Framework for Analyzing SOAP Messages in Web Service Environments

Martin Kuehnhausen Graduate Student Member, IEEE and Victor S. Frost, Fellow, IEEE

Abstract— **Service Oriented Architectures can be quite complex which makes managing and monitoring them hard. Message exchanges as well as complete message flows are important when it comes to analyzing the performance of systems and identifying problems. The use of SOAP as the common message format for web services enables interoperability and extensibility. Furthermore it can be used for logging and analysis because its components follow web service specifications. We present a framework that allows for logging, analyzing and visualizing these messages across a distributed system.**

The proposed framework consists of a flexible logging module that captures incoming and outgoing messages of a web service. The log parsing library provides various methods for normalizing, correlating across geographically distributed sites and analyzing messages. The visualization tool is able to display relationships between the web services as well as message flows.

In utilizing this adaptable framework for analyzing SOAP messages it is possible to overcome the challenges of complexity and disparity that monitoring and management approaches face in web service environments

Index Terms—**Logging, Log analysis systems, Visualization, Transport protocols, Data communication, Software engineering**

I. INTRODUCTION

THE use of Service Oriented Architectures (SOA) in application systems is widespread. The idea is to implement specific functionality in web services that communicate with each other using standardized interfaces. Message exchanges in general use the flexible SOAP message format [1]. This has a number of advantages as for instance message routing and security are available as extensions to it. However, in many cases and especially in geographically distributed systems as shown in Figure 1, there exists no simple way to analyze and visualize the message flow through the entire system as well as measuring performance of components in order to identify potential bottlenecks.

This paper describes a framework that is able to analyze a variety of timing measurements such as message transmission and service processing times. Furthermore it correlates messages which can then be used to analyze message flows. Finally a graphical user interface is presented that allows the visualization of individual system components and their message interactions.

M. Kuehnhausen and V. S. Frost are with the Information and Telecommunication Technology Center, The University of Kansas, Lawrence, KS, 66045, USA; e-mail: mkuehnha@ittc.ku.edu and frost@ittc.ku.edu

This work was supported in part by Oak Ridge National Laboratory (ORNL)—Award Number 4000043403. This material is also partially based upon work supported while V. S. Frost was serving at the National Science Foundation.

II. PROBLEM AREA

Web service environments consist of disparate systems and technologies. In order to efficiently monitor and analyze message flows the following areas need to be addressed.

A. Logging

Chuvakin et al. [2] explain that logs often contain "valuable information about systems, networks, and applications". In particular logging is important when it comes to auditing because laws often mandate so-called audit trails, for example in the health industry and the banking sector. Another important observation that Chuvakin et al. make is that logging in distributed systems needs to be addressed differently as web services "are by their nature distributed across multiple systems, disparate technologies and policies, and even organizational domains". This means that logging needs to be addressed on a global basis either with a centralized logging server or with a common logging format that is used by each individual service.

B. Analysis

Apart from auditing purposes log file analysis is often used to study or detect failures in systems. Furthermore usage and bandwidth monitoring in distributed systems is important for load balancing in order to keep costs down. Lim et al. [3] outline particular aspects that need to be considered in logging systems, in their case an enterprise telephony system, that allow efficient and effective analysis.

They also note that one of the problems is that data in logs is often unstructured and it is therefore hard to automatically discover patterns. However, this is often solved by log preprocessing such as message normalization and clustering. After cleaning the logs data mining approaches such as finding frequent item sets, frequency analysis and anomaly detection may be applied. Furthermore data in logs can be correlated and dependencies of messages can lead to message flow analysis. In terms of performance analysis measurements such as processing and transmission times are important.

