



Run-Time Monitoring of SLA for WSBS at Application Layer

Ajaya. K. Tripathy^{1,3}, Manas. R. Patra² and Sateesh K. Pradhan¹

¹Dept. of Computer Science and Application, Utkal University, India

²Dept. of Computer Science and Application, Berhampur University, India

³Dept. of CSE, Silicon Institute of Technology, India

Abstract: Run-Time Monitoring of SLA for Web Services Based Systems (WSBS) is an interesting as well as challenging topic in the field of Web Service Monitoring. Several works already address different aspects of the Run-Time monitoring. But most of the approaches are limited to composition layer (BPEL process). The behavior of composed web services is also affected by the behavior of other layers of the web service application domain. In particular, it is necessary to monitor the web services from the application layer. In this paper we propose an approach to achieve Run-Time monitoring of SLA for WSBS at application layer.

Keywords: Web Services Based Systems, SLA monitoring, Web Service Composition, BPEL

I. INTRODUCTION

Web Services [1] allow components to be exported as services for external use. Service providers publish service descriptions, which include functional and non-functional properties. Service discovery is based on matching the published service descriptions with the required service specification. Once the service is discovered, it may be bound and invoked remotely. New services may also be created by composing existing services. For full flexibility, the binding between a service request and the service provision may be set dynamically at run-time. Dynamic binding and decentralized management of external services by independent authorities may lead to violation of intended behaviour of Web Service at run time, however, demands run-time monitoring [2] of functional and non functional attributes of Web Services.

There already exist approaches for run-time monitoring of Web Services (e.g., [2-11]). These approaches monitor the Web Services at the level of the BPEL process (composition layer). The behaviour of Web Services is affected by the behaviour of other related services, users and different layers of Web Service application domain. (For example, increasing load in virtual machine may increase process completion duration. That is, behaviour of virtual machine layer can affect behaviour of process layer.) This demands to perform run-time monitoring of Web Services at application layer of the Web Service application domain.

The monitoring approach [11] is the suitable approach to extend for monitoring at application layer due to its expressive nature. Because this approach totally separate the monitoring process from the BPEL process and the monitoring engine runs parallel to the BPEL engine. In this paper we propose some important issues needed to make [11] successful at application layer. We also propose a novel approach to resolve the issues.

The paper structure is as follows. Section II, presents an explanatory example used through the paper. Section III, introduces some important existing run-time monitoring approaches. Section IV, gives a brief introduction of RTML (Run-Time Monitor specification Language) for Run-Time Monitoring of Web Services, while Section V, describes some issues to be resolved in order to make RTML successful at the application layer. Section VI, outlines approaches to resolve the issues described in Section V. Section VII, concludes the paper and describes the future work.

II. ONLINE PAYMENT SERVICE

The Online Payment Service (OPS) offers a payment service by interacting with one external Card Validation Service (CVS) (see Fig. 1). When a client sends the card information to the OPS for an online payment, OPS sends the card information to CVS for authentication. If the card is valid, OPS get back a positive acknowledgement. In this case, the OPS proceeds for payment and the client is acknowledged positively. Otherwise the client is acknowledged negatively. The abstract BPELs of both the services are depicted in Fig. 2.

After receiving a request from a client, the OPS, gets activated, and the request includes information about the card owner name (CN), card ID (CID) and amount (*[RECEIVE]paymentRequest box*). If the payment is successful, the client is informed about this (*[invoke]paymentDebit*). Otherwise the client is informed that payment failed (*[INVOKE]paymentFail box*). The validity condition (*[SWITCH] valid?*) depends on an interaction between the OPS and CVS which is hidden to the client.

After getting a request from the OPS, the CVS gets activated, and the request includes CN and CID. The CVS checks the validity of the card (*[SWITCH]valid?*), and replies either that the card is valid if the card is valid (*[INVOKE] cardValid*) or invalid (*[INVOKE]cardNotValid*).

III. RUN-TIME MONITORING OF WEB SERVICES

This section introduces a number of important and relevant run-time monitoring approaches. Our paper based on one of these approaches [11]. Spanoudakis and Mahbub [6,7] propose a framework for monitoring the systems composed of web services. This framework assumes systems composed of web services that are coordinated by a service composition process expressed in BPEL4WS and uses event calculus to specify the properties to be monitored. This framework collects behavioral properties of the BPEL process and assumptions about the behavior of services. Collected information is stored in a database. Properties are then confirmed against the collected data, using integrity-checking techniques in temporal databases.

