A Z Specification for Reliability Requirements of a Service-based System

Manoj Lall[1*]
Department of Computer Science
Tshwane University of Technology
Pretoria, South Africa

John A. Van Der Poll[2]
Graduate School of Business Leadership
University of South Africa
Pretoria, South Africa

***Abstract*—The utilization of a Web services based application depends not only on meeting its functional requirements but also its non-functional requirements. The nonfunctional requirements express the quality of service (QoS) expected from a system. The QoS describes the capability of the service to meet the requirements of its consumers. In the context of Web services, considerations of QoS are critical for a number of reasons. *Reliability* is among the important QoS requirements of such distributed components as it enhances confidence in the services provided. Although the importance of QoS requirements are well established, they are often ignored until the end of the development cycle. Reasons cited for this are that they are difficult to *define* and *represent precisely*, and relay on entities that may *not be known at early stages*. This articles aims to address the challenges of incorporating the QoS at an early stage of service development and represent it in a precise manner. To achieve this goal, this paper makes use of a process model to facilitate the incorporation of the QoS attributes and Z as the specification language for its formalism. *Reliability* is used to exemplify the process. The Z schemas have been checked for syntax and type using the *Fuzz* type checker.**

***Keywords*—Reliability; non-functional requirements; Web services; Quality of Service; Formal specification; UML modelling; Z**

I. Introduction

Web services have enjoyed rapid acceptance in recent years. One of the motivating factors for this is the reliance on open standards for loose coupling and platform independent interface definition. Besides satisfying the functional requirements of an application, a Web service also has to cater for an equally important non-functional requirements (NFRs) such as reliability and availability. Several definitions of reliability in the context of software systems exists in the literature. For example,[1,2] have defined reliability as the probability of failure-free operation of a computer program for a specified time in a specified environment. In the context of Web services, [3] have defined reliability as the probability that a service invocation will be completed successfully. In the Web services domain, autonomous services depend on one another for their functioning hence their reliability is crucial for proper functioning of the entire system [4]. In addition, service requester may decide on the use of a particular service depending on the level of its reliability [5, 6].

A significant requirement of Web service applications is to operate in such a way that they should be functionally reliable and deliver consistent service at a variety of levels. These requirements do not focus only on the functional properties of services, but also on the QoS. The functional requirements of software is the required behavior of that software, whereas QoS specify the global constraints that must be satisfied by that software [7]. For instance, functional requirement of a service could be stated as “the service must be able to provide the physical address of an individual given a phone number”. Whereas, QoS requirement for the same service could be “the service must reliably operational for at least 90% of the time. Satisfying functional requirements has been the main priority of software development process due to business demands. However, QoS are an important concept in requirements engineering which plays a crucial role in the success of a software system [8]. Although the importance of QoS are well established, they are often ignored until the end of the development cycle, or even neglected altogether [9,10]. Frequently the reasons cited for this are that they are difficult to define and represent precisely, and may relate to entities that are not known at early stages [11,12,7]. For a Web service based application, it is important that both the functional and QoS are met.

This research recognizes that there is a fundamental need to define and specify the QoS (in this case, reliability) of Web services in an unambiguous (formal) manner. The benefits of representing the QoS in an unambiguous manner is enhanced precision and clarity in the specifications, rigour in its reasoning and proofs leading to the early detection of problems in the requirements.

A popular specification language used in formalization of systems is Z [13, 14]. Z applies typed sets, relations and functions within the context of first-order predicate logic. Its extensible toolkit of mathematical notations; its schema notation for specifying structures in the system, and for structuring the specification itself has enabled it to be used in specifying various types of systems [15, 16]. For instance, Z has been deployed in several application domains including safety critical systems, security systems, and other general purpose systems. The use of mathematics for specifying a system offer benefits such as precision, clarity, rigor in its reasoning and proofs, leading to the early detection of problems in the requirements [17]. It is due to these reasons that we have used Z.