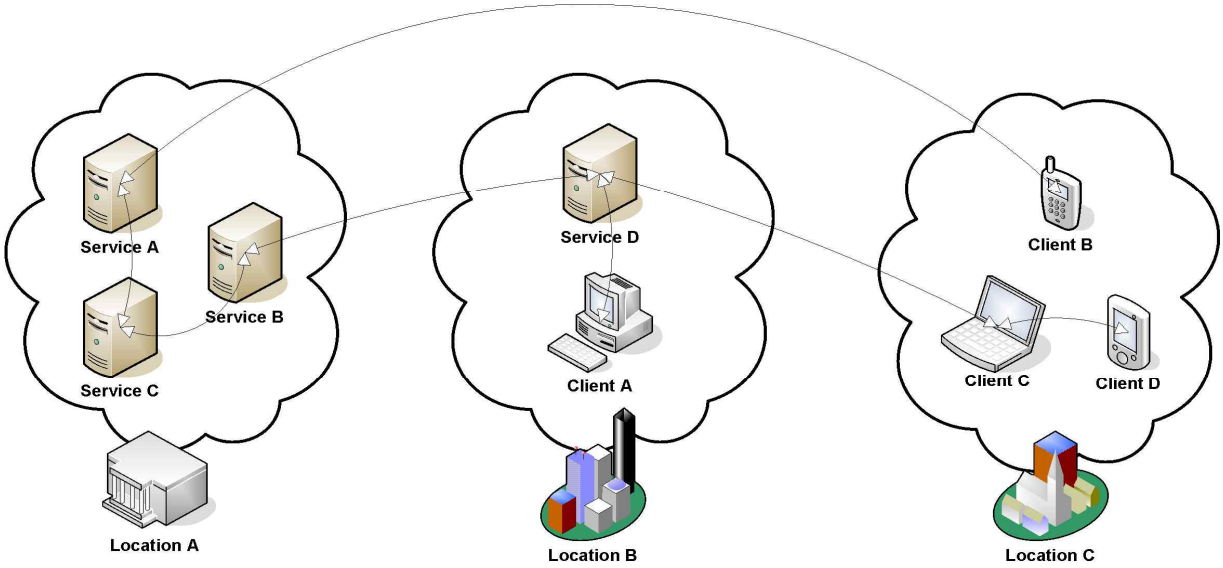


Fig. 1. Geographically distributed services

C. *Visualization*

Since there is often an abundance of data present it is not necessarily easy to grasp important aspects or quickly analyze data. This is especially true regarding message flows. While time measurements and message statistics can easily be represented using a table format, sequences and dependencies need more complex structures such as graphs and trees.

III. Related Work

A. *Logging systems*

Cinque et al. [4] propose an system that is based on specific logging rules that allow "effective dependability evaluation of complex systems". By defining a common set of rules such as service start, service end, entity (resource) interaction start and end an observer is able to follow certain event flows. Given estimated computation duration and considering potential timeouts the system generates alerts whenever it detects a problem in an event flow.

Vaarandi [5] introduces two compact log file analysis tools called Simple Logfile Clustering Tool (SLCT) and LogHound which apply two of the most common approaches to log analysis. SLCT uses a density based clustering method that is able to detect outliers while LogHound "employs a frequent itemset mining algorithm".

Makanju et al. [6] provide an overview of "network information visualization tools" as well as propose their own version called LogView. They are using a variety of techniques such as plots and treemaps to visualize network traffic, intrusion detection and application logs.

Logging web sessions that track specific users generates a lot of data. In contrast to other logging systems the focus here lies on analyzing flows and user behaviors. Session Viewer [7] by Lam et al. is a visual tool that consists of various panels for data aggregation, cluster and flows analysis. This allows a statistical overview as well as detailed analysis of so-called session logs.

An approach that applies directly to web services is proposed by Simmonds et al. [8]. They describe a runtime monitoring system based on a subset of UML 2.0 Sequence Diagrams which are used for checking conditions and messages exchanges. After the constraints are correctly defined in sequence diagrams they are handed over to a "monitoring manager" that receives events from a "message manager" and checks those against the events and flows specified. Simmonds et al. also discuss various related work in terms of the difference in "offline" and "online" monitoring as well as "global" and "local" property checking. Their proposed system is able to efficiently check the correctness of events and flows at runtime.

Service Oriented Architectures are inherently based on web service specifications. These specifications clearly define standard properties and functionality of web services that implement a particular specification. Within a logging system this leads to the ability of extracting information according to these standards independent of how they are implemented and in a flexible and extensible manner. This is not necessarily true in most logging system that depend on a particular log format or are specifically built for one particular purpose. Whenever more information needs to be captured the log format needs to be adapted in these systems while using a web service based approach this is not necessary.

B. *Web Service Specifications*

In web service environments there are two common types of message exchanges. The first is a one-way message transfer from a service to another service or client which is often used in notification scenarios. The second is a common two-way exchange in which a request message is sent to a particular service, the service then fulfills the request and sends out a response accordingly. Apart from these basic message exchange patterns it is possible to set up a subscription system and then have a web service automatically deliver messages to a client.

It is important to note that the messages themselves can be encrypted in order to protect private information which can pose issues when trying to analyze message flows. In the following the web service specifications related to message exchanges, subscriptions and security are discussed.

1) WS-Addressing:

The *WS-Addressing* core specification [9] and its SOAP binding [10] defines how message propagation can be achieved using the SOAP message format. Usually the transport of messages is handled by the underlying transport protocol but there are several advantages of storing this transport information as part of the header in the actual SOAP message. For example, it allows the routing of messages across different protocols and management of individual flows and processes within web services.

WS-Addressing uses so-called *EndPointReferences* which are a collection of a specific address, reference parameters and associated metadata that further describe its policies and capabilities. The *Addressing Header* header fields defined by the specification are the following:

- *To* which represents the destination of the message
- *From* contains the source, a so-called *EndPointReference*
- *ReplyTo* specifies that in case of a response, a message is supposed to be sent to this *EndPointReference,* which might be different from the *From* field
- *FaultTo* defines the *EndPointReference* for the fault message in the case of an error
- *Action* identifies the purpose of the message, in particular the web service operation, and is the only required field
- *MessageID* uniquely identifies every message
- *RelatesTo* references the *MessageID* of the request message in request-response message exchanges; the relationship can also be specified explicitly by defining a so-called *RelationShipType*

2) WS-Eventing:

In order to allow for subscriptions to web services, the *WS-Eventing* specification [11] has been defined. It describes the process of establishing subscriptions as well as how the subsequent publications are delivered to the subscribers. The specification relies on *WS-Addressing* for the routing of messages. The two main components of a subscription in this specification are the *Subscribe* and the *SubscribeResponse* message. After subscriptions have been created, publications will be sent out accordingly.

a) Subscribe

The client that wants to subscribe to a particular web service needs to define the following:

- The *Action* field of the *WS-Addressing* header is set to http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
- *ReplyTo* is the *EndPointReference* that receives the response to this subscription request
- A *MessageID* that uniquely distinguishes multiple requests from the same source
- *EndTo* defines an *EndPointReference* that is used when the subscription ends unexpectedly
- *Delivery* contains the *EndPointReferences* that are to receive the publications
- An *Expires* field that defines the expiration time of the subscription
- *Filter* that by default defines an *XPath* expression as the *Dialect*, but could be any form of expression that is applied to potential publications in order to filter them

b) SubscribeResponse

The response to a subscription request is generated by the so-called *subscription manager*. It sends back a message with these fields:

- The *Action* field of the *WS-Addressing* header is set to http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
- *RelatesTo* specifies the subscription request that this is a response to
- *SubscriptionManager* that contains its own *Address* and the unique Identifier for the subscription
- An *Expires* field that defines the expiration time of the subscription

The *WS-Eventing* specification also offers message constructs for the renewal, status retrieval and unsubscribing of subscriptions. Additionally a so-called *subscription end* message is automatically generated by the service that publishes information in order to notify subscribers of errors or other reasons for it being unable to continue the subscription.

It has to be noted that without additional specifications like *WS-ReliableMessaging* the delivery of publications is based purely on best effort and is not guaranteed.

3) WS-Security

The *WS-Security* specification [12] deals with the many features needed to achieve so-called *end-to-end* message security. This provides security throughout message routing and overcomes the limitations of so-called *point-to-point transport layer security* such as HTTPS. Furthermore, the specification aims to provide support for a variety security token formats, trust domains, signature formats and encryption technologies.

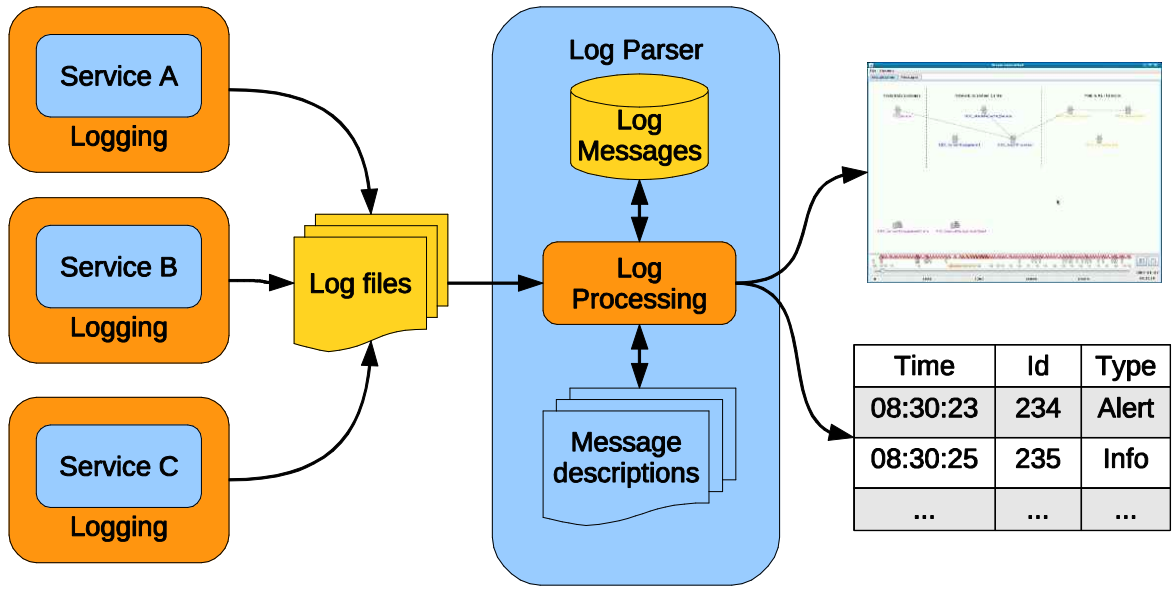


Fig. 2. Framework overview: logging, log parser and visualization (from left to right)

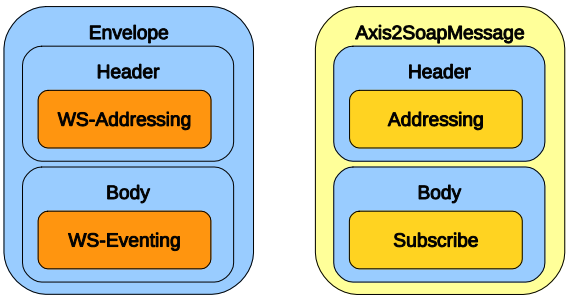


Fig. 3. SOAP message (left) to Log parser classes (right) comparison

IV. Proposed Solution

The framework developed here allows the capture and analysis of SOAP messages in one and two-way communications as well as subscriptions. Logging these messages that are based on web service specifications and extracting information from them overcomes common problems previous approaches faced such as the necessity of a common log format across different platforms and the inability to extend the system later to log more information without breaking existing functionality.

The individual components are shown in Figure 2 and are explained in the following sections. Note that the logging part is positioned in between the message sender/receiver and the security layer which makes it possible to capture message information without compromising security functionality in the web service environment.

A. Logging

The logging component that was developed as part of this framework was initially designed to support a particular system, the Transportation Security SensorNet (TSSN) [13]. However, it can easily adapted to other systems. The TSSN is based on the Axis2 web service stack which uses modules for implementing web service specifications. While modules for WS-Addressing and WS-Eventing come as part of Axis2, a module that implements the desired logging functionality had to be developed separately.