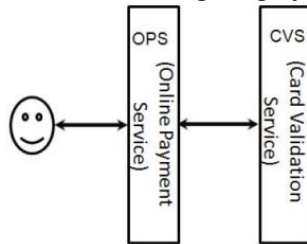


Fig. 1. The Online Payment Service example

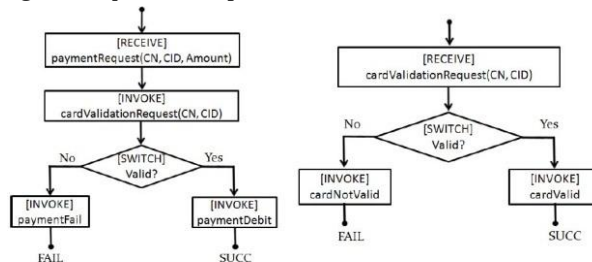


Fig. 2. Abstract BPELs of the OPS and CVS

Lazovik et al. [8] 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.

Baresi et al. [10] proposed an approach and designed a toolset to support service monitoring. This approach monitors both functional correctness of BPEL orchestration and QoS agreements set between the service provider and the client. They provide a language called WSCoL (Web Service Constraint Language) [9] which allows designers to specify constraints on BPEL orchestration. Appropriate external services called monitoring managers are responsible for analyzing WSCoL constraints. If some constraints are not met, that is, if some monitoring rules are not satisfied, the monitoring manager is informs the BPEL. The business logic is unaffected by monitoring, but to allow the process to interact with the external monitors, additional BPEL code is added to the process at deployment time.

Fabio et al. [11] proposes a novel solution to the problem of monitoring web service compositions. They clearly separate the business logic of a Web Service from its monitoring functionality, propose a language RTML (Run-Time Monitoring specification Language) to specify the monitoring properties and also devised a technique for automatically translating all these kind RTML specified properties to monitors (Java programs). A detailed description of this solution is given in the next section.

IV. RUN-TIME MONITORING OF SLA FOR WSBS

In [11], monitors are independent software modules that run parallel to BPEL processes by intercepting the input/output messages of the processes. This approach supports two different kinds of monitors: "instance monitors" and "class monitors". Instance monitors observe the execution of a single instance of a BPEL process. Class monitors report aggregated information on all the instances of a given BPEL process. Fabio et al. [11] designed a language (RTML) for the specification of the instance and class monitors. RTML allows for specifying boolean, static and time-related properties. [11] also devised a technique to automatically generate and deploy monitors, using the RTML specified properties. The monitoring framework is described in three steps:

- The language for specifying the "events" that is relevant for the evolution of monitors.
- On top of events, the language for specifying instance monitors.
- On top of instance monitors, the language for class monitors.

Events:

- The creation and termination of a process instance; these two events are modeled through keywords "start" and "end" in RTML.
- The input and output of messages; in this case, RTML requires specifying the link on which the message is received or sent, the fact that the message is an input or an output, and the message type. Examples: *msg(OPS.input = paymentRequest)*, *msg(OPS.output = paymentDebit)*, etc.

Complete Grammar for Events e:

- $e ::= \text{start} \mid \text{end} \mid \text{msg}(\text{link.input/output} = \text{msg}[\text{opt-constraints}]) \mid \text{cause}(\text{link.var} = \text{val}) \mid \text{cause}(\text{link.state} = \text{label})$

Grammar for Instance Monitors:

- Boolean formula $b ::= e \mid Yb \mid Ob \mid Hb \mid bSb \mid n = n \mid n > n \mid \sim b \mid b \wedge b \mid true$
- Numeric formula $n ::= count(b) \mid time(b) \mid b?n:n \mid n + n \mid n - n \mid n * n \mid n/n \mid 0 \mid 1 \mid \dots$

Where Yb means " b was true in the previous step"; Ob means " b was true (at least) once in the past"; Hb means " b was true always in the past"; $b_1 S b_2$ means " b_1 has been true since b_2 ".

Grammar for Class Monitors:

- Boolean formula $B ::= And(b) \mid YB \mid OB \mid HB \mid BSB \mid N = N \mid N > N \mid \sim B \mid B \wedge B \mid true$
- Numeric formula $N ::= Count(b) \mid Sum(n) \mid N + N \mid N - N \mid N * N \mid N / N \mid 0 \mid 1 \mid \dots$

$And(b)$ checks if property b is true for all the instances. $Count(b)$ counts the number of instances for which formula b holds. $Sum(n)$ is similar, but aggregates numeric instance module formulas. For a detailed description of RTML, we refer the interested reader to [11].

