MSL Based Service Based e-Governance Systems Monitoring

Ajaya Kumar Tripathy, Department of Computer Science, Utkal University, Bubaneswar, India
Manas Ranjan Patra, Department of Computer Science, Berhampur University, Berhampur, India
Sateesh Kumar Pradhan, Department of Computer Science, Utkal University, Bubaneswar, India

In many occasions citizens have to interact with multiple government organizations to complete a task which may involve lot of manual effort. In order to streamline the task different governments have adopted e-governance. But this has raised the issue of interoperability. In recent years web services seems to provide a solution towards interoperability both at the organizational and application level. It is possible to integrate inter-related tasks of different organizations through service composition techniques to form single service based applications. However, as government organizations function autonomously it is necessary to monitor these service based systems (SBS) at run-time. Monitoring SBS in a non-intrusive and composition platform independent manner is a real challenge. In this paper, we propose a framework for monitoring the compliance of SBS for which a set of requirements have been pre-specified. A Monitor Specification Language has been developed to specify the properties of the system to be monitored at run-time. The monitoring framework has been applied to a government document verification scenario as a proof of concepts.

Categories and Subject Descriptors: H.3.5 [Web-based services]: Web-based services
General Terms: Verification, Language, Design
Additional Key Words and Phrases: e-Governance, Web Services, Monitoring, Document Verification

1. INTRODUCTION

In most cases government organizations are quite independent in nature, in the sense that they operate as administrative silos within their defined functional boundaries. However, in many occasions they need to interact with each other on issues involving multiple organizations. Normally, the nature of interaction is in the form of document flow, enquiries, request for verification of authenticity of certain documents, etc. Invariably such interactions involve lot of manual official processes. As a result the process becomes quite cumbersome, lengthier, time consuming and sometimes error prone leading to citizen dissatisfaction and degradation in organizational performance. Therefore, there is a need for suitable mechanism that can facilitate interoperability among organizations.

In recent years, the Service Oriented Architectural framework along with the Web-Service technologies is increasingly being used as a platform for designing independent applications as services which can easily interoperate. Several technologies such as BPEL, ESB are being used as the enabling technologies for service composition and a means to achieve interoperability among organizational processes. Now a day’s SOA becomes the most popular platform for designing e-government systems. That is e-government services becomes web service based systems.

A web service is an application that exports a description of its functionality and makes it available using standard network technologies. These functionalities can be accessed through standard XML messages over internet [W3C 2002]. Software systems that are composed of autonomous web services through a composition process are referred to as ”Web Service Based Systems” (SBS).

The ability to set up a SBS monitoring framework has been increasingly recognized as one of the essential preconditions for the deployment of a SBS as an e-government service. This is because the government organizations involved in the e-government service are running independently. Such organization’s services are developed and managed autonomously and can change without notification leading to run-time problems. As and when such deviations are detected those must be captured and analyzed so as to take appropriate action. For instance a student applying for declaration of value (DOV) of his certificate at the
consulate, this DOV process involves consulate, state government, university, and foreign affairs ministry departments. The applicant may want to know why the process got delayed. The occurrence significant delayed of this process has to be reported as soon as possible so that the responsible authority can take prompt action.

Some of the recent works have addressed different aspects of monitoring a SBS, e.g., [Barbon et al. 2006; Chau et al. 2008; Keller and Ludwig 2003; Mahbub and Spanoudakis 2007; Sahai et al. 2002]. In this paper we propose a novel solution to the problem of web services based e-government system monitoring SBS.

A monitoring framework has been developed which is independent of any business logic and service composition platform. The monitoring engine works in parallel with the SBS and allows for easy adaption of the business process. The SBS sends interesting events from the business layer, service layer (incoming/outgoing messages to/from the services used in the SBS) as well as from the infrastructure layer and feeds those into an event bus at run-time. A monitor observes the events from the event bus and accordingly monitors the functional and non-functional service composition assumptions and requirements of the SBS.

Further, we provide a temporal logic based, RTML (Run-Time Monitor specification Language) type expressive language for specifying service composition assumptions, and functional as well as non-functional requirements of an SBS. The language allows for specifying boolean, static and time related properties. Beyond composition assumptions, we can also specify properties related to cross layer events. We have also designed a monitor generator which automatically generates a C program for the monitor which is deployed at run-time, thus reducing the design and implementation efforts.