1) Logging Module

The *logging* module as described in the following provides extensive logging capabilities to the *web services*. It was *engaged* during development and testing of the entire TSSN system since it logs all messages that are sent and received. In addition, it also writes the raw contents of the SOAP messages that are sent and received into log files. In particular the following information is captured:

- *Time* when the message was sent or received
- *Service* which is used
- *Operation* that is being executed
- *Direction* of the message, which can be either incoming or outgoing. Note that there are special directions that
- deal with incoming and outgoing faults.
- *From* address of the message
- *Reply to* address that may differ from the *From* address
- *To* address of the message
- *Schema* element that is being "transported" as part of the operation containing the request parameters or the response elements
- *Size* of the message in bytes
- *Message* which represents the entire SOAP message in a readable form

In terms of analyzing the TSSN and its performance the

TABLE I
XPATH EXPRESSIONS FOR WS-ADDRESSING

XPath expression	Method equivalent
//To/text()	getTo()
//ReplyTo//Address/text()	getReplyTo()
//From/Address/text()	getFrom()
//MessageID/text()	getMessageId()
//RelatesTo/text()	getRelatesTo()
//Action/text()	getAction()

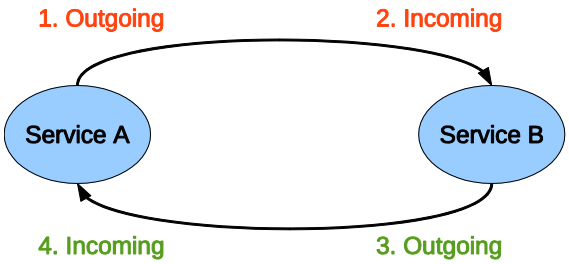


Fig. 4. Two transmit-receive pairs (red and green)

logging module was engaged in all services. More information on the findings can be found in V.

2) Addressing:

An implementation of the *WS-Addressing* specification as described in III-B1 comes as part of the *addressing* module in the Axis2 core. It fully supports all components of the standard and its *ReplyTo* and *RelatesTo* fields are used among other things to allow for asynchronous communication in the TSSN.

3) Savan:

The *Savan* module enables web services and clients in Axis2 to make use of various forms of subscription mechanisms as defined by the *WS-Eventing* specification (see III-B2).

B. Log Parser

The log parser enables parsing, processing and merging of log files. It transforms the raw SOAP messages into Java elements that can then be filtered and analyzed.

1) Abstraction Layer Model:

Since SOAP is essentially XML, information from the so-called log messages can retrieved using *XPath* [14] *path expressions*. For this purpose the *log parser* provides an object *abstraction layer model* that corresponds to the specific parts in the SOAP message.

An example mapping is shown in Figure 3. It displays the structure of the original SOAP message (for more information on the individual SOAP messages see III-B) on the left and the equivalent *log parser* objects on the right. Note that the corresponding objects highlighted in yellow are actual classes while the *Header* and *Body* are not abstracted separately.

The log parser objects would then provide access to their properties using *XPath expressions*. In this case they correspond to their respective *web service* specifications but they could also be defined according to the XML schema definitions of any other element. For example, for the *WS-Addressing* (see III-B1) equivalent object the path

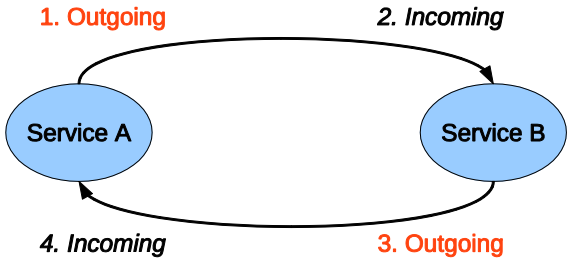


Fig. 5. A message couple (red)

expressions in Table I are used:

This mapping process is easily defined because it corresponds to the web service specifications and allows for an in-depth analysis of the messages that are sent and received.

2) Message Types:

Since the *logging* module is enabled on both ends of a message exchange, the *log parser* is able to correlate messages. In order to do this it makes use of the so-called *message id* that is provided by the *WS-Addressing* specification. It has to be noted that a requirement for the following analysis is that the times on both ends of the message transfer are synchronized. Within the TSSN system this is done using NTP [15] but it is also possible with GPS.

Without this assumption the computed times are questionable and represent an estimation at best. The following two types of message associations are present in the log files:

a) Transmit-Receive Pair:

Whenever a message is sent out by a particular client or service it is captured by the logging module. The receiving service logs the message as well but as an incoming message. The content of the message is essentially the same which can also be seen by the fact that they have the same message id. The outgoing and the incoming message are combined and form what is called a transmit-receive pair.

This allows us to compute the message transfer or so-called transmit time which describes how long it takes to transmit the message from one entity to another using the following equations:

$$transmitTime_1 = time_{2.\text{Incoming}} - time_{1.\text{Outgoing}} \quad (1)$$

$$transmitTime_2 = time_{4.\text{Incoming}} - time_{3.\text{Outgoing}} \quad (2)$$

As shown in Figure 4 the *log parser* automatically detects the *transmit-receive* pairs and stores them in a particular list for further analysis.

b) Message Couple:

The most common message exchange pattern is the *In-Out* pattern. It defines *request-response* based message transfers which the *log parser* calls *message couples*. A single message

