Formal Modeling and Verification of Messaging Framework of Simple Object Access Protocol

Manzur Ashraf
Faculty, Dept of CSE,
BRAC University,
66 Mohakhali, Dhaka 1212
Email: manzur_bd@bracuniversity.ac.bd

Abstract
Since novel and complex multi-party edge services are deployed on the Internet, different upper-layer protocols with complex as well as reactive communication models and event dependencies are increasingly being specified and adopted. To ensure that such protocols (and compositions thereof with existing protocols) do not result in unacceptable behaviors (e.g., deadlocks or live locks), a methodology is desirable for the automated checking of the "correctness" of these protocols. In this paper, we present ingredients of such a methodology. Specifically, we show how SPIN, a tool used for the formal systems verification purposes, can be used to verify as well as quickly identify problematic behaviors (if any) in core component of emergent web-service architecture with non-trivial communication constructs—such as Simple Object Access Protocol (SOAP). In this paper, we identify essential elements, model and verify the most recent SOAP 1.2 Processing model of Messaging Framework – one of the main parts of SOAP 1.2 using SPIN. It will evidently provide an insight into the scope and utility of formal methods based on state space exploration in testing larger and complex systems, for example the complete Web-service suit.

Keywords: Formal verification, SOAP, Model checking, SPIN, PROMELA, Verification of protocols, Verification tools, Protocol Engineering.

1 INTRODUCTION

Modeling is the process of abstracting the functional specifications of a system into a minimal working specimen that enables us to understand and analyze a particular aspect of the system more closely. Verification means process of examining this specification for the presence of various errors that could lead to improper system operation. There are five elements of specifications particularly service part, assumption, vocabulary, encoding (format) & procedure rules for a protocol.

Spin [8, 11] is a verification system that supports the design and verification of finite state asynchronous, distributed and concurrent systems. Programs are constructed in the PROMELA programming language, which is similar to an ordinary programming language, with additional non-deterministic specification-based constructs. Processes communicate either via shared variables or via message passing through buffered channels. Properties to be verified are described in Linear Temporal Logic (LTL). The Spin model checker can automatically determine whether a program satisfies a property and in case the property does not hold, an error trace is generated.

This paper documents an application of SPIN to model , simulate and formally verify an emergent communication protocol for web services – Simple Object Access protocol (SOAP) [12,13]. SOAP is characterized by distributed asynchronous computing which makes it a potential candidate for model checking and verification and it can render apprehension into the breadth and applicability of formal methods based on state space exploration in modeling and verifying larger software systems.

2 ORGANIZATION OF THE PAPER

The remainder of the paper is organized as follows. In next section, we present an overview of Simple Object Access Protocol (SOAP) and evolution of it. In section 4, we elaborately describe the five elements of a protocol in terms of SOAP characteristics. In section 5, we highlight some research works specially a related protocol-verification using SPIN. Our level of abstraction for the purpose of modeling is illustrated in section 6, followed by the flow-diagram of processing model of SOAP 1.2 Messaging framework. In section 8, we discuss the simulation and verification results of this protocol, followed by some future research directions in this regard.

3 OVERVIEW OF SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

Web services [13] are emerging to provide a systematic and extensible framework for application-to-application interaction, built on top of existing Web protocols and based on open XML standards. Web services aim to simplify this process by defining a standardized mechanism to describe, locate, and communicate with online applications. Essentially, each application becomes an accessible Web service component that is described using open standards. The Web services framework is divided into three areas- a) the simple object access protocol (SOAP, www.w3.org/2000/xp) which enables communication among Web services; b) the Web Services Description Language (WSDL, www.w3.org/TR/wsdl.html), which provides a formal, computer-readable description of Web services; and c)

the Universal Description, Discovery, and Integration (UDDI, www.uddi.org) directory, which is a registry of Web services descriptions.

Given the Web's intrinsically distributed and heterogeneous nature, communication mechanisms must be platform-independent, international, secure, and as lightweight as possible. XML is now firmly established as the lingua franca for information and data encoding for platform independence and internationalization. Building a communication protocol using XML is thus a natural answer for Web services. SOAP, which was initially created by Microsoft and later developed in collaboration with Develop mentor, IBM, Lotus, UserLand, etc, is an XML-based protocol for messaging and remote procedure calls (RPCs). Rather than defining a new transport protocol, SOAP works on existing transports, such as HTTP, SMTP, MQSeries, etc.