The rest of the paper is structured as follows. In Section 2, we describe the state of the art in the Web Service Technology and existing research approaches to the monitoring of SBS. Section 3, describes an example scenario. Section 4, describes our monitor specification language. Section 5, gives complete description of the framework. In Section 6, we conclude our work and describe our future direction of research.

2. STATE OF THE ART

WEB SERVICES

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 a 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.

SBS 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.
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 adapt to new property requirements.

Baresi 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 affect 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) [Thatte 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.

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 MSL (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 SBS 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 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. DECLARATION OF VALUE (DOV) OF CERTIFICATES

Here we describe a typical document verification scenario which requires interaction among different organizations. Whenever students apply for higher studies in foreign universities they need to produce a Declaration of Value (DOV) certificate of the qualifying degree from the consulate office of the concerned country. Normally it is a cumbersome process and involves four major establishments, namely, the consulate of the foreign country where admission is sought, external affairs ministry of the country to which the candidate belongs, provincial secretariat office, and the university who has issued the degree for which the DOV is required. The entire process is depicted in the following steps.

(1) Candidate applies for DOV to the consulate office after obtaining admission related papers from the foreign university.
(2) The consulate office verifies the DOV application and the admission related papers. If satisfied, it asks the candidate to bring the legalization of the qualifying degree.
(3) Next, the candidate applies to the secretariat office of the province from where the degree has been obtained.
(4) After preliminary scrutiny of the application the secretariat sends the degree certificate by post to the issuing university for further verification.
(5) The university verifies the certificate and informs the correctness of the degree to the secretariat again by post.
(6) On receiving the reply from the university, the secretariat returns the degree certificate to the applicant after proper attestation.
(7) Next, the applicant applies to the external affairs ministry to obtain the final legalization of the degree.
(8) The foreign affairs ministry verifies the attestation of the degree from the provincial secretariat and issues the final legalization of the document.
(9) The applicant submits the legalization document to the consulate. After this the consulate checks the corresponding value of this degree in their country and prepares the DOV.

Suppose the DOV process provided by the consulate is implemented as web service based system (SBS) called DVS that consists of a service composition process that interacts with following web services. Fig. 1 depicts the working process flow of DVS.
(1) **Certificate Verification Service (CVS):** This service is provided by the university where the certificate was issued. Where the CVS verifies the certificate and informs the correctness of the degree.

(2) **Certificate Legalization Service (CLS):** This service is provided by the province government office. Where the CLS verifies the identification of the applicant and uses CVS to know the correctness of the certificate. Then issue the legalization proof of the certificate.

(3) **Legalization Authentication Service (LAS):** This service is provided by Indian government foreign affairs ministry office. Where the LAS receive the legalized certificate issued by the CLS and verify the identification of the authority who issued this. Then issue the final legalization document of the certificate.

(4) **Client Service (CS):** This service provides the DVS service a frontend that handles interaction with the end-user.

In this example scenario, we assume the message flows of DVS are as follows. CS activates the DVS for getting the DOV of a degree certificate by sending a DOV request message containing applicants identification, previous university from where the certificate was issued, the province in which the issuing university belong, certificate need to submitted for getting DOV, university where the student got selected and admission selection proof i.e, `dovReq(appID, prevUniv, province, , certificate, futureUniv, selectionProof)`.

The DVS verifies the selection proof of the student, if it is not ok then it stops the process by sending the CS a reject message with reason i.e, `reqRej(reason)` otherwise invokes CLS by sending a certificate legalization request message with applicant identification, province and certificate i.e, `legReq(appID, certificate, prevUniv)`.

CLS invokes the CVS to verify the correctness of the certificate by sending a verification request message i.e, `verReq(appID, certificate)`.

CVS verifies the certificate, if it is correct then replies with a correct message i.e, `correct`, otherwise with a not correct message with reason i.e, `notCorrect(reason)`.

![Fig. 1. DVS process work flow](image)
If the CLS gets a notCorrect message it acknowledge the DVS that the certificate is not correct likewise DVS to CS, otherwise the CLS replies the DVS a legal message with a attestation signature of competent authority i.e, legalized(certificate, authoritySign, province).

