpful for the MRS to know the most popular rent offer.)

*The MRS provider may also be interested in measuring the time spent to perform certain activities, for example:* Property 6: Measure the MRS transaction completion time. (Sudden increase/decrease of MRS transaction time may lead to some functional problem.)

Property 7: Measure the MRS transaction completion time excluding payment time. (Sudden significant increase/decrease of MRS transaction completion time may lead to some functional problem, excluding RPS.)

*The MRS provider may also be interested in collecting statistical information, related to all instances of MRS. For example:* Property 8: The rent request is never refused by MRS.

Property 9: Count the number of times the bike is unavailable.

Property 10: Measure the average MRS transaction completion time. (This can give a statistical idea about MRS transaction completion time.)

*The MRS provider may also be interested in monitoring properties, related to a particular subset instances of MRS, satisfying certain constraints on internal variables of events. For example:* Property 11: Number of times bike is not available at "Barmunda bike park". (This will help to take decision to increase number of bikes in "Barmunda bike park".)

Finally, the MRS provider may also be interested in monitoring properties related to events coming from different layers. *For example:* Property 12: Number of times bike is not available at "Puri bike park" during "Car Festival". (This will help to take decision to increase number bikes in "Puri bike park" during "Car Festival".)

4. WS-AGREEMENT

WS-Agreement is a standard developed by the Global Grid Forum for specifying agreements between service providers and service consumers [Andrieux et al. 2004]. Its objective is to specify guarantee terms that should be satisfied during service provisioning. A WS-Agreement is expressed as an XML schema which consists of two sections, namely, the Context section and the Terms section.
The Context section specifies the consumer and the provider of the service that have created the agreement (i.e., the parties of the agreement) and other general properties of the agreement such as duration of service availability, links to other agreements, if any.

The Terms section of a WS-Agreement specifies the service that the agreement is about and the conditions that the service should fulfill. This section is divided into two sub sections: the Service Description Terms and Service Guarantee Terms. The service description terms constitute the basic building block of an agreement and define the functionalities of the service that is to be delivered under the agreement. An agreement may contain any number of service description terms. The guarantee terms specify assurances on functional and non-functional properties, quality of service and business assumptions of the SBS that need to be monitored and enforced during service provisioning.

5. MONITORING FRAMEWORK

Our SBS monitoring framework has been designed with the objective to support three different key monitoring features for SBS. The three key features of this approach are: (i) the monitoring is performed in parallel with the operation of an SBS without affecting its performance, (ii) non-intrusive SBS monitoring (i.e., monitoring without interfering the SBS process execution or without changing the original SBS), and (iii) the monitoring framework is independent of the service composition platform.

The framework facilitate the specification of SLA guarantee terms included in SLA between service provider and service consumer, as well as specification of SBS properties for monitoring. The framework monitor the operations of SBS at run-time to see whether certain specified properties and/or SLA guarantee terms are satisfied or not and indicate any deviations once they are detected.

Our monitoring framework is depicted in Figure 3. Here, it is assumed that at runtime a process execution engine executes the composition process of an SBS and delivers its functionality while capturing events from all layers (business layer, service layer and infrastructure layer) and pushes the events into an Event Bus.

The framework has 3 main components, namely an Event Bus, a User Interface and a Monitoring Engine. Figure 3 shows a high level representation of the proposed framework.

5.1. Event Bus

The "Event Bus" collects events from the SBS and puts the events in an event queue. The monitors consume the events from the queue. The types of events the Event Bus receives
Fig. 3. SBS monitoring Framework

The format of the events are as follows:

[sourceID]eventName{[varType:varName=val]*}

where, sourceID is the source identification number (If the event coming from service layer then sourceID is the process instance number of the SBS. If the event is coming from infrastructure layer then sourceID is -1. The sourceID of business layer is 0.), eventName is the name of the event, varName is the name of a internal variable of the event. varType is the type of the variable varName (different varType are int/double/string), val is the value of the variable varName. One event can have no or some internal variables. Each variable is in the form of varType : varName = val. Two variables are separated by a ",".

The types of eventName the event bus accepts are as follows:

partnerService_I messageName
partnerService_O messageName

where, partnerService is the name of the partner service, I: indicates that the message is a input message for the partner service, O: indicates that the message is an output message for the partner service, messageName is the name of the message.

Examples of events:

[1]CS_O_rentRequest{string:UID=XYZ67432,string:location=Puri}
This is a service layer event with process instance number 1 with event name CS_O_rentRequest with two internal variables "UID" and "location". This event name indicates that rentRequest is a out going message from the partner service CS of MRS SBS.

[-1]virtualMachineLoad{int:load=60}: This is an example of infrastructure layer event, where virtualMachineLoad is event name with one internal variable "load".

[0]cartFestivalStart: This is an example of business layer event with event name cartFestivalStart.