*Corresponding Author

The rest of this article is structured as follows. In Section 2, we present an overview of related work and the methodology in Section 3. The implementation and discussion of the formalization process as applied to Reliability is presented in Section 4. The formal specification is developed using the Z's Established Strategy as presented in [18]. Our main contributions and directions for future work appear in Section 5.

II. Related Work

There is a huge volume of research work related to Web services and NFRs. Specifications dealing specifically with Reliability in the context of Web services are the WS-ReliableMesssaging [19] and WS-Addressing [20]. The WS-ReliableMessaging specification aims at providing for a robust communication framework. This is ensured by establishing standards for the acknowledgement of successful message delivery and the notification of transmission failure. WS-Addressing provides transport-neutral mechanism to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages [21]. The benefits of sending XML files using Web Services are: ensures platform independence, makes communication between the applications flexible, collaborative, and compatible. These specifications are mostly presented in informal natural language and semi-formal XML, hence subject to misinterpretations.

Incorporating QoS into the development phases of a systems has been proposed by [22]. They have employed a declarative approach for specifying QoS requirements. This approach is dedicated to control-loop systems such as avionics, robotics, and pervasive computing. A process model is proposed by [23] for integrating usability of interactive systems in Software Engineering life-cycle. Extension of Web Services architecture in order to increase system reliability and maintain client transparency has been proposed in [24]. Their proposed fault tolerant architecture makes use of several servers grouped in one autonomous unit based on servers and Web services to achieve enhanced reliability. In another attempt, [25] has focused on the reliability analysis of Web services by considering not only on the Web service component but also the middleware located beneath the Web service using a multilayered approach. The prediction of reliability of Web services have been researched by [3]. They have used the K-mean clustering techniques. Our research presented in this article complements the works of other researchers mentioned above by catering for an early incorporation of QoS requirements into the development cycle.

Some researchers have applied formal methods to various aspects of Web services reliability. For example, [26] has proposed a stochastic petri net based approach to predict the reliability of Web service composition. In another attempt, [27] have used the higher-order-logic theorem proving to conduct the reliability analysis of Logistics service supply chains. Higher-order-logic theorem has been used by [28] to ensure accurate and reliability of hardware components. Formal model for Web services composition has been proposed by [29]. They have studied an AI planning-oriented functional composition of Web services using the Causal link matrix. Formalization of availability, an important a QoS, using Z specification language has been carried out by [30].This article extends the research conducted by other researchers by providing a process model for converting QoS of Web services presented in an informal manner into a formal manner, and providing a formal specification of *reliability* using the general purpose specification language Z.

III. Research Methodology

In this section, we present a short overview of the process used to convert an informal specification into a formal one. Fig. 1 depicts the steps followed in the conversion process.

The process consists of the following five steps: In the first step, the requirements reflecting both the functional and the QoS attribute are specified using a natural language In the next step, a conceptual model representing entities that support the QoS are identified. These entities could be platform independent technologies, paradigms and mechanisms that support the QoS. A conceptual model is essentially a block diagram representing the requirements in a visual language. In the third step, the interactions of the supporting entities are represented in a framework. In essence, the framework models the structure and the behavior of the entities in the conceptual model. The next step involves the representation of the structure and the behavior of these entities using the Unified Modeling Language (UML) diagrams. The last step in this series of transformations involves mapping the UML diagrams to a Z representation, thus giving rise to the formal specification of the QoS. This article makes use of method described in [32] to map the UML representations to Z.