V. RUN-TIME MONITORING OF WEB SERVICE AT APPLICATION LAYER

The behaviour of underlying business logics in Web Services is sensitive not only to the behaviour of BPEL process layer but also to the behaviour of other layers. For example OPS payment can fail for a particular user with $CN="X"$, payment time increase if assigned virtual machine is loaded above 70 percent, there can be a payment delay due to delay of card validation, etc. Monitoring all these types of abnormality is not possible at BPEL process layer. So instead of monitoring the web services at BPEL process layer, it is necessary to monitor at application layer.

The approach we propose in this paper is to extend [11] for the application layer. Initial salutations and applications were published by the authors in [13-19]. This section describes the fundamental needs of web service monitoring at application layer. We then discuss the limitation of [11] with respect to these fundamental needs. That means, we move from "monitoring at composition layer" to "monitoring at application layer".

Let's consider some monitoring properties at the application layer that OPS may want to monitor, as follows:

- Whether the CVS replies "*cardNotValid*" in the past or not.
- What is the time taken by the OPS to complete the recent request?
- What is the average time taken by the OPS?
- Average payment time taken by OPS for $CN="X"$.
- Time taken by OPS excluding the time taken by CVS.
- Time taken by OPS to complete the request when the assigned virtual machine is loaded over 60 percent.

If we analyse these properties carefully we can classify them into five categories. The first category consists of properties related to single instances. Property 1, 2 come under this. These properties can be handled by instance monitors specified by RTML as follows:

Property 1: $O msg(CVS.output = cardNotValid)$

Property 2: $time((OPS.state = SUCC, FAIL) S msg(OPS.input = paymentRequest))$

The second category is the properties related to all instances of a single BPEL process. Property 3 comes under this. This property can be handled by a class monitor specified by RTML as follows:

Property 3: $Sum(property\ 2)/Count(msg(OPS.input = paymentRequest))$

Category 1 and 2 include BPEL process layer properties, so RTML is capable to monitor them. Now we will analyse the remaining properties, which are related to different layers of Web Service application domain and cannot specified using RTML. For each of these properties we will identify the major issues we need to resolve in order to make RTML capable of specifying it.

BPEL Engine Mediator: provides events from the BPEL engine to Monitor generator engine.
PR: paymentRequest, PD: paymentDebit, PF: paymentFail

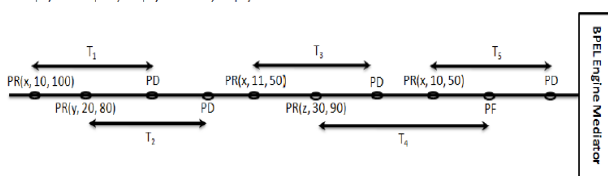


Fig. 3. Time line diagram of events for explanation of Issue 1

BPEL Engine Mediator: provides events from the BPEL engine to Monitor generator engine,
PR: paymentRequest, PD: paymentDebit, PF: paymentFail,
CVR: cardValidationRequest, CV: cardValid, CNV: cardNotValid

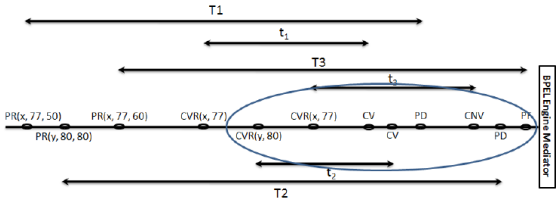


Fig. 4. Time line diagram of events for explanation of Issue 2

Cross Instance Correlation:

The third category of properties is the properties related to a subset of the instances of a single BPEL process which have a particular input data. Property 4 comes under this category. This property cannot be monitored by instance monitors or by the class monitors as described in RTML. To illustrate the problem more clearly, consider the time line diagram in Fig. 3.

Fig. 3 shows some of the events generated by the BPEL Engine Mediator. The events are correlated by the BPEL process Ids [11]. In this figure, $T_1, T_2, \dots$ are the durations computed by Property 2, i.e., the time taken by OPS to complete each request. Taking all the values calculated by Property 2, we are able to specify Property 3 in RTML. However, Property 4 cannot be specified in RTML. The reason is that for calculating Property 4 the monitor generator needs to choose those T 's which are related to $CN="X"$. There is no such grammar in RTML

which may correlate the instances related to client "X". This problem demands cross instance correlation in RTML.