5.2. Guarantee terms/SBS properties specification

Service guarantee terms i.e, functional and non-functional requirements and SBS business process assumptions are specified in our framework using an XML schema that is based on a temporal logic based language called Monitor specification language(ML)[Tripathy and Patra 2010a], called XMSL. In the following, an overview of MSL is given and then a complete definition of XMSL.

5.2.1. Overview of MSL. As we discussed in Section 1, the SLA guarantee terms that need to be monitored are needed to formally specified. A temporal logic based, executable language
called MSL [Tripathy and Patra 2010a] designed by the authors is used to formally specify the SLA guarantee terms, which is defined as follows:

In MSL, SBS properties are specified in terms of events. Here, an event is something that occurs at a specific instant in time in SBS domain. Events are categorized into three categories: Service Layer Events, Business Layer Events and Infrastructure Layer Events. For example,

— Sent/received messages by the composed service to/from the atomic services used in the composition. These service to service message communication events are classified as service layer events.

— Something interesting occur in the SBS business domain which may significantly affect the business of the SBS are categorized as business layer event. For example, if a carnival is takes place in a city then the business of ”Travel Agent Service” of that city may increase. So, the happening of carnival is a interesting event for ”Travel Agent Service”. And since these event occur in the business layer, these events are categorized as business layer events for ”Travel Agent Service”.

— If a service provider uses 4 virtual machines to run the services. If the of one of the virtual machine is exceeding the normal load limit then in near future there is a chance of that virtual machine failure. So this may be a interesting event for a service provider. Since this event is related to service infrastructure, we categorize the events from infrastructure layer as infrastructure layer event.

The grammar for specifying events in MSL is as follows:

\[
event ::= eventName | eventName.(condition) \\
eventName ::= [a-z][a-zA-Z0-9]* \\
condition ::= type var cond value | condition V condition | condition \land condition \\
type ::= int | double | string \\
var ::= [a-zA-Z0-9]+ \\
cond ::= \neq | = | > | < \\
value ::= [-+][0-9]+ | [-+][0-9]+.0-9]+ \\
\]

Semantics. From previous discussion we can assume that, finally for the framework the event is a message with a message name with zero or more internal variable with variable name, variable type and variable value. This part of the grammar facilitates the specification of events with the condition on the internal variables of the event, eventName specifies the message name, type, var and val specifies internal variable type (which may be int/double/string), internal variable name (which is a string) and internal variable value (which may be an integer number/a real number/a string) respectively. Condition is defined as type var cond value: where type is the data type of the variable(int/double/string), var is the name of the variable, cond is a logical condition (= / \neq / > / <) on variable and value is a value(number/string) to compare with the variable value.

The following grammar defines the boolean, temporal and statistical formulas. We distinguish boolean formulae b, which monitor properties that can be either true or false, and numeric formulas n, which monitor properties that define a numerical value (which include temporal and statistical formulae).

\[
b ::= event | b \lor b | b \land b | b \Rightarrow b | b = b | n = n | n > n | Y b | O b | H b | S b \\
n ::= C(b) | T(b) | b?n : n | n + n | n - n | n * n | n / n | NUM \\
NUM ::= [0-9]+ | [0-9]+.0-9]* \\
\]

A boolean formula can be an event, or an event with some comparison between internal variables of the event, or a past LTL [Emerson 1990] formula (operators Y, O, H and S), or a comparison between numeric formulas, or a logic combination of other boolean formulas. A numeric formula can be either a counting formula (operator C), or a time measurement formula (operator T), or an arithmetic operation on numeric formulas.
The operators $\lor$, $\land$, $\neg$, $=\,$, $>$, $<$ and $\Rightarrow$ have the same meaning as logical $\lor$, logical $\land$, logical $\neg$, logical $=\,$, logical $>$, logical $<$ and logical $\Rightarrow$. Past LTL formulas have the following meaning: $Y b$ means "$b$ was true in the previous step", $O b$ means "$b$ was true (at least) once in the past", $H b$ means "$b$ was true always in the past" and $b_1 S b_2$ means "$b_1$ has been true since $b_2$". Numeric formula $C(b)$ counts the number of times that boolean formula $b$ has been true since the creation of the process instance. Formula $T(b)$ counts the sum of the time-spans the formula $b$ remains true.

5.2.2. XMSL. An extension of the WS-Agreement is proposed here. In this extension following protocol is defined to specify the guarantee terms of the SLA. Figure 4 depicts guarantee term specification protocol.

The specification of guarantee terms can be done at the time of the specification of the WS-Agreement or can be done independent of the WS-Agreement specification and integrate at the end.

