International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 13 (2015) pp 33336-33340

Web Service Message Contract Validation using Server Logs at Implementation level

M.Priyadharshini[1], T.Vimala[2], R.Baskaran[3], K.Manju Bharathi[4]

[1, 3] *Computer Science Department, Anna University, India*
mpriya1977@gmail.com, baaski@annauniv.edu
[2, 4] *Department of Information Science and Technology, Anna University, India*
vimalaraj11@gmail.com, kmanjubharathi@gmail.com

Abstract
Web Service request and response using SOAP messages is found to be popular in exhibiting a stateful behavior only on satisfying a set of constraints termed as "message contracts". The message contract includes the set of constraints referring to the data elements in the message invocations. The process of monitoring such message contracts enables a specific message sequences to be exchanged with clients. The Monitoring process done at runtime could be performed at client side as well as at the server side of Message invocation. The Server side monitoring involves validating the message invocations against the properties that are represented in form of Message Contract Language using temporal operators over data in the XML messages. The Service Logs maintained at the server side containing the prior service invocation traces are used for the monitoring process at server side. This process aims at preventing the erroneous messages being exchanged during transaction.

Keywords: Web Service, Server Side Monitoring, Message Contract Language, Server Log.

1 Introduction

A Functionality accessed as a service coined as "web service" forms an integeral part of commercial applications today. SOAP messages being a major means of communication to access the web services is sateless and hence could not persist the data in the messages for across various service invocations. There are set of properties that are to be validated as a part of requirements in addition to that of the WSDL contents involving the data present in the SOAP requests for valid service invocations.These properties are presented as Message Contract Language(MCL) using temporal operators especially formulated to monitor the message sequences.The Message contract validator does validation of the MCL for the corresponding message contracts defined by the service provider and agreed upon by the service requestor.The messages that are needed for validation are recorded in form of service logs maintanied at the server.The messages are needed for validation are generated from the service logs and are used by message contract validator to validate the properties enforced using MCL. This proposed work is aimed at implemening a runtime monitor restricting the invocations of erroraneous message sequences and hence leading to increase in the availability of services.The proposed architecture for Message contract validator is presented and discussed upon in the rest of the paper. The related works are presented along with the conclusion of the proposed work.

2 Proposed Architecture for Message Contract validation

The proposed architecture is implemented at the Web Server of the Service Requestor. The message contract validator being the significant component does two sub process one making entries in server logs during execution and other process performing MCL verification. Fig.1. presents the proposed architecture including message contract validator and its sub-processes.

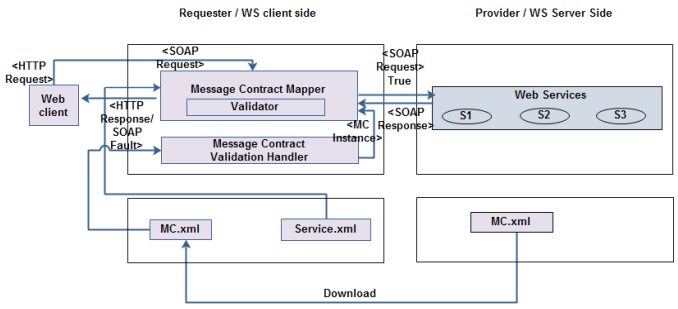

Fig. 1. Message Contract Validator with Server Execution Log

2.1 Message Contract Validator

Message contract validator or runtime verification is the software component that has an observer that monitors the service invocation and execution as well as check its conformity with a requirement specification, usually represented in form of WSDL (Web Service Description Language).Sometimes added up with constraints written in a temporal logic. Runtime verification could also be applied to automatic evaluation of test runs, where as proposed system does analysis of stored Server Execution Logs where the invocation and execution traces are stored. Message Contract Language (MCL) has been used for those attempts involving runtime verification. Formalization and monitoring is one possible way to ensure compliance of service invocations with message contracts.

Service Execution Log

At Web Server of the service requestor a log named Service Execution Log is created once the service is accessed, which in turn is used for the MCL verification of the Message Contracts; a sample of which is presented in the Table 1. The Server Log structure is as given in form of XML Schema Definition (XSD) as in Table 2.