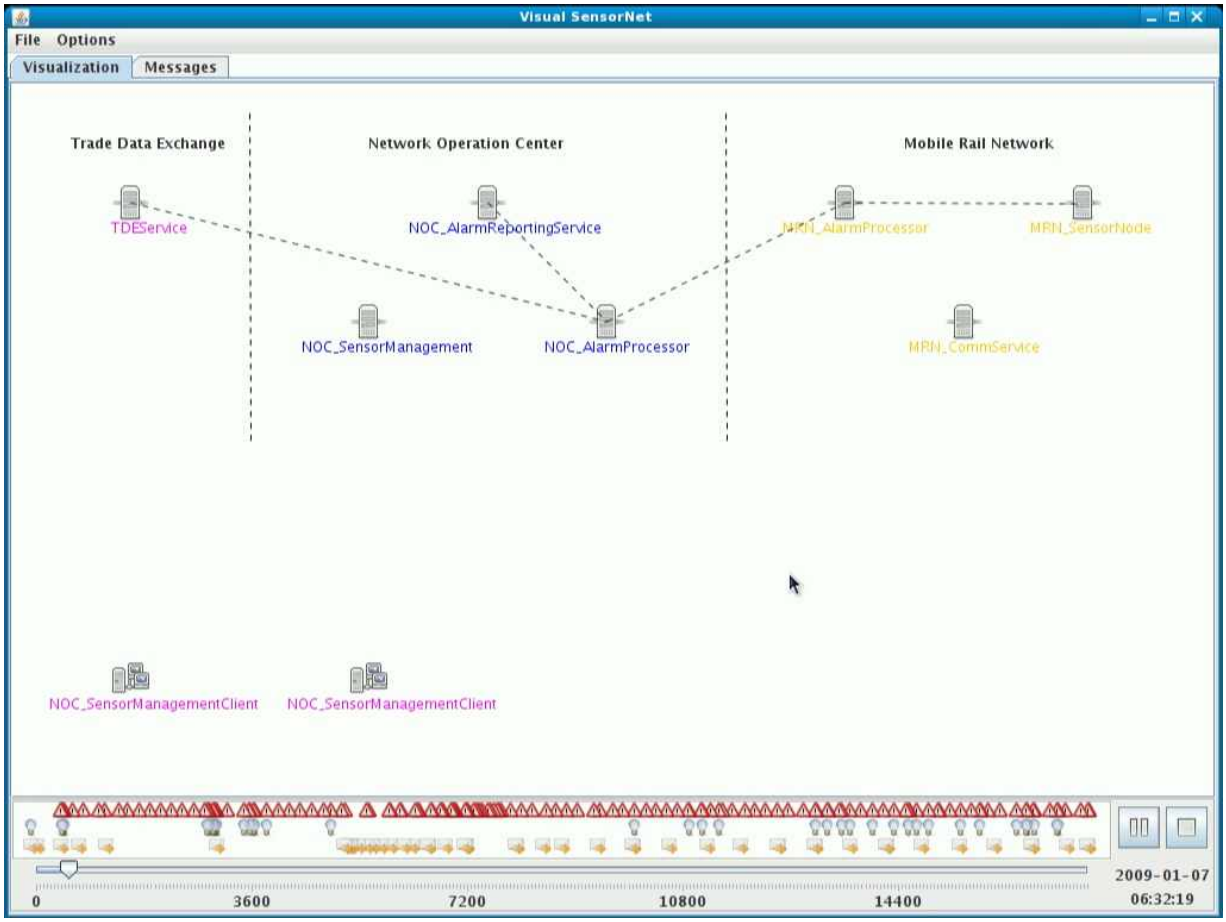
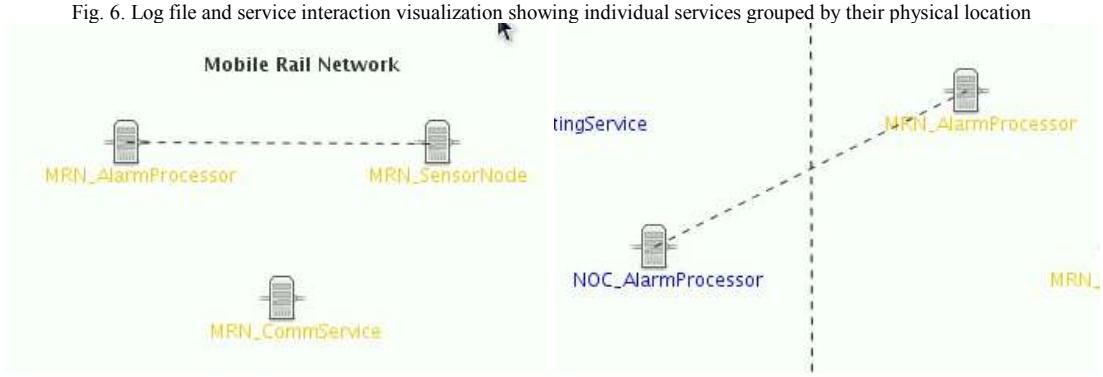
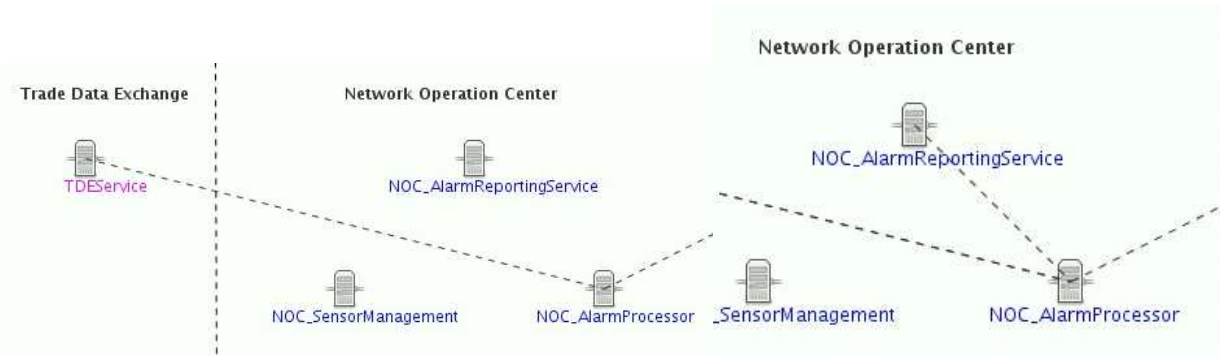