3.1 Evolution of SOAP

To help developers build Web services and link heterogeneous components over the Internet and in the context of Microsoft Windows DNA 2000 solution, Microsoft submitted to the Internet Engineering Task Force (IETF) an Internet draft specification for the Simple Object Access Protocol (SOAP) in September 13, 1999 [3, 4].

Later, in May 8, 2000, World Wide Web Consortium (W3C) acknowledged receipt of a submission request including the SOAP 1.1. Eleven (11) predominant companies including Microsoft, IBM, etc participated to that submission. It includes the text of SOAP 1.1 specification along with three parts: SOAP Envelope, Encoding scheme and RPC mechanism. The only bindings described how to use SOAP in combination with HTTP and HTTP Extension Framework.

In June 9, 2003, W3C released first public working draft for SOAP 1.2. It had an additional part other that previous three: a binding convention for exchanging message with any underlying protocol. However, the only bindings defined in the draft were how to use SOAP in combination with HTTP and HTTP Extension Framework. Earlier in May 7, 2003, W3C published SOAP 1.2 as a 'proposed recommendation' (status). It provided four parts: SOAP 1.2 Part 0: Primer, SOAP 1.2 Part1: Messaging Framework, SOAP 1.2 part 2: Adjuncts and SOAP 1.2 part 3: Specification Assertions and Test collections. It removed ambiguities found in SOAP 1.1 and included improved error messages to facilitate interfaces.

In June 24, 2003, W3C released SOAP 1.2 as a 'W3C recommendation' which is equivalent to a web-standard, indicating that the specification is stable, contributes to web-interoperability, had been reviewed by the W3C members, who favored its adoption by the industry.

4 IN-DEPTH ANALYSES OF FIVE ELEMENTS OF SOAP

The SOAP Version 1.2 specification consists of four parts as was described in previous section. Part 1 of the SOAP Version 1.2 specification defines the SOAP messaging framework [10] consisting of:

a) The SOAP processing model defining the rules for processing a SOAP message.
b) The SOAP Extensibility model defining the concepts of SOAP features and SOAP modules.
c) The SOAP underlying protocol binding framework describing the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes.
d) The SOAP message construct defining the structure of a SOAP message.

SOAP processing model [9,10] assumes a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries. The SOAP distributed processing model can support many Messsage Exchange Patterns (MEPs) including but not limited to one-way messages, request/response interactions, and peer-to-peer conversations for a description of the relationship between SOAP message exchange patterns and the SOAP extensibility model.

We will concentrate our modeling and verification in SOAP 1.2 Processing Model of SOAP Message Framework. We briefly present here the five elements of specification particularly covering those areas that we are going to model and validate formally.

4.1 Services

SOAP is a lightweight protocol for exchange of structured and typed information between peers in a decentralized, distributed environment using XML. Therefore, this protocol is used for client-server architectural model. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to Remote Procedure Calls (RPC).

4.2 Assumptions about the Environment

The environment in which the protocol is to be executed consists a server, at least one client and a transmission channel. The users can be assumed to simply submit a request for initiating a process (e.g., query into a database) and response is provided by server in terms of SOAP response. In case of RPC calls, server provides response by initiating a procedure in native language, resident to server and giving response to the client. The three major assumptions of SOAP processing model of messaging framework in this paper are:

a) Message/Request is sent to an ultimate SOAP receiver via zero SOAP intermediaries.
b) Message corruption or disruption is handled in lower protocol-binding level [10].
c) The manner in which the underlying protocol is used to transmit information between adjacent nodes in the message path is handled by SOAP protocol binding framework [10,12]. There is no such issues in Processing model.

However, the transmission channel is assumed to message-losses.