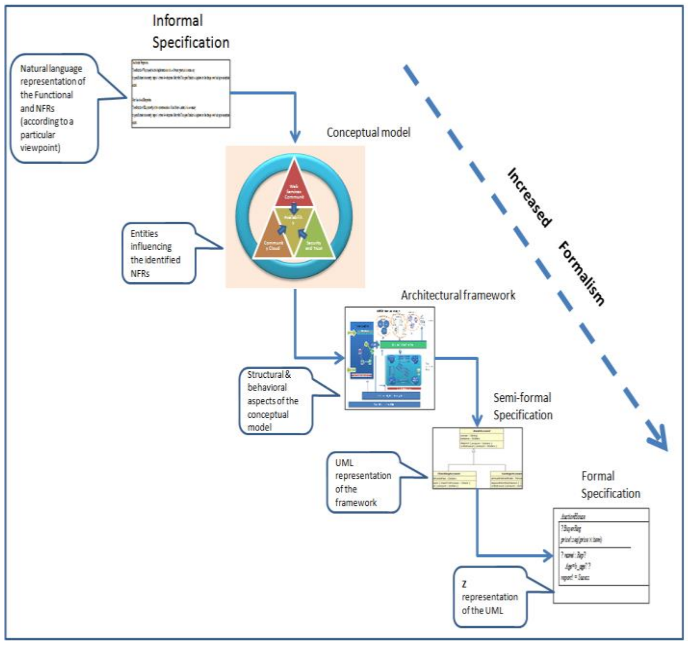
Informal Specification
Natural language representation of the Functional and NFRs (according to a particular viewpoint)
Conceptual model
Increased Formalism
Entities influencing the identified NFRs
Architectural framework
Structural & behavioral aspects of the conceptual model
Semi-formal Specification
UML representation of the framework
Formal Specification
Z representation of the UML

Fig. 1. Natural Language to Formal Specification Transformation Process Model (Adopted from [31]).

IV. Implementation and Discussion

Although WS-ReliableMessaging is a specification that enhances reliability of message transmission, it is specified using natural language and XML. Since XML is a natural language text based notation, it lacks formal semantics, and therefore, benefits of formal reasoning cannot be realized. Following the process depicted in Fig. 1, the requirement for reliable messaging is stated in a natural language. The next step requires that a conceptual model be constructed using the entities that have been identified to have an influence on this QoS requirement. For example, virtualization of networks and storage will have an influence on the availability of the network infrastructure and hence on the reliability of the messages that depend on these infrastructures. As virtualization is an enabling technology for Cloud computing, Cloud computing is considered as an entity in this conceptual model. Furthermore, to monitor and manage the networks, agents could be useful. Management in this regard would involve ensuring proper adherence to the agreed upon protocols such as AtMostOnce or ExactlyOnce and assisting with routing problems e.g. determining the optimal path to the destination. Reliability of the message will also be influenced by the security implementations as these messages often travel over public infrastructure. Similarly, trust becomes an important entity especially when messages cross organizational boundaries. The entities in the proposed reliability conceptual model are depicted by Fig. 2.

The next step in the process is the development of a framework based on the conceptual model. In the proposed reliability framework, the message that needs to be communicated is packaged according to the SOAP messaging structure. Before a message is sent to the next node, it is stored on some storage device. This process is followed for returning messages as well. The agents at the sender and receiver nodes manage the process depending on the protocol agreed upon (e.g. AtLeastOnce). In addition, the agents may be tasked with the responsibility of monitoring the status of the network (e.g. network traffic situation, network failure), to make informed decisions and enhance reliability of messages. The security and trustworthiness of the messages may be implemented using mechanisms such as encryptions and/or policies. The proposed framework for enhanced message reliability is shown in Fig. 3.

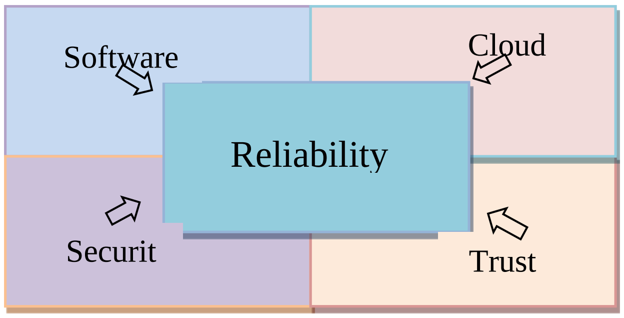

Fig. 2. A Conceptual Model of Message Reliability.