Fig. 6. Log file and service interaction visualization showing individual services grouped by their physical location





couple consists of two messages, the outgoing request and the outgoing response on the receiving entity, which is shown in Figure 5. They can be correlated using the *WS-Addressing* specification. The request will carry a *message id* and the response a so-called *relatesTo id* in addition to its own unique *message id*.

Note that a *message couple* can also be seen as a combination of two *transmit-receive* pairs. This relationship is extremely useful in computing measures such as round trip and processing times:

$$roundTripTime = time_{4.\text{Incoming}} - time_{1.\text{Outgoing}} \quad (3)$$

$$processingTime = time_{3.\text{Outgoing}} - time_{2.\text{Incoming}} \quad (4)$$

The *log parser* provides functionality to associate messages and analyze complete end-to-end message flows. More details on the performance measurements and test results can be found in V.

C. Visualization

In order to understand the message flows better without needing too much of a technical background, a visualization tool was developed. It makes use of the *log parser* to display services, clients and messages that are present in log files.

The user is able to load and merge log files to create a visualization of services and clients as shown in Figure 6. Lines connecting individual services appear as communication occurs. The layout of these services is defined according to their membership in a particular *service cloud*. Furthermore, any point in time that is part of the log files can be "jumped to" using the time line. It displays significant events in the log files:

- *Alarms, alerts* and *sensor node events* with a warning sign
- *Requests* such as location retrieval with a light bulb sign
- *Control messages* such as *start monitoring* with a message sign

The scenario that was captured by the log files can also be played back in portions or in its entirety. Using the visualization tool, it is therefore possible to analyze service interactions and message flows conveniently. An example message flow is shown in Figure 7.

V. RESULTS

The framework described here enabled analysis of field trials of the TSSN. The Transportation Security SensorNet [13] uses a Service Oriented Architecture approach for monitoring cargo in motion along trusted corridors. The complete system provides a web services based sensor management and event notification infrastructure that is built using open standards and specifications. Particular functionality within the system has been implemented in web services that provide interfaces according to their respective web service specifications. This web services based implementation allows for platform and programming language independence and offers compatibility and interoperability with other systems.

Furthermore, unlike existing proprietary implementations the TSSN allows sensor networks to be utilized in a standardized and open way through web services. Sensor networks and their particular communication models led to the implementation of asynchronous message transports in SOA and are supported by the TSSN.

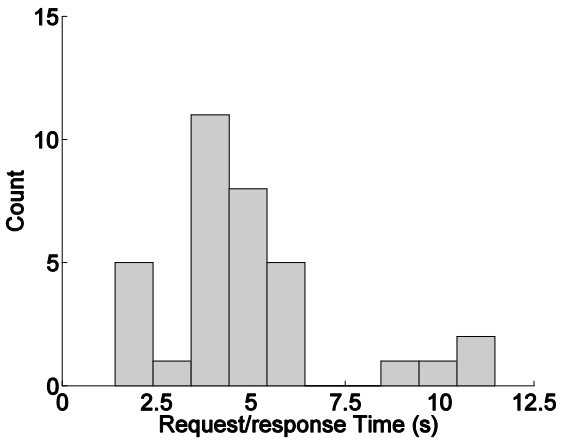


Fig. 8. Request performance from [16]

An in-depth analysis of the real world scenarios that were performed to test the TSSN is given by [16]. For the tests the *Trade Data Exchange* was deployed in Overland Park, the *Virtual Network Operation Center* at the *University of Kansas* in Lawrence and the *Mobile Rail Network* either on a truck or on a train. Note that in both cases the communication between the *Mobile Rail Network* and the *Virtual Operation Center* was established using a GSM modem. The main findings are as follows:

A. Short Haul Rail Trial

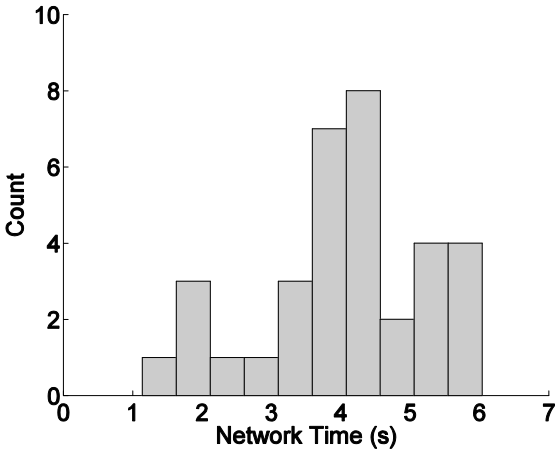
This more advanced scenario was performed by deploying the *Mobile Rail Network* on a locomotive of a train along with sensors attached to containers for it to monitor. The train traveled approximately 35 kilometers during the trip, from a *rail intermodal facility* to a rail yard.