**MSL Specification of MRS Properties:**

Some of the properties we have introduced in Section 3 can be defined by the following MSL formulae:

Property 1: A bike should not enter to a bike park unless it is dispatched from any bike park. $MRS_I_{bikeIn} \Rightarrow O(MRS_I_{bikeOut})$

Property 2: Allow the client to pay only if there is a bike available in a bike park nearer to the requested location by the client. $MRS_I_{startPayment} \Rightarrow O(MRS_I_{available})$

Property 3: A bike should not go out of a bike park if a key for that bike is not issued by the MRS prior. $MRS_I_{bikeOut} \Rightarrow O(MRS_O_{bikeKey})$
Property 4: A person can take a bike out of the bike park if and only if a bike key is issued to him/her earlier and he/she has already paid the rent.
\[ MRS.I_{bikeOut} \Rightarrow (O(\text{MRS.O_{bikeKey}}) \Rightarrow O(\text{MRS.I_{paymentSuccess}})) \]

Property 5: Count the number of offers offered to the user before the user accepts.
\[ C(MRS.I_{notAvailable}) \]

Property 6: Measure the MRS transaction completion time.
\[ T(\neg(\text{MRS.O_{bikeKey}}) S (\text{MRS.I_{rentRequest}})) \]

Property 7: Measure the MRS transaction completion time excluding payment time.
\[ T(\neg(\text{MRS.O_{bikeKey}}) S (\text{MRS.I_{rentRequest}})) - T(\neg(\text{MRS.I_{paymentSuccess}} \lor \text{MRS.I_{paymentFail}}) S (\text{MRS.O_{makePayment}})) \]

5.3. Monitor Engine

Monitor engine is the most important and most complex part of the framework. It has 4 main components, namely Monitor Generator which generates the monitors, Monitor Repository which stores all the monitors, Monitor Handler which receives events from event bus and sends the received events to appropriate monitors in the Monitor Repository and Monitor result DB stores the results of the monitors.

5.3.1. Monitor Generator. This is a MSL compiler, designed using Bison [Donnelly and Stallman 1992] as parser generator and Flex [Paxson et al. 1988] as lexical analyzer generator. This compiler translates the MSL specified formula to a C program named MonitorID.c and stores it in the Monitor Repository, where ID is the serial number of the monitor. Also the name of the created monitor (i.e, MonitorID) is registered (i.e, stored) in the Monitor Registry (i.e, a registry which stores name of created monitors). MonitorID.c contains a parse tree of the MSL formula and a parse tree update function implementing Algorithm 1.

Each node of the parse tree along with its child sub trees represent a formula. Each node of the parse tree stores the formula values (truth/numerical). Hereafter, ”node value” would mean the value of the formula it represents. The root node stores the value of the total formula i.e, value of the monitor. When the Monitor Handler wakes up a monitor by sending an event, the Update-Tree function updates the formula value at each node of the parse tree of the corresponding monitor according to the following formula value update rules.

**Algorithm 1: Update-Tree(event, Monitor)**

if eventName is matching with a node of the parse tree of Monitor.c then
  — Update node values of all nodes of the parse tree using “Formula Updating Rules”.
  — Store root node value as the current monitor result of this monitor in Monitor result DB.
end

**Formula Update Rules.** : Update-Tree function uses following rules to update the parse tree node values i.e, the value of sub formulas of a total formula.

1. \( fVal(condition) \) i.e. \( fVal(type \ var \ cond \ value) \)
   i.e, Formula value of a condition := true
   "If one of the internal variable of the occurring event has \( type = \) type, \( name = var \) and has the value satisfying \( cond(i.e, = | \neq | > | <) \) comparison to value”.

2. \( fVal(condition \land \ condition) := fVal(condition) \land fVal(condition) \)
3. \( fVal(event) := true \) ”If event is occurring”.
4. \( fVal(event.condition) := fVal(event) \land fVal(condition) \)
5. \( fVal(Y \ b) := oldFVal(b) \)
6. \( fVal(O \ b) := old \ fVal(Ob) \lor fVal(b) \)
(7) \( fVal(H b) := old fVal(Hb) \land fVal(b) \)  
(8) \( fVal(b1 S b2) := fVal(b2) \lor (old fVal(b1 S b2) \land fVal(b1)) \)  
(9) \( fVal(C b) := \text{if } fVal(b) \text{ then } (old fVal(C b) + 1) \text{ else } fVal(C b) \)  
(10) \( fVal(T b) := \text{if } fVal(b) \land old fVal(b) \text{ then } (old fVal(T b) + elapsed) \text{ else } old fVal(T bf) \)

Note
---
- \( fVal \) of \( b \lor b2 \mid (b1 \land b2) \mid (b1 \Rightarrow b2) \mid \neg b1 \mid (b1 = b2) \mid (b1 > b2) \) are as per the normal logical operator rule. For example: \( fVal(bf1 \land bf2) := fVal(bf1) \land fVal(bf2) \)  
- \( fVal \) of \( b?n1 : n2 \mid n1 + n2 \mid n1 - n2 \mid n1 \ast n2 \mid n1 / n2 \) are as per the standard mathematical rules.