4.3 Protocol Vocabulary

According to [9, 10, 12], a SOAP message is an ordinary XML document containing the following elements: a required 'Envelope' element that identifies the XML document as a SOAP message, an optional 'Header' element that contains header information, a required 'Body' element that contains call and response information and an optional 'Fault' element that provides information about errors occurred while processing the message

4.3.1 The SOAP Envelope Element

The required SOAP Envelope element is the root element of a SOAP message. It defines the XML document as a SOAP message. It should always have the value of "http://www.w3.org/2001/12/soap-envelope*"* as its name-space (defined by xmlns: soap).If a different namespace is used, the application must generate an error and discard the message. The SOAP *encodingStyle* attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will be applied to that element's contents and all child elements. A SOAP message has no default encoding. It contains header & fault codes.

4.4 Message format

Message format of SOAP-envelop is shown in following:

```
enum Boolean = {0 ,1};
enum Error { Version_mismatch, mustUnderstand,
Client, Server};
 struct soap_envelop {
        char
*namespace="http://www.w3.org/2001/12/soap-
envelope";
        char
*encodingStyle="http://www.w3.org/2001/12/soap-
encoding";

        struct soap_header{   /* Optional */
        char *actor;
        enum Boolean mustUnderstand;
        char
*encodingStyle="http://www.w3.org/2001/12/soap-
encoding";
        };

        struct body_element{  /* Optional */
                struct Fault_element{
                enum Error faultCode;
                char *faultString;
                char *faultAction;
                char *detail;
                };
        unsigned char* data;
        char *name_spec_qualified;
        };

};
```

4.5 Procedure rules

The SOAP processing model specifies how a SOAP receiver processes a SOAP message. It applies to a single message only, in isolation from any other SOAP message. The SOAP processing model itself does not maintain any state or perform any correlation or coordination between messages, even, for example, when used in combination with a SOAP feature which involves sending multiple SOAP messages in sequence, each subsequent message depending on the response to the previous message. It is the responsibility of each such features to define any combined processing.
The following List 1 gives a typical SOAP request-response example:

List 1:

Request :

```
<SOAP-ENV: Body>
        <m:GetLastTradePrice xmlns:m="some-URI">
                <symbol> DEF </symbol>
        </m:GetLastTradePrice>
</SOAP-ENV: Body>
```

Response:

```
<SOAP-ENV: Body>
        <m:GetLastTradePriceResponse
xmlns:m="some-URI">
                <price> 23 </price>
        </m:GetLastTradePriceResponse>
</SOAP-ENV: Body>
```

5 RELATED WORKS

One of the important aspect of SOAP research is the 'transmission of SOAP envelops using HTTP'. Binding SOAP to HTTP provides the advantage of being able to use the formalism and decentralized flexibility of SOAP with the rich feature set of HTTP. SOAP 1.1 naturally follows the HTTP request/response message model providing SOAP request parameters in a HTTP request and SOAP response parameters in a HTTP response. For SOAP 1.2 this HTTP binding is optional.

Different verification of HTTP [1,2,3,4,5,6] from intuitive justification of human and formal verifiers (tools) was performed with continuous improvement most of the time. Among them the following work is

remarkable to this context as well as verification assessment of SOAP in terms of HTTP-binding.

Bradley et. el. [7] analyzed HTTP using SPIN and came up with following important findings:
First, their attempts to build models from the RFCs highlighted several textual ambiguities (or, arguably, errors): 1. The two interpretations of 100 Continue in RFC2068, 2. The absence of a backward-compatibility option for RFC2616 proxies to mirror the option for RFC2616 servers, and 3. The natural presence of potential deadlock cases for RFC2616 servers choosing not to implement the backward-compatibility option. Second, they verified that an entirely HTTP/1.1 world (regardless of which revision) is deadlock-free, except in the presence of RFC2616 servers which choose not to implement the backward-compatibility option. Third, their experiments uncovered several classes of agent arrangements in which combinations of HTTP/1.1 agents leading up to an HTTP/1.0 server (or equivalent) agent are prone to deadlock; while this was partially addressed in the RFC2616 revision to HTTP/1.1, certain failure cases remain in environments of mixed RFC2068 and RFC2616 agents.

In case of SOAP transportability, while the underlying binding protocol is HTTP type, we may encounter similar problem as in [7]. Actually the correctness of SOAP transmission depends on the type of underlying protocol used.