Cross Service Correlation:

The fourth category of properties is related to the correlation of properties and events generated by different services. Property 5 comes under this category. To explain this problem we will use Fig. 4. The time durations $T_1, T_2, T_3, \dots$ are calculated by Property 2 for each OPS BPEL process instance. Similarly $t_1, t_2, t_3, \dots$ are calculated by Property 2 for each CVS BPEL process instance.

For payment time (T) there is related card validation duration (t). Now if we calculate Property 5, the result should be $(T - t)$. Since RTML does not support cross service correlation, it does not support this correlation between T and t. If you consider the events before the ellipse (in Fig. 4) the problem is very clear. When CVR(X, 77) is created it is not clear whether it is related to PR(x, 77, 50) or PR(X, 77, 60). In other words, using RTML we cannot correlate events coming from different services. Hence there is a need of cross service correlation.

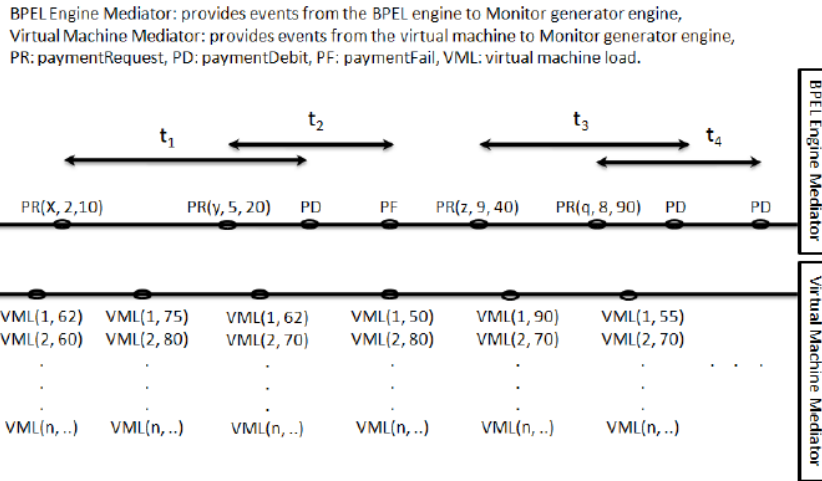


Fig. 5. Time line diagram of events for explanation of Issue 3

Cross Layer Correlation:

The fifth category of properties is related to the events from BPEL process layer as well as from other layers of web service application domain. Property 6 belongs to this group. Property 6 is related to the events coming from the process layer as well as from virtual machine layer. For calculating property 6, it is necessary to correlate the data coming from both layers. The problem is clearer, if we consider the example shown in Fig. 5. Here, we are getting events from BPEL engine mediator as well as the Virtual machine mediator. The BPEL engine mediator generates the events as described in Section IV. The virtual machine mediator periodically sends the load information of each virtual machine. It is clear that, correlating the events coming from both sources is out of the scope of RTML, and demands cross layer correlation in RTML.

Cross instance, cross service and cross layer correlations are the major issues needed to be resolved in RTML to make it fit at application layer.

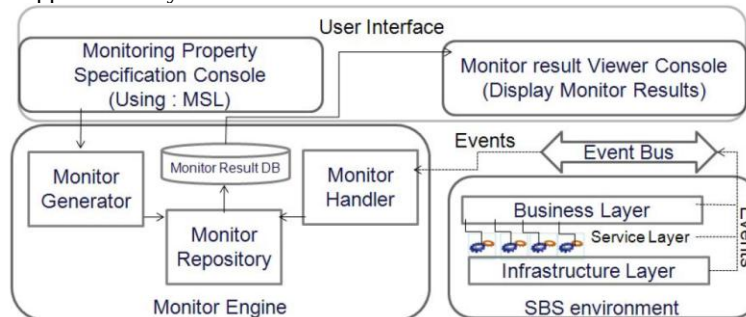


Fig.6. Modified monitoring framework.