Then DVS invokes the LAS to verify the legalization made by province by sending a verification message i.e, legVer(certificate, authoritySign, province). The LAS verifies the signature. If it is ok then it replies DVS saying legal with legal(certificate) message, otherwise replies saying not legal with reason by sending notLegal(certificate).

If DOV gets the notLegal(reason) message from LAS, it acknowledges the CS about it and stops the process otherwise, it makes the required process and prepare the DOV of the certificate and sends to the CS with DOV(certificate, dov). Fig. 2 depicts the message flows of DVS.

Despite of the simplicity of the domain, due to the independent functioning nature of the involved organizations in this DOV process, there are several things that the DVS needed to monitor at run time due to the interest of all stockholders. For example:

1. Unless the consulate receives a request for DOV from the CS it will not request the secretariat for the certificate legalization process.
2. Count the number of times the DVS replies that the certificate is not legal.
3. Compute the time required to complete the legalization procedure in the secretariat.
4. Count the number of times the DVS replies that the certificate is illegal for "Bihar" (a name of a province) province.
5. Count the number of DOV request during March.

4. MONITOR SPECIFICATION LANGUAGE

The properties of the SBS that need to be monitored are expressed in a temporal logic based, executable language, which is defined as follows:

In this language SBS properties are specified in terms of events. An event is something that occurs at a specific instant in time. Grammar:

\[ event ::= \text{eventName} | \text{eventName.\text{(condition)}} \]
Logical Monitoring Engine. Fig. 3 shows a high level representation of the proposed framework.

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

eventName ::= [a−z][a−zA−Z0−9]*

condition ::= type var cond value | condition V condition | condition ∧ condition
type ::= int | double | string
var ::= [a−zA−Z0−9]+
cond ::= ≠ | = | > | <
value ::= [−+][0−9]+ | [−+][0−9]+[.0−9]* | [a−zA−Z]*

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 (= / ≠ / > / <) 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 formula n, which monitor properties that define a numerical value (which include temporal and statistical formulae).

b ::= event | b ∨ b | b ∧ b | b ⇒ b | ¬ b | n = n | n > n | Y b | O b | H b | 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(int/double/string), 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 V, ∧, ¬, =, >, < and ⇒ have the same meaning as logical V, logical ∧, logical ¬, logical =, logical >, logical < and logical ⇒. 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 b1 S b2 means "b1 has been true since b2. 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. 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 with out changing the original SBS), and (iii) the monitoring framework is independent of the service composition platform.

In this framework a human user (typically, provider of an SBS) can request to monitor the runtime operations of a system to see whether certain specified properties are satisfied or not and indicate any deviations once they are detected.

Our monitoring framework is depicted in Fig. 3. Here, it is assumed that at run-time 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. Fig. 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 are: messages received from the composition process or sent to the composition process by one of its partner services, interesting events from business layer/Infrastructure layer.

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 an 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 an 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]CVS_O_DOV{string:reason=selection proof is not correct}: This is a service layer event with process instance number 1 with event name CVS.O.DOV with one internal variable "reason". This event name indicates that DOV is a outgoing message from DVS service.

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

[0]monthStart{string:monthName=March}: This is an example of business layer event with event name monthStart.

MSL Specification. The properties we have introduced in Section 3 can be defined by the following MSL formulae:
(1) Unless the consulate receives a request for DOV from the CS it will not request the certificate legalization process.

\[ \text{CLS}_I\text{legReq} \Rightarrow O(\text{CS}_0\text{dovReq}) \]

(2) Count the number of times the DVS replies that the certificate is not legal.

\[ C(\text{CS}_I\text{notLegal}) \]

(3) Compute the time required to complete the legalization procedure in the secretariat.

\[ T(\neg(\text{CLS}_0\text{notLegal} \lor \text{CLS}_0\text{legalised}) \lor \text{CLS}_I\text{legReq}) \]

(4) Count the number of times the DVS replies that the certificate is illegal for province "X".

\[ C(\text{CS}_I\text{notLegal} \Rightarrow O(\text{CS}_0\text{dovReq}.\text{(province=X)})) \]

(5) Count the number of DOV request during March.

\[ ((\neg \text{monthEnd.(mname=March)} \lor \text{monthStart.(mName=March)}) \Rightarrow C(\text{CS}_0\text{dovReq}) : O \]

**MONITORING**

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.

5.1.1. Monitor Generator. This is a MSL compiler, designed using Bison [Donelly 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.