6 LEVEL OF ABSTRACTION USED IN OUR MODEL

The key to building useful and analyzable models is to abstract away enough details to make the models a manageable size while retaining enough detail to be meaningful and reflective of the underlying processes and behaviors. We have partitioned SOAP processing model into two parts: Requester (e.g., Client) and Responder (e.g., Server). We have abstracted all the message types of SOAP- Envelope (request/response) into three major types of message types as request, response and SOAP processing error. The envelope from Requester is a message type ‘request’, the error-free response-envelope from responder is a message type ‘response’ and faulty response-envelope from responder due to processing error in receiver side is a message type ‘err’.

As was said earlier (Section 4), SOAP 1.2 Messaging Framework distinguishes between (a)’processing model’ and (b) ‘underlying protocol binding’ (e.g., with HTTP). We will model and validate the ‘processing model’ part which interacts with ‘underlying protocol binding’ part – responsible for transmission issues (discussed in previous section). We will abstract the functionality of the later part. In assumption part of protocol specification (Section 4.2), we illustrate that message corruption is handled by underlying protocol. Our model will catch message loss only.

7 FRAMEWORK OF SOAP 1.2 PROCESSING MODEL

We have abstracted all the message types of SOAP-Envelope into three major types of message types as follows:
mtype = { request, response, err };

All the channels are declared as global objects. The following channels are central to the model. The channel *to_sndr* represents the communication between requester (Figure 1) and responder (Figure 2) which carries SOAP request with sequence number. The channel *to_rcvr* denotes communicating channel from responder to requester.
chan to_sndr = [1] of { mtype, bit };
chan to_rcvr = [1] of { mtype, bit };

8 SIMULATION & VERIFICATION RESULTS

After constructing the code in XSPIN environment [11], we run the simulation. The simulation output and message sequence chart are shown in Figure 3.

8.1 Correctness

We would like to prove that correct implementations of the model acting in all combinations of roles are *well behaved*, by which we mean that the rules of the protocol prevent the system from entering undesirable states such as deadlock (all agents blocked waiting for others to act) or livelock (agents interacting in a way that produces no “progress”). Alternatively correctness properties can be expressed as a) properties of reachable states (Safety) and as properties of sequences of states (Liveness).

Checking ‘Safety’ comprises two things: (1) checking local process assertions and invariants (if any), and (2) checking proper termination points of progress (end state levels – if any).

Validating ‘Liveness’ comprises (1) looking for acceptance cycles, (2) looking for non-progress cycles, (3) using never claims – which defines an observer process that executes synchronously with the system, and (4) trace assertions – to reason about valid or invalid sequences of send or receive statements.

We have used the major never claim here as follows. It represents whenever a message is sent by the ‘Responder’, it will eventually accepted by the ‘Requester’.

The following LTL claim us used:

$$!([](p \rightarrow X(<>q)))$$

Where p corresponds *‘to_rcvr?[request(1)]’* and q corresponds to *‘to_sndr?[response(1)] OR to_sndr?[err(1)]’*.