Table 1. Service Execution Log Sample

```
<?xml version=”1.0” encoding=”UTF-8”?>
<sl:slog xmlns:sl=”http://acmonitor.org/slog” provider=”UIA”>
<sl:services >
<sl:service entryDate=”2014-11-12”>
<sl:servicename>”InsuranceService”</sl:servicename >
<sl:operation>validatePolicy</ sl:operation>
<sl:parameter>Pno</sl:parameter>
<sl:access>
<sl:value>26559</sl:value>
<sl:time>11:30:55</sl:time>
<sl:access>
</sl:service>
<sl:service entryDate=”2014-11-12”>
<sl:servicename>”InsuranceService”</sl:servicename>
<sl:operation>claimSubmission</ sl:operation>
<sl:parameter>claimID</sl:parameter>
<sl:access>
<sl:value>2807</sl:value>
<sl:time>11:45:00</sl:time>
<sl:access>
</sl:service>
</sl:services>
</sl:slog>
```

Table 2. XSD for Service Execution Log

```
<?xml version=”1.0” encoding=”UTF-8”?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns=”http://acmonitor.org/slog.xsd”
targetNamespace=”http://acmonitor.org/slog.xsd”
elementFormDefault="qualified">
<xsd:element name="slog" type="tns:slogType"/>
<xsd:complexType name="slogType">
<xsd:sequence>
<xsd:element name="services" type="tns:servicesType" minOccurs="1"/>
</xsd:sequence>
<xsd:attribute name="provider" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="servicesType">
<xsd:sequence>
<xsd:element name="service" type="tns:serviceType" minOccurs="1"/>
</xsd:sequence>
<xsd:attribute name="entryDate" type="xsd:date"/>
</xsd:complexType>
<xsd:complexType name="serviceType">
<xsd:sequence>
<xsd:element name="servicename" type="tns:serviceType"
minOccurs="1"/>
<xsd:element name="operation" type="xsd:string"/>
<xsd:element name="parameter" type="xsd:string"/>
<xsd:element name="access" type="accessType"/>
</xsd:sequence>
<xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="accessType">
<xsd:sequence>
<xsd:element name="value" type="xsd:string/>
<xsd:element name="time" type="xsd:time"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Message Contract Language

An XML file representing the Temporal Logic is used to represent the contract is Message Contract Language(MCL) . When developing applications LTL has achieved a prominent role in the formal specification and verification of concurrent reactive systems. Various useful concepts could be formally, and concisely, specified using temporal logics. MCL are formed using the basic operators of TL.

□ :always ◇ :sometime or eventually ○ :next time or next step

U :until

In the proposed system property of message sequences are represented inform of MCL. The properties generally represented using LTL-FO$^+$ are Safety properties, Liveliness properties and Fairness properties where as for the proposed system; MCL is defined with Invocation properties. The Properties are defined based on Hospital Information System Client over Insurance Provider Services. Runtime Properties given below are as per the message contracts and is represented using Temporal Logic as given in Table 3.