VI. CROSS LAYER MONITORING

In order to resolve the issues mentioned in Section V, following steps were followed:

- *Collecting events from all layers:* Along with all the events mentioned in Section IV we can also collect the relevant events from all layers of the Web Service application domain. For example as in Fig. 5, VML(Virtual Machine ID, Load) from the virtual machine layer. Further, we collect the correlated events between different layers. From the BPEL engine mediator, such an event is PAV(Process ID, Assigned Virtual Machine ID). This event correlates processes with virtual machines. Taking both VML and PAV, it is possible to calculate property 6.

- *Extending the grammar of RTML*: It is necessary to extend the language to support cross instance, cross service and cross layer correlation among the events.
- *Extending the monitor generator engine with respect to the extended RTML*.

The modified monitoring framework is depicted in Figure 6.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented a framework for monitoring several properties of web service based 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 behaviour of the system. Further, we plan to provide an experimental evaluation of the usability and practical effectiveness of the proposed framework in different application domains.

ACKNOWLEDGEMENT

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

1. Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju. Web Services. *Springer Verlag* 2004, chapter 5, pages 123-149.
2. Luciano Baresi, elisabetta Di Nitto (Eds.). Test and Analysis of Web Services. *Springer* 2007, chapter 9, pages 237-246.
3. L. Baresi, C. Ghezzi, and S. Guinea. Smart Monitors for Composited Services. In *Proc. of Int. Conf. on Service-Oriented Computing*, 2004.
4. W. Robinson. Monitoring Web Service Requirements. In *IEEE Int. Conference on Requirement Engineering*, 2003.
5. A. Sahai, V. Machiraju, A. van Morsel, and F. Casati. Automated SLA Monitoring for Web Services. In *Int. Workshop on Distributed Systems: Operations and Management*, 2002.
6. K. Mahbub and G. Spanoudakis. A Framework for Requirements Monitoring of Service Based Systems. In *Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 8493. ACM, 2004.
7. K. Mahbub and G. Spanoudakis. Run-Time Monitoring of Requirements for Systems Composed of Web-Services: Initial Implementation and Evaluation Experience. In *Int. Conf. on Web Services (ICWS)*, 2005.
8. A. Lazovik, M. Aiello, and M. P. Papazoglou. Associating Assertions with Business Processes and Monitoring their Execution. In *Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 94104. ACM, 2004.
9. L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *Proc. ICSOC 2005*, pages 269282. Springer, 2005.
10. L. Baresi and S. Guinea. A dynamic and reactive approach to the supervision of BPEL processes. In *Proc. ISEC 2008*, pages 3948. 2008.
11. Fabio Barbon, Paolo Traverso, Marco Pistore, Michele Trainotti: Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *Proc. ICWS 2006*:63-71.
12. A. K. Tripathy and M. R. Patra. An event based, non-intrusive monitoring framework for web service based systems. In *International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, 2010, pages 547–552. IEEE, 2010.
13. A. K. Tripathy and M. R. Patra. Service based system monitoring framework. In *International Journal of computer Information Systems and Industrial Management Applications: IJCISIM*, 3:924-931, 2011.
14. A. K. Tripathy and M. R. Patra. Modeling and monitoring sla for service based systems. In *Proceedings of the International Conference on Intelligent Semantic Web-Services and Applications: ISWSA*, 60-65, 2011.
15. A. K. Tripathy and M. R. Patra. Cost Effective, Requirement Oriented Web Service Composition and Adaptation. In *the International Journal on Recent Trends in Engineering & Technology: IJRTE*, Volume 05, No. 01, Page 95-98 Mar 2011. ACEEE USA.
16. A. K. Tripathy and M. R. Patra. Automating WS-Agreement Monitoring. In *International Journal of ACM Jordan: The research Bulletin of Jordan ACM Volume II (I)*, Page 99-113 *Jordan* 2011.
17. M R Patra, A. K. Tripathy and R. K. Das. Monitoring of Service Based e-Government Systems. In *the 5th International Conference on Theory and Practice of Electronic Governance: ICEGOV 2011* Estonia, Pages 353-354 ACM 2011.
18. A. K. Tripathy, M. R. Patra. MSL Based Service Based e-Governance Systems Monitoring. In *International Journal of ACM Jordan: The research Bulletin of Jordan ACM Volume II (II)*, Page 90-101 *Jordan* 2012.
19. A. K. Tripathy, M. R. Patra, S. K. Pradhan, R. Rafique, P. Mohanty and S. R. Dash. Graph Based Service Selection for Composition and Adaptation. In *International Journal of ACM Jordan: The research Bulletin of Jordan ACM Volume II (II)*, Page 83-89 *Jordan* 2012.