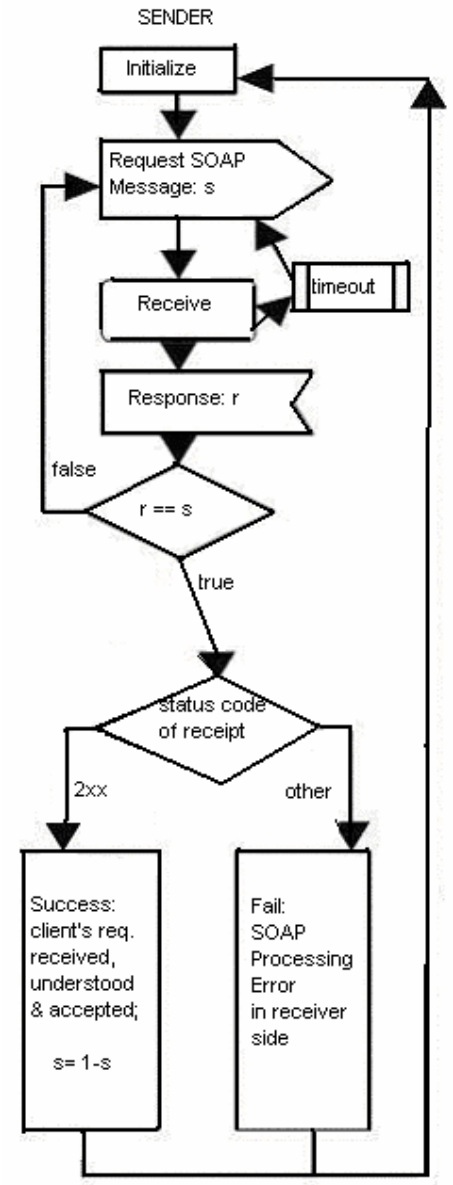

Figure 1: Sender/Requester Process

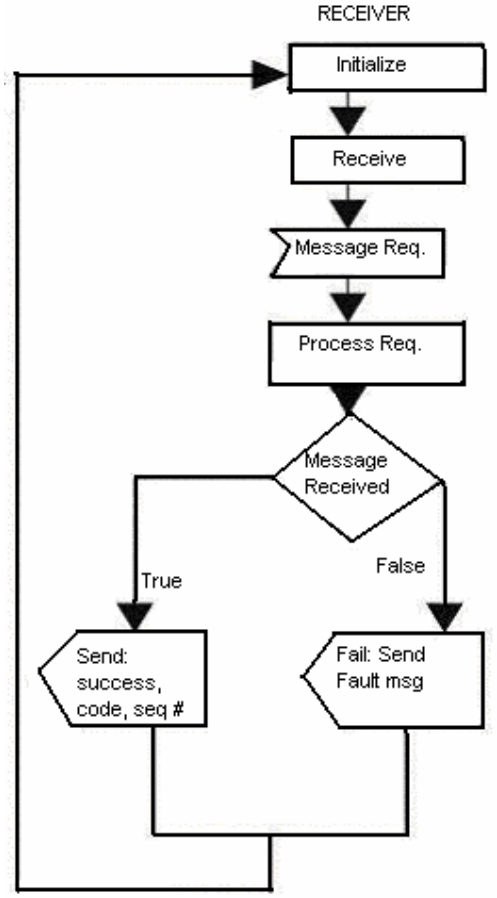

Figure 2: Receiver/Responder Process

The results of checking 'Safety' and 'Liveness' property in 'Hash-compact search mode' is illustrated in table 1.

From the verification results, hash factor is very large (>100). It means we are confident of sufficient coverage. All the validation runs confirms that the correctness requirement of the SOAP 1.2 Processing model of Messaging Framework is properly met.

However, if all messages sent by the 'Responder' are lost, an acceptance cycle will be detected, meaning that the never claim is matched.