The system faced some of the same issues as during the truck trials such as loss of GPS, GSM and sensor communication. The data that was collected however shows that again the *Transportation Security SensorNet* was able to deal with them and send out alarm notifications reliably. The log files were analyzed using the *log parser* and led to the following:

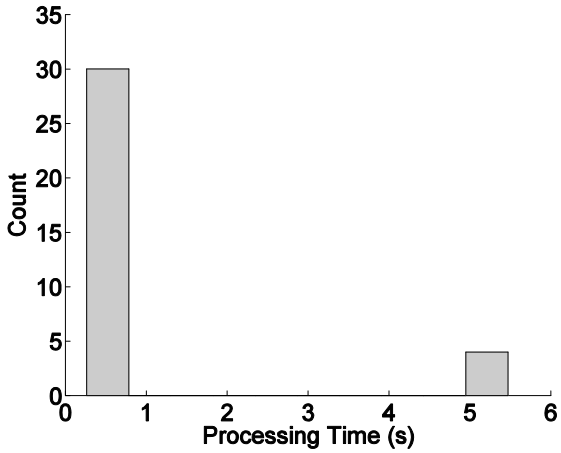
1) Message Counts:

During the *short haul rail trial* the *Sensor Node* reported 546 alerts[1] to the *Alarm Processor*. After filtering 131 alarms were sent to the *Alarm Processor* at the *Virtual Network Operation Center*. For 63 of them, shipment information was queried from the *Trade Data Exchange* and 33 were stored as so-called *validated alarms*. All of the 131 alarms that the *Alarm Processor* received were sent out to *Alarm Reporting service* which notified the according contacts via SMS and email. There were also 30 inquiries from the TDE for the location of the *Mobile Rail Network*.

[1] Alerts were generated manually by opening and closing an electronic seal in the locomotive and automatically by things such as losing a GPS fix



(a) Network transmission performance from [16]



(b) Processing performance from [16]

Fig. 9. Network transmission and processing comparison from [16]

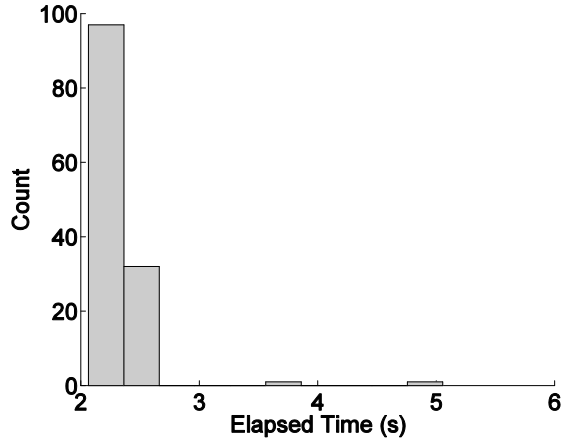


Fig. 10. System alarm notification performance from [16]

2) Message Sizes:

Looking at the communication between the *Virtual Network Operation Center* and the *Mobile Rail Network* one can notice the following pattern. So-called *control messages* such as *startMonitoring* or *getLocation* are always initiated at the *Virtual Network Operation Center*. Since these messages usually transmit only a small functional request, the average message size is around 690 bytes. On the other hand, Alarms are always sent from the *Mobile Rail Network* and contain of a lot of valuable information. Hence the average message size is about 1420 bytes.

a) Request Performance:

As shown in Figure 8, the time it took for messages from the *Virtual Network Operation Center* (*Sensor Management*) to send requests to the *Mobile Rail Network* (either *Sensor Node* or *Alarm Processor*) and receive a response was about 4.4 seconds on average. The fastest request was answered in 0.9 seconds while the slowest took about 11 seconds.

Overall these numbers meet the expectations of the transportation industry. Performing a location inquiry given an average train speed of 30 km/h and 60 seconds to retrieve the location, the actual position and the reported one may differ by as much as 500 meters. However, the *Transportation Security SensorNet* provides location information in less than 5 seconds resulting in a maximum difference of just 41.7 meters.

The bottleneck here is the message transmit time as defined in Equation 1. As shown in Figure 9, processing on the *Sensor Node* took only 0.6 seconds on average whereas about 85% of the time is spent on message transmission. This percentage is likely to increase when switching to satellite communication instead of communicating with the GSM modem which was used in the trials.

b) Alarm Notification Performance:

Because of the problems with the clock drift, the measured times for messages coming from the Mobile Rail Network going to the Virtual Network Operation Center are unreliable. However, taking our previous findings about the request performance the time for this particular transmission can be estimated using the average round trip and the processing times:

$$\bar{t} = \frac{1}{n}\sum_{i=1}^{n}(roundTripTime_i - processingTime_i) \quad (5)$$

$$= \frac{4.4seconds - 0.6seconds}{2} \quad (6)$$

$$= 1.9seconds \quad (7)$$

Given this estimate transmit time t, we can compute the total time it takes from for an alarm to go through the entire TSSN as shown in 10.