- Runtime Property 1: The invocation of Claim Status Verification happens till Claim Sanctioned state achieved.
- Runtime Property 2: The Claim Submission as next step of Policy Verification.
- Runtime Property 3: The Claim Status Verification always after Claim Submission.

Table 3. MCL Properties Representation

Service and States in the invocation process are given as: o Claim Submission(CS) o ClaimStatusVerfication(CSV) o ClaimSanctioned(CSN) o PolicyVerification(PV)
Invocation Property "invocation will happen" o Runtime Property1: CSV *U* CSN o RuntimeProperty2: ◌ PV □ CS o RuntimeProperty 3: □ CS □ CSV

Message Contract Validation is done using Message Contract Mapper, which creates and validates message sequences against Message Contract. The Message Contract Validation Handler formulates the instance of Message Contract as per the message been invoked and used by Message Contract Mapper for validation as shown in the Figure 2.

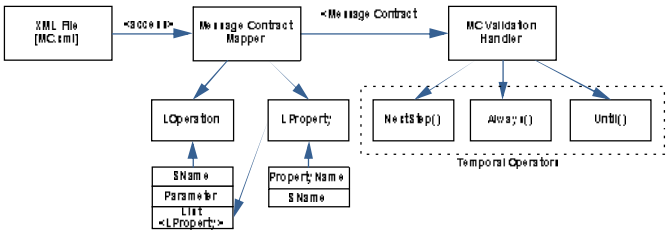

Fig. 2. Message Contract Validation Model

3 System Experimentation and Results

The proposed architecture of message contract validator is implemented considering Hospital and Insurance Provider with the Message contracts represented as MCL. Runtime Properties are validated for various lengths of message traces and it is found to show a valuable difference from that of the BeepBeep Monitors usually implemented at client side for the above purpose. The number of requests sent to the server for execution is reduced highly and in particular for the services involving more number of message contract entries. The experimentation is performed using 25 different invocations of the services and the results are formulated as graph using the average values.

The Validate Policy Service does not have Message Contract Validation that need to be validated before invocation. Hence for every invocation, logger need not be checked for the values but checks the database for validating the policy. The processing time taken for validating policy in both cases (true and false) remain more or less the same where the time to make entry in the log is only applicable for true condition as shown in Figure 3.1a.

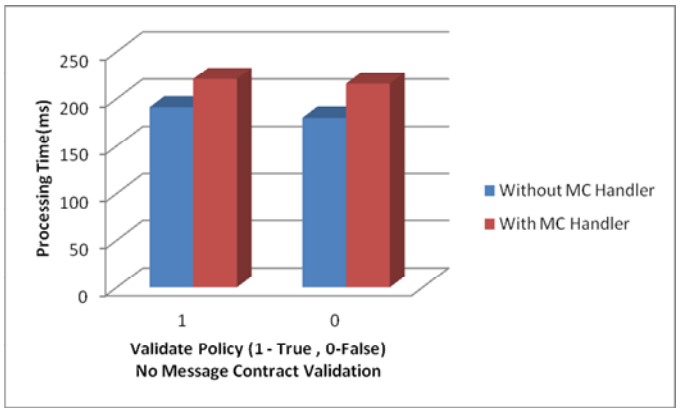

Fig.3.1a. Validate Policy Service Invocation

The processing time taken for claim submission service in true case(where policy validation is done as previous step) is found to remain more or less the same whereas claim submission without validating the policy processing time is more without MC handler since it has to be checked at the server side as shown in Figure 3.1b.

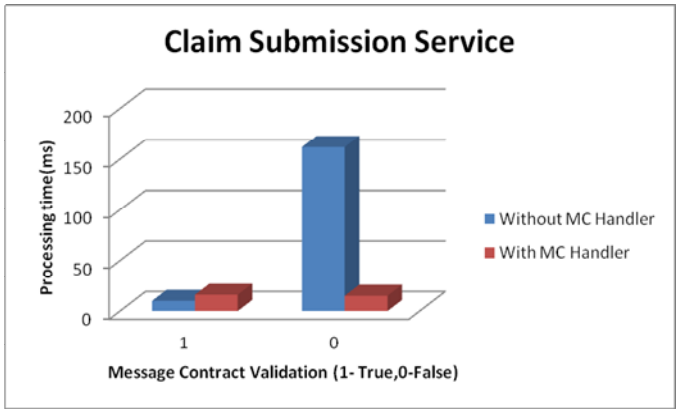

Fig.3.1b. Claim Submission Service Invocation

The processing time taken for claim status verfication service in true case, where claim is submitted and sanctioned is more without MC handler since it has to be checked at the server side, also same in case of claim not submitted. The processing time for claim status verification where claim is submitted and not sanctioned without MC Handler is more since it has time to