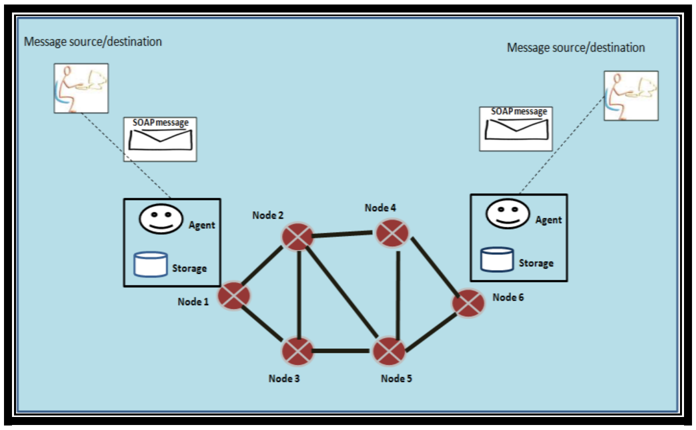

Fig. 3. An Enhanced Message Reliability Framework.

After the architectural framework, the next step in the methodology is to represent the elements that support the QoS in a UML model. Fig. 4 represents the class diagram and Fig. 5 represents the sequence diagram of the agent system that supports the QoS requirements. The agents used for monitoring and managing the networks (referred to as the MonitoringAgent) are instances of stationary agents which in turn is a specialization of the abstract class Agent. The MonitoringAgent is tasked with responsibility of storing the message, creating message packages and monitoring the network to keep up-to-date information of the network.

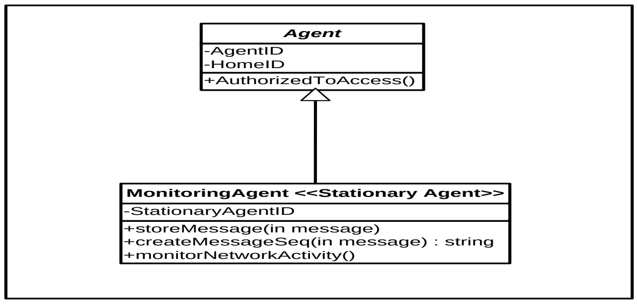
Agent
-AgentID
-HomeID
+AuthorizedToAccess()
MonitoringAgent <<Stationary Agent>>
-StationaryAgentID
+storeMessage(in message)
+createMessageSeq(in message) : string
+monitorNetworkActivity()

Fig. 4. Class Diagram of the Agents in the Enhanced Reliability Framework.

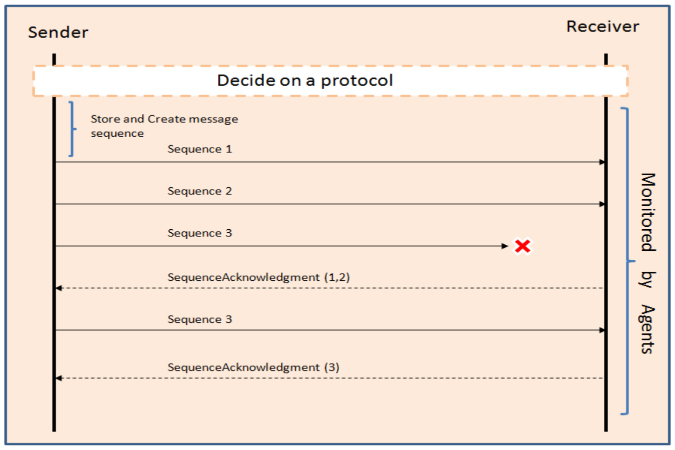

Fig. 5. Sequence Diagram of Enhanced Reliability Framework.

Before the message can be exchanged, the sender and the receiver of the message decide on protocols to be followed to support reliability. For instance, it may be decided that atLeastOnce protocol be used and that the receiver sends an acknowledgement after every third package is received. Fig. 5 shows how the protocol deals with a lost package. As the message is persisted on some storage device, it can be retrieved and re-sent.

V. Formal Specification

Once the QoS is modeled in UML, the next step in the methodology is to represent it in Z. Following the Established Strategy for presenting a Z specification [18], the enhanced reliability framework is formalized.