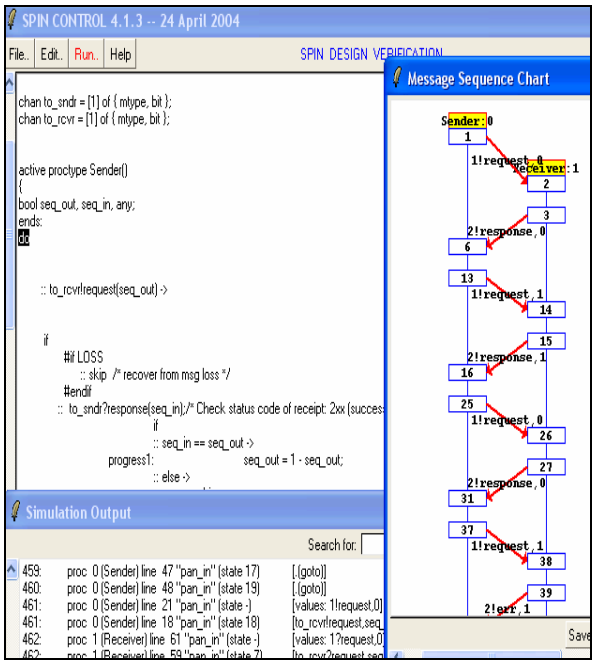

Figure 3: XSPIN output of simulation run

Table 1: Verification in Hash-compact search mode

Safety	Liveness
(Spin Version 4.1.3 -- 24 March 2004) + Partial Order Reduction Hash-Compact 4 search for: never claim + assertion violations - (disabled by -A flag) cycle checks - (disabled by -DSAFETY) invalid end states - (disabled by never claim) State-vector 44 byte, depth reached 31, errors: 0 27 states, stored 5 states, matched 32 transitions (= stored+matched) 0 atomic steps hash conflicts: 0 (resolved) (max size 2^19 states) Stats on memory usage (in Megabytes): 0.001 equivalent memory usage for states (stored*(State-vector + overhead)) 0.303 actual memory usage for states (unsuccessful compression: 21582.05%) State-vector as stored = 11215 byte + 8 byte overhead 2.097 memory used for hash table (-w19) 0.320 memory used for DFS stack (-m10000) 2.622 total actual memory usage	Spin Version 4.1.3 -- 24 March 2004) + Partial Order Reduction Hash-Compact 4 search for: never claim + assertion violations + (if within scope of claim) non-progress cycles + (fairness disabled) invalid end states - (disabled by never claim) State-vector 44 byte, depth reached 31, errors: 0 64 states, stored 37 states, matched 101 transitions (= stored+matched) 0 atomic steps hash conflicts: 0 (resolved) (max size 2^19 states) Stats on memory usage (in Megabytes): 0.003 equivalent memory usage for states (stored*(State-vector + overhead)) 0.303 actual memory usage for states (unsuccessful compression: 9104.93%) State-vector as stored = 4727 byte + 8 byte overhead 2.097 memory used for hash table (-w19) 0.320 memory used for DFS stack (-m10000) 2.622 total actual memory usage

9 CONCLUSION & FUTURE WORK

We have attained the major goal of modeling and verifying SOAP Ver. 1.2 Processing Model of Messaging Framework. We have also verified the five parameters of that protocol-part. We have demonstrated only the processing model by separating underlying protocol binding framework. Investigation into the combined framework using SPIN model checker is a future concern. Similarly the verification of the complete 'web-service' suite incorporating HTTP server, SOAP server, UDDI service and requester (client) is future objective. Comparative analysis of formal specification of SOAP using UML and constructed specification from model using SPIN is another direction for comprehensive assessment of SOAP.

REFERENCES

[1] Berners-Lee, T., Fielding, R. and Frystyk, H., Hypertext transfer protocol – HTTP/1.0, 1996. RFC1945.

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H. and Berners-Lee, T., Hypertext transfer protocol – HTTP/1.1 (obsolete), 1997. RFC2068.

[3] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T., Hypertext transfer protocol – HTTP/1.1, 1999. RFC2616.

[4] Balachander, K. and Martin A., PROCOW: Protocol compliance on the web. Technical Report HA1630000-990803-05TM, AT&T Labs- Research, August 3 1999.

[5] Balachander, K., Jeffrey, C. and David, M., Key differences between HTTP/1.0 and HTTP/1.1. In Proceedings of the WWW-8 Conference, Toronto, May 1999.

[6] Balachander, K. and Jennifer, R., En passant: Predicting HTTP/1.1 traffic. In Proceedings of Global Internet Symposium, December 1999.

[7] Adam, D. B., Azer, B., Assaf, J., Safe Composition of Web Communication Protocols for Extensible Edge Services, In Proceedings of 7th International workshop on web content caching and distribution (WCW 2002), 2002.

[8] Gerard, J. H., Design and Verification of Protocols, Prentice hall, 1990.

[9] SOAP Version 1.2 Part 0: Primer, W3C Recommendation 24 June 2003, http://www.w3.org/TR/soap12-part0/

[10] SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation 24 June 2003, http://www.w3.org/TR/soap12-part0/

[11] SPIN and XSPIN tools, http://spinroot.com/spin/whatispin.html

[12] SOAP overview & evolution, http://xml.coverpages.org/soap.html

[13] Web-services information, Tome Clements, Overview of SOAP, Technical Article, Sun Microsystems,2002, http://java.sun.com/developer/technical/Articles/xml/webservices/