Towards Reliable Web Service Discovery through Behavioural Verification and Validation

Ervin Ramollari[1], Dimitrios Kourtesis[1], Dimitris Dranidis[2], and Anthony J. H. Simons[3]

[1] South East European Research Centre (SEERC)
17 Mitropoleos Str., 54624 Thessaloniki, Greece
{erramollari,dkourtesis}@seerc.org

[2] Computer Science Department, CITY Liberal Studies,
Affiliated Institution of the University of Sheffield
Tsimiski 13, 54624 Thessaloniki, Greece
dranidis@city.academic.gr

[3] Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
a.simons@dcs.shef.ac.uk

Abstract. Currently, the issues of trust and dependability on third-party Web services are becoming key challenges in the success of service oriented computing in industrial environments. One major obstacle in addressing these issues is that the current standard of Web service interface description (WSDL) neglects Web service behaviour, leading to incorrect bindings and unpredictable results. In this paper we address the behavioural facet by proposing an approach where the service provider augments the WSDL document with a formal model of the Web service behaviour, expressed in the stream X-machine (SXM) formalism. This model is both utilised by the service broker during publication to derive a test set and verify Web service behavioural conformance, and by the service consumer during discovery to perform service selection based on behavioural properties.

Key words: behavioural verification and validation, formal modelling, Web services, SOA, stream X-machines

1 Introduction

Service-oriented computing, which is based on the Service Oriented Architecture (SOA), is an emerging paradigm for development of distributed applications using powerful abstractions called *services*. The vision of SOA is to bridge the gap between the business and IT domains, to promote business agility, and to facilitate integration of heterogeneous systems within and across organizational boundaries. Currently, the most promising implementation alternative for SOA is the Web services framework, leveraging on widely accepted standards, such as WSDL, SOAP, and UDDI.

With the increasing number of Web services available over the Web today, the issues of trust and dependability on third-party providers are given increasing importance. Consumers need to ensure that advertised Web services satisfy their requests in multiple aspects and that they have verified implementations, before they are integrated into their systems.

One problem with the current standard for Web service interface description (WSDL) is that it lacks the support for semantic specifications, and the behavioural and non-functional (QoS) description facets. Therefore, it is not possible to guarantee that a service advertisement matches a service request in every aspect, which leads to incorrect bindings and unpredictable results.

The approach that we put forward in this paper addresses the behavioural facet, to assist in validation of service advertisements against consumer needs and in verification of behavioural conformance of service implementations. To model the behaviour of a Web service, we choose stream X-machines (SXMs) [15], which we believe to be a powerful and intuitive formalism for this purpose. The approach involves all three main actors in a SOA environment, i.e. the service provider, the service broker, and the service consumer. The provider's role is to create a SXM model reflecting the behaviour of the provided Web service implementation, and attach it to WSDL during the publication process. Based on this model, the broker is able to derive the necessary test cases, which are run in order to verify behavioural equivalence between the advertised model and the implementation. Only services with successful test results are accepted (i.e. checked-in) in the registry. On the other hand, during the discovery process, the consumer is provided with a number of service candidates fulfilling the request. Through the provided SXM models, the consumer can validate the behaviour of candidate services against consumer needs, a process that aids in the selection of the most appropriate service. Therefore, our approach ensures that clients bind with Web services providing a suitable behaviour, and a verified implementation.

The rest of this paper is structured as follows. *Section 2* presents a summary of related work. *Section 3* gives an overview of the proposed approach for reliable Web service discovery based on stream X-machines. *Sections 4, 5, and 6* focus on more detailed steps from the perspectives of the provider, the broker, and the consumer, respectively. The scenario of a shopping cart Web service is used to illustrate certain tasks, such as modelling with a stream X-machine and derivation of test sequences. *Section 7* concludes the paper by summarising the main points of the presented work, discussing strengths and limitations, and suggesting directions for future work.

2 Related Work

A number of approaches have been proposed for verifying Web services with model-based testing. The authors in [18] propose an algorithm, which translates Web service descriptions annotated in WSDL-S into an equivalent Extended Finite State Machine representation. The EFSM model is then exploited to generate an effective set of test cases to verify behavioural conformance of a stateful Web service.

Keum et al [13] present an approach based on EFSMs, a variant of finite state machines that has been extended with memory, as well as with computing blocks and predicate conditions for state transitions. A procedure is described for semi-automatically deriving the EFSM model from a WSDL specification and additional user input. The model covers behavioural aspects of stateful Web services, and the resulting test cases represent sequences of invocations of Web service operations. The authors provide experimental results showing that their method has the potential to find more faults compared to other methods, but notably, with a resulting test case set that is much larger and takes more time to execute.

Some other works have proposed the application of model-based verification of Web services in the context of more complete approaches. Bertolino et al [1] provide a framework where the provider augments the WSDL document with behavioural descriptions in a UML 2.0 Protocol State Machine (PSM) diagram, which is then translated to a Symbolic Transition System (STS). On the other hand, the broker utilises the attached STS model to automatically generate the test cases and run them on the provided Web service for behavioural conformance verification. Upon successful test results the Web service is published in the UDDI registry as a certified service. For this reason, the authors call their approach an "Audition" framework, where the Web service undergoes a monitored trial before being put "to stage".