A. Given Sets and Globalvariables

It is customary in Z to first define the basic types of the specification. In our specification ADDESSS and BODY are a suitable representation of the set of all address and all message body.

$[ADDRESS, BODY]$

The set of addresses is a subset of the type ADDRESS and CreateSeq is a function that creates a sequence of messages sent to a particular destination. This is useful for the proper implementation of the transmission protocol (in this case atLeastOnce). The CreateSeq is a partial function as not all messages need to have a sequence number associated to it.

$addresses : \mathbb{P}\ ADDRESS$

$CreateSeq : Message \rightarrow\!\!\!\!\!+ \ \mathrm{seq}_1\ Message$

The set of responses generated by the various operations are defined as –

$REPORT ::= Message\_copied\_on\_disk \mid Message\_sequence\_created \mid Message\_is\_new\_and\_not\_stored \mid Message\_already\_stored \mid No\_message\_sequence\_should\_be\_created$

B. Abstract State Space

An agent is modelled as having an identity (agentID) and a home where it is created. Additionally, it is created to perform a set of tasks for which it requires access to certain resources at a particular host, denoted by accessResource.

```
_Agent________________________________
| agentID : IDENTITY
| home : NODE
| tasks : ℙ TASK
| accessResource : NODE ⇸ ℙ RESOURCE
|______________________________
| agentID ∈ identities
| home = createdOn agentID
| home ∈ dom accessResource
| tasks ⊆ userTasks
| accessResource home ⊆ resources
 ______________________________________
```

Given a particular agentID, createdOn returns the node on which the particular agent was created (i.e. gives the home of the agent). This information is useful in establishing trust amongst the agents in a system. For example, knowledge of where a particular agent was created may give an indication of who created that agent, and if such agent is trusted then all agents created at that particular host may also be trusted. Agents are created for meeting certain user requirements, hence the tasks the agent is assigned are part of userTasks. At its home node, an agent has access to all the resources available at that host.

A mobile agent is modelled as an agent that has the ability to migrate to different nodes (hosts), whereas, a stationary agent has no such ability. For the purpose of this research, no distinction is made between an agent and a stationary agent.

```
_Mobile_Agent__________________________
| Agent
| agentItinerary : seq NODE
|__________________________
| # (agentItinerary) = # (ran agentItinerary)
 _______________________________________
```

For the migration of a mobile agent, the agent can follow a predefined itinerary made up of nodes or construct one as it migrates from one node to another. In this paper, our agents are given an itinerary when it is created, and it can visit a node more than once on the same journey. An agent itinerary gives an indication of where the agent was created (by determining the itinerary head), and where it has been. The number of elements in the mobile agent's itinerary is equal to the number of nodes the agent has visited. This information is useful in establishing a trust level for the agent.

Besides having specific tasks, a stationary agent may be modelled to be the same as an agent.

$MA \triangleq Agent$

Defining the abstract state space of the system, a message is modeled as having an identity number, source and destination addresses and a body consisting of the actual content to be delivered.

```
_Message____________________________
| messageID : IDENTITY
| sourceAdd : ADDRESS
| destinationAdd : ADDRESS
| body : BODY
|______________________________
| sourceAdd ∈ addresses
| destinationAdd ∈ addresses
| sourceAdd ≠ destinationAdd
| body ≠ ∅
| messageID ∈ identities
 _____________________________________
```

C. Initial States

In the initial state of the *Message* schema denoted by the schema *Init_Message,* the system generates a *messageID* (*mid*), the source and the destination addresses are empty. Furthermore there is no message to be transmitted hence the body of the message is also empty. The *Init_Message* schema is defined as shown below:

$\_Init\_Message\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$

$| \; Message'$
$| \; mid! : IDENTITY$
$| \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$
$| \; sourceAdd' = \emptyset$
$| \; destinationAdd' = \emptyset$
$| \; body' = \emptyset$
$| \; messageID = mid!$
$| \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$

and

$Init\_MA \triangleq Init\_Agent$

D. A Proof Obligation

The next step is to show that the Init_Message and Init_MA can be realized. This can be done by showing that the variables sourceAdd′ and destinationAdd′, body′ and messageID are each of the type indicated and the predicates of Message and MA still holds. The initialization theorem can be stated as:

$\vdash \exists$ Message′ • Init_ Message (1)

$\vdash \exists$ *MA′* • *Init_ MA* (2)

The Proof of (1) is as follows:

- Given the predicate sourceAdd′ = ∅ ∧ destinationAdd′ = ∅ ∧ body′ = ∅ ∧ messagID′ = mid!, it is required to show that sourceAdd′ ∈ ∅ ∧ destinationAdd′ ∈ TASK ∧ body′ ∈ ∅ ∧ messageID′ ∈ IDENTITY. The proof is quite apparent, since ∅ is an element of ADDRESS, ∅ is an element of BODY, and mid! is also an element of IDENTITY.
- Since the set sourceAdd′, destinationAdd′, body′ are all empty, and messagID′ is of type IDENTITY, therefore all four predicates below the second horizontal line in schema Init_Message hold.

The proof for (2) is obtained by proving that agentItinerary′ ∈ seq NODE, given agentIninerary′ = ⟨ ⟩. The proof is quite apparent, since ⟨ ⟩ is an element of seq NODE.

- Since agentItinerary′ is empty, the predicate agentItinerary′ = ⟨ ⟩ in schema Init_Mobile_Agent holds.

E. Partial System Operations

From the sequence diagram (Fig. 5), the main operations that can be identified are the store message and create message sequence. The copy of the message is saved on some persistent device before the message is sent. The message copy is defined as:

$MessageCopy \triangleq Message$

The store message operation may be defined as a function:

$\_StoreMessage\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$

$| \; message? : Message$
$| \; copy! : MessageCopy$
$| \; report! : REPORT$
$| \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$
$| \; message?.sourceAdd = copy!.sourceAdd$
$| \; message?.destinationAdd = copy!.destinationAdd$
$| \; message?.body = copy!.body$
$| \; message?.messagID = copy!.messageID$
$| \; report! = Message\_ \; copied\_on\_disk$
$\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$

The createMessageSequence schema models create a sequence of message operation. This happens only for a series of messages sent to a particular destination.

$\_CreateMessageSequence\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$

$| \; m1?, m2? : Message$
$| \; report! : REPORT$
$| \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$
$| \; (m1?.sourceAdd = m2?. \; sourceAdd \land m1?.destinationAdd =$
$| \; m2?.destinationAdd) \Rightarrow ((CreateSec \; m1?) \frown (CreateSec \; m2?))$
$| \; report! = Message\_sequence\_created$
$\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$

F. Enquiry Operations

To enquire if a particular message has been saved before being sending, the QueryStoreMessage is presented:

$QueryStoreMessage\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$

$| \; \Xi \; Message$
$| \; message? : \; Message$
$| \; report! : REPORT$
$| \; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$
$| \; message?.messagID \notin identities \Rightarrow report! =$
$| \; Message\_is\_new\_and\_not\_ \; stored$
$| \; message?.messagID \in identities \Rightarrow report! =$
$| \; Message\_already\_stored$
$\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$

G. Tabulating Preconditions

As prescribed by the Established Strategy for presenting a Z specification, we next summarise the partial operations together with their inputs, outputs, and their preconditions in Table I.

TABLE I. SUMMARY OF PARTIAL OPERATIONS OF THE RELIABILITY FRAMEWORK

Operation	**Input and Output**	**Preconditions**
StoreMessage	*message*? : *Message* *copy*! : *Message* *report*! : *REPORT*	*message*? ≠ ∅
CreateMessageSequence	*m*1?, *m*2? : *Message* *report*! : *REPORT*	*m1 ≠ m2*
QueryStoreMessage	*message*? : *Message* *report*! : *REPORT*	*message*? ∈ *identities*