access Log as well as to verify in the database as shown in Figure 3.1c

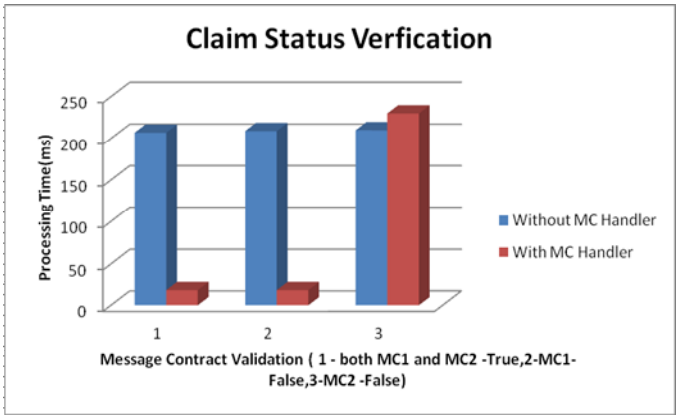

Fig.3.1c. Claim Status Verification Invocation

Increase in Log size also doest affect the processing time much due to the Message Contract Mapper that extracts only the entries needed for that particular service as shown in Fig 3.2 a,3.2b,Fig3.2c where the graph for processing time taken by three services represented with different Log Sizes.

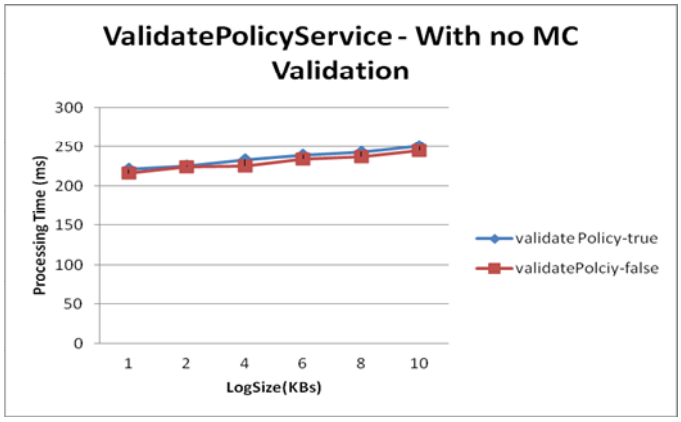

Fig.3.2a. PolicyValidation Service Invocation with and without MC Handler

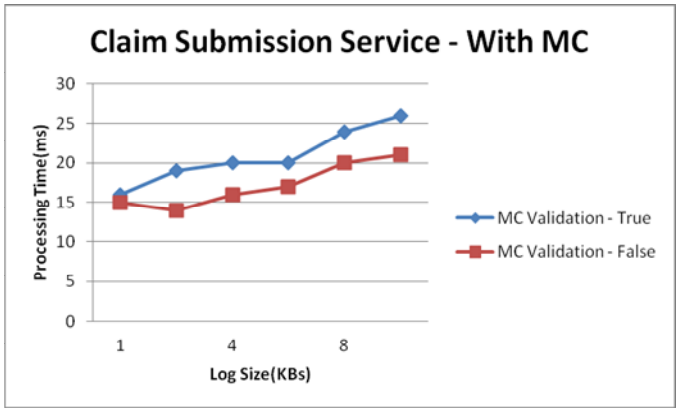

Fig.3.2b. Claim Submission Service Invocation with and without MC Handler

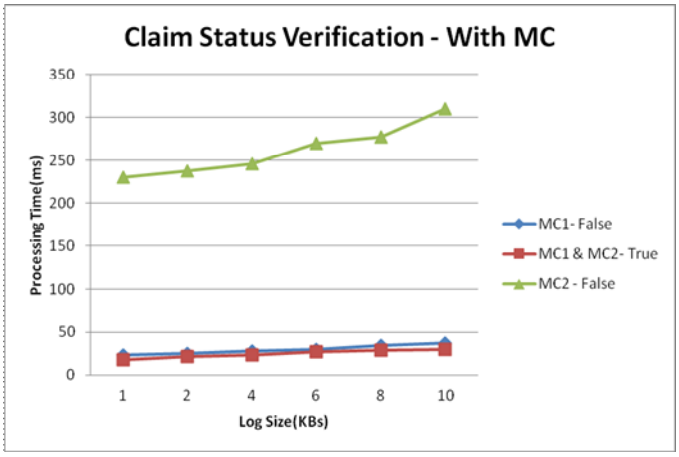

Fig.3.2c. Claim Status Verification Service Invocation with and without MC Handler

4 Related works

Havelund, K., [1] proposed runtime verification during the last decade seen a multitude of systems for monitoring event sequences (traces) emitted by a running system. Halle S et al., [2] proposed an increasing number of popular SOAP web services that exhibit a stateful behavior, where a successful interaction is determined as much by the correct format of messages as by the sequence in which they are exchanged with a client. Havelund, K., [3] proposed a form of automaton, referred to as data automata, suited for monitoring sequences of data-carrying events, for example emitted by an executing software system. Barringer, H., et al., [4] proposed event logs that have been widely used over the last three decades to analyze the failure behavior of a variety of systems. Barringer, H., et al., [5] proposed trace contract, an API for trace analysis, implemented in the SCALA programming language.

Havelund, K., [6] proposed Runtime verification (RV) that consists of checking execution traces against user-provided formalized specifications. Barringer, H., et al., [7] proposed a rule-based framework for defining and implementing finite trace monitoring logics, including future and past time temporal logic, extended regular expressions, real-time logics, interval logics, forms of quantified temporal logics, and so on. Stolz, V., et al., [8] proposed model checking techniques to debug Concurrent Haskell programs. LTL formulas specifying assertions or other properties are verified at runtime. Ghezzi, C., et al.,[9] et al investigated continuous monitoring of SOAs, with particular emphasis on web services. Halle, S., [10] has defined the notion of causality for interface contracts expressed in a first-order extension of Linear Temporal Logic. He presented CTL-FO+, an extension over Computation Tree Logic that includes first-order quantification on message content in addition to temporal operators and also proposed how CTL-FO+ is adequate for expressing data-aware constraints, give a sound and complete model checking algorithm for CTL-FO+, and establish its complexity to be PSPACE-complete. Simmonds, J., et al [11] proposed runtime monitoring of conversations between partners as a means of checking behavioral correctness of the entire web service system. Specifically, a subset of UML 2.0 Sequence Diagrams was identified as a property