Example: The following example conceptually shows the function of the Update-Tree algorithm. As an example let us conceptually demonstrate the monitor for Property 1 as mentioned in Section 3 i.e, the conceptual structure of the generated ”parse tree” and the function of Update-Tree on the generated ”parse tree”.

Property : Payment process starts only after client accepts the offer.

MSL specification: \( MRS \circ makePayment \Rightarrow O(MRS \circ startPayment) \)

Left hand side of the Figure 5 shows conceptually the structure of parse tree and the right hand side figure of Figure 5 shows the effect of Update-Tree function on it after receiving the event \( MRS \circ startPayment\{\text{int:accInfo=546718}\} \).

5.3.2. Monitor Handler. It is responsible for receiving new events from Event Bus, creating the required new instances of the existing monitors in the Monitor Repository and waking up appropriate monitors to consume the incoming event. The following Event-Monitor-Handler algorithm does all these tasks.

<table>
<thead>
<tr>
<th>ALGORITHM 2: Event-Monitor-Handler(event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find sourceID of the event.</td>
</tr>
<tr>
<td>if sourceID ( \leq 0 ) then</td>
</tr>
<tr>
<td>Wake up all instances of all monitors stored in the Monitor Repository to consume the incoming event.</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>for each MonitorID stored in the Monitor Registry do</td>
</tr>
<tr>
<td>if sourceID in a new sourceID then</td>
</tr>
<tr>
<td>— Add the sourceID in the sourceID list against the MonitorID.c</td>
</tr>
<tr>
<td>— Create a new instance of MonitorID.c and save this instance as MonitorID.sourceID.c in the Monitor Repository.</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>Update-Tree(event, MonitorID.sourceID.c)</td>
</tr>
</tbody>
</table>
5.4. Aggregation Functions

To monitor the properties related to multiple instances of SBS systems two aggregation functions named **ForAll** and **AddAll** are added in the framework. These functions are designed above the results of monitors (specified using MSL) stored in MonitorResultDB. In addition to that a run time sql query on MonitorResultDB is added in the framework to facilitate the user to make some reasoning on the outputs of the running monitors if necessary.

— **ForAll(b)**: If \( b \) (a boolean MSL formula) is true for all instances of SBS process then **ForAll(b)** is true, otherwise false.
— **AddAll(n)**: It add value of \( n \) (a numeric MSL formula) for all instances of SBS process.

Using these two aggregation functions now we can specify and monitor SBS properties and SLA guarantee terms related to multiple running instances of SBS process. For example, property 8, 9, 10, 11 & 12 for MRS can be specified using these two aggregation functions and MSL as follows.

Property 8: The rent request is never refused by MRS.

\[
\text{ForAll}(\neg \text{MRS\_rentRequest})
\]

Property 9: Count the number of times the bike is unavailable.

\[
\text{AddAll}(C(\text{MRS\_notAvailable}))
\]
Property 10: Measure the average MRS transaction completion time.
\[
AddAll\left( T\left( \neg\left( MRS_O\text{bikeKey} \right) \right) S\left( MRS_I\text{rentRequest} \right) \right) / AddAll\left( C\left( MRS_I\text{rentRequest} \right) \right)
\]

Property 11: Number of times bike is not available at “Barmunda bike park”.
\[
AddAll\left( C\left( \left( MRS_I\text{notAvailable} \land \neg\left( MRS_I\text{rentRequest}.\left( \text{location} = \text{"Barmunda"} \right) \right) \right) \right) \right)
\]

Property 12: Number of times bike is not available at “Puri bike park” during “Car Festival”.
\[
AddAll\left( \neg\left( \text{CartFestivalEnd} \right) \text{S}\left( \text{CartFestivalStart} \right) \right)
\]

6. CONCLUSIONS
An event based, non-intrusive monitoring framework has been proposed that separates business logic from the monitoring functionality and supports cross-layer SLA monitoring. The proposed framework does not depend on the service composition platform. A protocol using MSL and XML has been developed to formally specify SLA guarantee terms. The SLA guarantee terms expressed in the form of XSML specifications are automatically translated into an executable C program which is used by the framework while monitoring the specified behavior of the system.

We continue our work to extend the proposed framework for service based system failure-handling, repair and adaptation triggered by information provided by the monitors. Further, we plan to provide an experimental evaluation of the usability and practical effectiveness of the proposed framework in different application domains.

ACKNOWLEDGMENTS
We thank the team members of the Service Oriented Application unit of FBK, Trento, Italy for their valuable suggestions and validation of concepts incorporated in this work.

REFERENCES