H. Error Conditions

Several errors conditions may arise in while storing the SOAP message. For example, saving an already saved message (shown by the schema WrongStoreMessageor creating a sequence of messages (WrongMessageSequence that is not meant to be sent to the same destination.

```
_WrongStoreMessage____________________
| Ξ Message
| message? : Message
| copy?: Message
| report! : REPORT
| ______________________________
| message?.messageID = copy?.messageID
| report! = Message_already_stored
 ______________________________________
```

```
_WrongMessageSequence__________________________
| Ξ Message
| message1?, message2 : Message
| report : REPORT
| _______________________________________
| message1?.destinationAdd ≠ message2.destinationAdd
| report ! = No_message_sequence_should_be_created
 _______________________________________________
```

Another error condition may arise when a message that is being queried does not exist:

```
_WrongQueryStoreMessage
_______________
| Ξ Message
| message? : Message
| report! : REPORT
| _____________________________
| message?.messagID ∉ identitres
| report! = message_does_not_exist
 ______________________________________
```

I. Total System Operations

Incorporating the partial operations listed in Table I with their error conditions presented in this section leads to the total (robust) operations:

TotalStoreMessage ≙ *StoreMessage* ∨ *WrongStoreMessage*

TotalCreateMessageSequence ≙ *CreateMessageSequence* ∨

WrongMessageSequence

TotalQueryStoreMessage ≙ *QueryStoreMessage* ∨

WrongQueryStoreMessage

VI. CONCLUSIONS AND FUTURE WORK

This paper has introduced a process for formalizing QoS attributes of Web service and demonstrated its applicability in formalizing Reliability. The formalization is achieved by makes use of a general purpose formal notation (i.e. Z). The formalism of the systems architecture leads to specifications that are more precise and therefore, more likely to lead to unambiguous statements during the implementation stages. The formalism, besides leading to specifications that are more precise, allows for reasoning about the specification to take place. Proofs of a number of fundamental properties of the specifications are presented to demonstrate the benefits offered by formalizing the specification. The very first step in the proposed process model requires that the quality attribute be specified, though informally. This leads to an early incorporation of QoS requirement into the development process and subsequently to benefits such as less costly error corrections.

There are several dimensions in which this work may be expanded. For instance, mappings between UML and other formal methods such as Rodin may be investigated. Augmenting existing mechanisms for the automatic generation of UML models from architectures ought to receive attention as well as enhancing tool support for the automatic generation of formal specifications from UML models.

REFERENCES

[1] Pham, H., Software reliability. 2000: Springer Science & Business Media.

[2] Musa, J., A. Iannino, and K. Okumoto, Engineering and managing software with reliability measures. 1987, McGraw-Hill.

[3] Silic, M., G. Delac, and S. Srbljic. Prediction of atomic Web services reliability based on k-means clustering. in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. 2013.

[4] Lall, M., L.M. Venter, and J.A. van der Poll, Evaluating the Second Generation Web Services Specifications for Satisfying Non-Functional Requirements, in World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2010, J. Sanchez and K. Zhang, Editors. 2010, AACE: Orlando, Florida, USA. p. 1919-1929.

[5] Aiello, M. and P. Giorgini, Applying the Tropos methodology for analysing Web services requirements and reasoning about Qualities of Services. CEPIS Upgrade-The European journal of the informatics professional, 2004. **5**(4): p. 20-26.

[6] Driss, M., Aljehani, A., Boulila, W., Ghandorh, H. and M. Al-Sarem, Servicing Your Requirements: An FCA and RCA-driven Approach for Semantic Web Services Composition. IEEE Access, 2020.

[7] Galster, M. and E. Bucherer. A Taxonomy for Identifying and Specifying Non-Functional Requirements in Service-Oriented Development. in Services - Part I, 2008. IEEE Congress on. 2008.

[8] Singh, P. and A.K. Tripathi, Exploring Problems and Solutions in estimating Testing Effort for Non Functional Requirement. International Journal of Computers & Technology, 2012. **3**(2): p. 284-290.

[9] Hasnain, M., Pasha, M.F., Ghani, I., Mehboob, B., Imran, M. and A. Ali, Benchmark Dataset Selection of Web Services Technologies: A Factor Analysis. IEEE Access, 2020. **8**: p. 53649-53665..