specification language and show that it is sufficiently expressive for capturing safety properties.

5 Conclusion

The monitoring process in the proposed work is based on the manual identification of properties and devising them into MCL and also found to utilize less cost in identifying the incorrect message sequence increasing the availability of services. Further the monitoring could be done based on the automated extraction of constraints from a service's source code or sample execution traces that needs the service providers to systematically document them in some machine-readable form. The mapping between MCL and UML models could be devised, in order to ease the specification of runtime constraints on web services.

References

1. Havelund, K., 2014 "Data Automata in Scala", Theoretical Aspects of Software Engineering Conference (TASE), pp. 1-9.
2. Hallé, S., and Villemaire, R., 2012 "Runtime Enforcement of Web Service Message Contracts with Data", IEEE Transactions on Services Computing, vol. 5, no. 2, pp. 192-206.
3. Havelund, K., 2014 "Monitoring with Data Automata", In Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications, Springer Berlin Heidelberg, pp. 254-273.
4. Barringer, H., Groce, A., Havelund, K., and Smith, M., 2010 **"**Formal Analysis of Log Files**,"** Journal of Aerospace Computing, Information, and Communication, vol. 7, no. 11, pp. 365–390.
5. Barringer, H., and Havelund, K., 2011 **"**TraceContract: A Scala DSL for Trace Analysis**,"** in 17th International Symposium on Formal Methods (FM'11), Springer, vol. 6664, pp. 57–72.
6. Havelund, K., 2014 "Rule-Based Runtime Verification Revisited,"Software Tools for Technology Transfer (STTT), published online..
7. Barringer, H., Goldberg, A., Havelund, K., and Sen, K., 2004 **"**Rulebased Runtime Verification**,"** in VMCAI, ser. LNCS, Springer, vol. 2937, pp. 44–57.
8. Stolz, V., and Huch, F., 2005 "Runtime Verification of Concurrent Haskell Programs," In Proc. of the 4th Int. Workshop on Runtime Verification (RV'04), ser. ENTCS, Elsevier, vol. 113, pp. 201–216.
9. Ghezzi, C., and Guinea, S., 2007 "Run-time Monitoring in Service-Oriented Architectures", In Test and analysis of web services, Springer Berlin Heidelberg, pp. 237-264.
10. Halle, S., 2011"Causality in Message-Based Contract Violations: A Temporal Logic "Whodunit", 15th IEEE International Enterprise Distributed Object Computing Conference, pp. 171-180 .
11. Simmonds, J., Gan, Y., Chechik, M., Nejati, S., O'Farrell, B., Litani E, and Waterhouse, J., 2009 "Runtime Monitoring of Web Service Conversations", IEEE Transactions on Services Computing, vol. 2, no. 3, pp. 223-244.