**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) \( fVal(\text{condition}) \) i.e. \( fVal(\text{type}\ \text{var}\ \text{cond}\ \text{value}) \)

\[ i.e, \ \text{Formula value of a condition} := \text{true} \]

"If one of the internal variable of the occurring event has type \( \text{type} \), \text{name}=\text{var} and has the value satisfying \( \text{cond} \) (i.e, \( = | \neq | > | < \) comparison to \( \text{value} \))."

(2) \( fVal(\text{condition} \land \text{condition}) := fVal(\text{condition}) \land fVal(\text{condition}) \)

(3) \( fVal(\text{event}) := \text{true} "\text{If event is occurring}". \)

(4) \( fVal(\text{event.condition}) := fVal(\text{event}) \land fVal(\text{condition}) \)

(5) \( fVal(\text{Y b}) := \text{old fVal(b)} \)

(6) \( fVal(\text{O b}) := \text{old fVal(Ob)} \lor fVal(b) \)

(7) \( fVal(\text{H b}) := \text{old fVal(Hb)} \land fVal(b) \)

(8) \( fVal(b1 \ S \ b2) := fVal(b2) \lor (\text{old fVal(b1 S b2)} \land fVal(b1)) \)

(9) \( fVal(C b) := \text{if fVal(b) then (old fVal(C b) + 1) else fVal(C b)} \)

(10) \( fVal(T b) := \text{if fVal(b) \land old fVal(b) then (old fVal(T b) + elapsed) else old fVal(T bf)} \)

**Note**
— $fVal$ of $(b_1 \lor b_2) \mid (b_1 \land b_2) \mid (b_1 \Rightarrow b_2) \mid \neg b_1 \mid (b_1 = b_2) \mid (b_1 > b_2)$ are as per the normal logical operator rule. For example: $fVal(bf_1 \land bf_2) := fVal(bf_1) \land fVal(bf_2)$
— $fVal$ of $b?n_1 : n_2 \mid n_1 + n_2 \mid n_1 - n_2 \mid n_1 * n_2 \mid n_1 / n_2$ are as per the standard mathematical rules.

**ALGORITHM 1:** Update-Tree($eventName$, Monitor)

```plaintext
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
```

**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 following i.e, the conceptual structure of the generated "parse tree" and the function of Update-Tree on the generated "parse tree".

Property 1: Unless the consulate receives a request for DOV from the CS it will not request the secretariat for the certificate legalization process.

MSL specification: $\text{CLS\_I\_legReq} \Rightarrow O(\text{CS\_O\_dovReq})$

Left hand side of the Fig. 4 shows conceptually the structure of parse tree and the right hand side figure of Fig. 4 shows the effect of Update-Tree function on it after receiving the following event $\text{CS\_O\_dovReq}(\text{string:appID=x, string:prevUniv=u1, string:province=y, string:futureUniv=fu, string:selectionProof=p})$

![Fig. 4. Left: Parse Tree, Right: Updated Parse Tree](image)

5.1.2. **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.

5.2. **USER INTERFACE**

The "User Interface" gives access to the monitoring service for human users. This interface provides a web page for defining new monitors using MSL and for viewing the monitoring results of the deployed monitors. For instance, in Fig. 5 shows snapshot of one of the web pages of User Interface.
ALGORITHM 2: Event-Monitor-Handler(event)

Find sourceID of the event.
if sourceID ≤ 0 then
    Wake up all instances of all monitors stored in the Monitor Repository to consume the incoming event.
end
for each MonitorID stored in the Monitor Registry do
    if sourceID is a new sourceID then
        — Add the sourceID in the sourceID list against the MonitorID.c
        — Create a new instance of MonitorID.c and save this instance as MonitorID_sourceID.c in the Monitor Repository.
    end
end
Update-Tree(event, MonitorID_sourceID.c)

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented a framework for monitoring several properties of web service based e-Government systems. An event based approach has been proposed that separates business logic from the monitoring functionality and supports cross-layer SBS monitoring. The proposed framework does not depend on the service composition platform. Further, a monitoring language has been developed to formally specify the properties of a service based system. The specification is automatically translated into an executable C program which is used by the framework while monitoring the specified behavior of the system. The usability of the proposed framework has been shown in a typical document verification scenario.

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