This includes the times from the *Sensor Node* to the *Alarm Processor* at the *Mobile Rail Network*, the approximated transmit time of 1.9 seconds, and the time from the *Alarm Processor* to the *Alarm Reporting service* at the *Virtual Network Operation Center*. On average this yields about 2.1seconds with the fastest time being just over 1.9 seconds and the slowest around 4.9 seconds.

Both, the road test with trucks and the short haul rail trial can be called successful because they displayed the capabilities of the TSSN, its good performance and that the functionality implemented in the web services worked. In particular, two of its main capabilities, location inquiry and alarm notification were extensively demonstrated. Furthermore, the time it took from registering alerts, propagating them through the *Transportation Security SensorNet* and sending out notifications accordingly is under 5 seconds and significantly smaller than expected for such a complex system.

The framework was used to capture and analyze the results presented here. For example we were able to break down the measurements into processing and transmission times as well as determine the performance of individual web services and the total alarm notification performance. Furthermore being able to visualize message flows helped identify problems and locate the particular part that was causing it.

VI. Conclusion

Capturing SOAP messages directly and using them as the basis for log analysis has the advantage of being a more structured approach because the SOAP messages adhere to specific web service specifications. This allows convenient mappings from the SOAP messages to elements defined in the specification and vice versa.

In this paper we presented a flexible framework for analyzing SOAP Messages in web service environments. The proposed solution consists of three parts. First, a logging module that can be attached to web services in order to capture SOAP messages and log their send and receive times. Second, a log parsing and processing library that allows for efficient correlation of messages, message flows and analysis. Finally, a visualization tool that provides convenient visual analysis of service interactions capabilities.

Acknowledgment

This work was supported in part by Oak Ridge National Laboratory (ORNL) Award Number 4000043403. This material is also partially based upon work supported while V. S. Frost was serving at the National Science Foundation.

References

[1] Y. Lafon and N. Mitra, "SOAP version 1.2 part 0: Primer (second edition)," W3C, W3C Recommendation, Apr. 2007, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[2] A. Chuvakin and G. Peterson, "Logging in the age of web services," *IEEE Security and Privacy*, vol. 7, pp. 82–85, 2009.

[3] C. Lim, N. Singh, and S. Yajnik, "A log mining approach to failure analysis of enterprise telephony systems," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, June 2008, pp. 398–403.

[4] M. Cinque, D. Cotroneo, and A. Pecchia, "A logging approach for effective dependability evaluation of complex systems," in *Dependability, 2009. DEPEND '09. Second International Conference on,* June 2009, pp. 105–110.

[5] R. Vaarandi, "Mining event logs with slct and loghound," in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, April 2008, pp. 1071–1074.

[6] A. Makanju, S. Brooks, A. Zincir-Heywood, and E. Milios, "Logview: Visualizing event log clusters," in *Privacy, Security and Trust, 2008. PST '08. Sixth Annual Conference on*, Oct. 2008, pp. 99–108.

[7] H. Lam, D. Russell, D. Tang, and T. Munzner, "Session viewer: Visual exploratory analysis of web session logs," in *Visual Analytics Science and Technology, 2007. VAST 2007. IEEE Symposium on*, 30 2007-Nov. 1 2007, pp. 147–154.

[8] J. Simmonds, Y. Gan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse, "Runtime monitoring of web service conversations," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, pp. 223–244, 2009.

[9] M. Gudgin, M. Hadley, and T. Rogers, "Web services addressing 1.0 - core," W3C, W3C Recommendation, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509.

[10] M. Gudgin, M. Gudgin, M. Hadley, T. Rogers, T. Rogers, and M. Hadley, "Web services addressing 1.0 - SOAP binding," W3C, W3C Recommendation, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509.

[11] D. Box, L. F. Cabrera, C. Critchley, F. Curbera, D. Ferguson, S. Graham, D. Hull, G. Kakivaya, A. Lewis, B. Lovering, P. Niblett, D. Orchard, S. Samdarshi, J. Schlimmer, I. Sedukhin, J. Shewchuk, S. Weerawarana, and D. Wortendyke, "Web services eventing (ws-eventing)," W3C, W3C Member Submission, Mar. 2006, http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315/.

[12] K. Lawrence, C. Kaler, A. Nadalin, R. Monzillo, and P. Hallam-Baker, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," OASIS, OASIS Standard, Feb. 2006, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[13] M. Kuehnhausen, "Service Oriented Architecture for Monitoring Cargo in Motion Along Trusted Corridors," Master's thesis, University of Kansas, Jul. 2009.

[14] S. Boag, A. Berglund, D. Chamberlin, J. Sim´eon, M. Kay, J. Robie, and M. F. Fern´andez, "XML path language (XPath) 2.0," W3C, W3C Recommendation, Jan. 2007, http://www.w3.org/TR/2007/REC-xpath20-20070123/.

[15] D. Mills, "Network Time Protocol (NTP)," RFC 958, Internet Engineering Task Force, September 1985, obsoleted by RFCs 1059, 1119, 1305. [Online]. Available: http://www.ietf.org/rfc/rfc958.txt

[16] D. T. Fokum, V. S. Frost, D. DePardo, M. Kuehnhausen, A. N. Oguna, L. S. Searl, E. Komp, M. Zeets, J. B. Evans, and G. J. Minden, "Experiences from a Transportation Security Sensor Network Field Trial," Information Telecommunication and Technology Center, University of Kansas, Lawrence, KS, Tech. Rep. ITTC-FY2009-TR-41420-11, June 2009.