[10] Tambe, S., A. Dabholkar, and A. Gokhale. CQML: Aspect-Oriented Modeling for Modularizing and Weaving QoS Concerns in Component-Based Systems. in Engineering of Computer Based Systems, 2009. ECBS 2009. 16th Annual IEEE International Conference and Workshop on the. 2009.

[11] Saleh, K. and A. Al-Zarouni. Capturing non-functional software requirements using the user requirements notation. 2004.

[12] Rosa, N., G. Justo, and P. Cunha, Incorporating non-functional requirements into software architectures. Parallel and Distributed Processing, 2000: p. 1009-1018.

[13] Pressman, R., Software Engineering: A Practitioner's Approach. 7th ed. 2010: McGraw-Hill, Inc.

[14] Gouasmi, T., A. Regayeg, and A.H. Kacem. Automatic Generation of an Operational CSP-Z Specification from an Abstract Temporal^Z Specification. in Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual. 2012.

[15] Woodcock, J. and J. Davies, Using Z: specification, refinement, and proof. Vol. 1. 1996: Prentice Hall.

[16] Utting, M., P. Malik, and I. Toyn, Transformation rules for Z, in Proceedings of the Fifteenth Australasian Symposium on Computing: The Australasian Theory - Volume 94. 2009, Australian Computer Society, Inc.: Wellington, New Zealand. p. 73-82.

[17] Palshikar, G.K., Applying formal specifications to real-world software development. Software, IEEE, 2001. **18**(6): p. 89-97.

[18] Potter, B., D. Till, and J. Sinclair, An introduction to formal specification and Z. 1996: Prentice Hall PTR.

[19] Davis, D., Web services reliable messaging (WS-ReliableMessaging). Technical report, Technical report, OASIS, 2006.

[20] Box, D., et al., Web services addressing (WS-Addressing). 2004, Citeseer.

[21] Vieira, M., N. Laranjeiro, and H. Madeira. Assessing Robustness of Web-Services Infrastructures. in Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on. 2007.

[22] Gatti, S., E. Balland, and C. Consel. A step-wise approach for integrating QoS throughout software development. in International Conference on Fundamental Approaches to Software Engineering. 2011. Springer.

[23] Granollers, T. User Centred Design Process Model. Integration of Usability Engineering and Software Engineering. in Proceedings of INTERACT. 2003.

[24] Toader, C., Increasing reliability of Web services. Journal of Control Engineering and Applied Informatics, 2010. **12**(4): p. 30-35.

[25] Rahmani, M., A. Azadmanesh, and H. Siy, Architecture-based reliability analysis of Web services in multilayer environment, in Proceedings of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems. 2011, Association for Computing Machinery: Waikiki, Honolulu, HI, USA. p. 57–60.

[26] Zhong, D. and Z. Qi. A petri net based approach for reliability prediction of Web services. in OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". 2006. Springer.

[27] Ahmad, W., et al. Towards Formal Reliability Analysis of Logistics Service Supply Chains using Theorem Proving. in IWIL@ LPAR. 2015.

[28] Hasan, O., S. Tahar, and N. Abbasi, Formal reliability analysis using theorem proving. IEEE Transactions on Computers, 2009. **59**(5): p. 579-592.

[29] Hernández, A.G. and M.N.M. García. A formal definition of RESTful semantic Web services. in Proceedings of the First International Workshop on RESTful Design. 2010.

[30] Lall, M., J.A. van der Poll, and L.M. Venter, Towards A Formal Definition Of Availability Of Web Services, in The International Conference on Computing, Networking and Digital Technologies (ICCNDT 2012). 2012: Gulf University, Bahrain. p. 154 - 165.

[31] Lall, M., J.A. Van Der Poll, and L. M. Venter, A Process Model for the Formalisation Of Quality Attributes of Service-Based Software Systems. Malaysian Journal of Computer Science, 2019: p. 284-303, 32(4).

[32] Shroff, M. and R.B. France. Towards a formalization of UML class structures in Z. in Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International. 1997.