Heckel and Mariani [8] describe a reliable Web service discovery approach, in which both the behaviour of the provided service and the consumer's requirements are modelled as graph transformation rules. A test case derivation method is employed to verify that the actual service implementation conforms to the provided model. This verification is performed by the service broker before services are accepted in the registry, resulting in what the authors refer to as high-quality service discovery agencies. In addition, the broker enables matchmaking of request and advertisement models that are expressed as graph transformation rules, in order to return service candidates satisfying the consumer's behavioural constraints.

One major weakness of the aforementioned approaches is that the test case derivation methods they use do not guarantee complete testing of the service implementations under test. In contrast, the X-machine functional testing algorithm that we employ is proven to generate a complete set of test cases that can reveal all faults in the implementation under test[9, 10]. In addition, the consumer role during service discovery and selection is substantially different in the proposed approach. The consumer no longer needs to specify a formal model of the service request to be matched with the advertised service models by the broker, since service selection is performed through behavioural validation at the consumer site. This is a significant advantage, given that it is impractical to assume that the consumer knows in advance the detailed behaviour of the requested Web service and can create a formal model of that behaviour.

3 Description of the Approach

The approach that we propose in this paper, as illustrated in Figure 1, involves all the three main participants in a SOA environment, i.e. the service provider, the service broker, and the service consumer. It is based on formally modelling behavioural aspects of Web services as stream X-machines, aiming to achieve formal verification and validation of Web services during the publication and discovery process. With behaviour of a service we refer to the interaction protocol that is assumed when the operations of that service are invoked in sequence, as well as the preconditions and effects of each operation. Behaviour is particularly relevant in stateful Web services, where responses of operations depend not only on the consumer input, but also on the internal state of the Web service, itself a result of previous operation invocations.

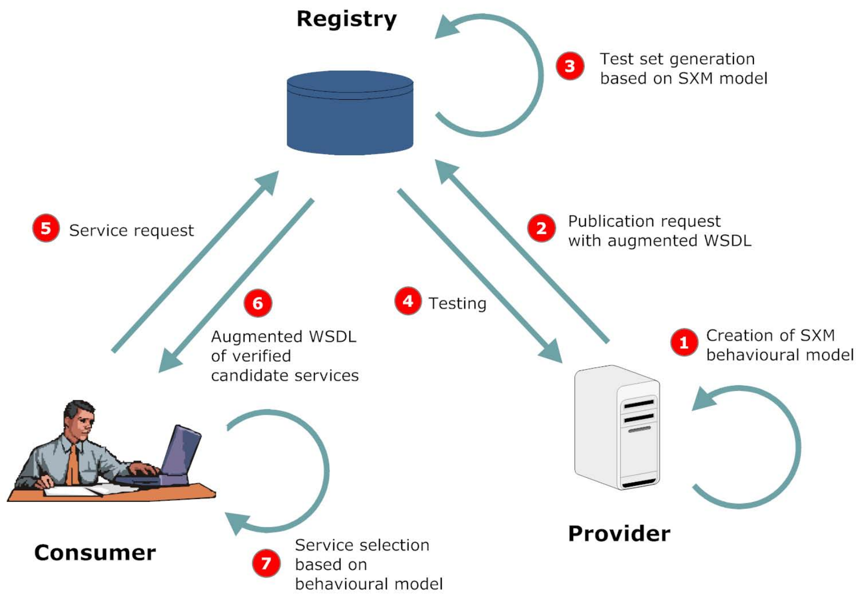
Registry
3 Test set generation based on SXM model
5 Service request
2 Publication request with augmented WSDL
6 Augmented WSDL of verified candidate services
4 Testing
1 Creation of SXM behavioural model
Consumer
Provider
7 Service selection based on behavioural model

Fig. 1. Service behavioural verification and validation approach based on stream X-machines.

The role of each participant in the scheme is fulfilled through multiple steps. The *service provider* creates a formal model reflecting the behaviour of the Web service that is to be published, using the stream X-machine (SXM) formalism [15, 9]. SXMs extend finite state machines (FSMs) with the addition of memory constructs and the application of processing functions during state transitions. The next section describes stream X-machines in more detail, and provides an example of creating the behavioural model of a stateful Web service. The SXM model, expressed in a markup language such as XMDL [11], is then added to the

WSDL document of the service. Practically, this may be achieved by adding an annotation, described in the SAWSDL semantic framework [7], which references the URI of the SXM markup document. The next step by the provider is the publication of the Web service to a service registry maintained by a broker. The publication query, which references the annotated WSDL document at the provider site, initiates the publication procedure at the broker site.

The role of the *service broker* is essentially to verify the behaviour of the provided Web service implementation through testing, and upon successful test results, to accept it in the UDDI registry. This step is necessary to ensure that the implementation of the provided Web service really conforms to the advertised behavioural specifications. It is possible that this might not be the case, either because of insufficient testing at the provider site, or because of malicious intent. With the attached SXM specification, the broker is able to derive the test sequences for verification automatically. The theory of complete functional testing from X-machines offers a method for deriving a complete, finite set of test cases, which is proven to find all faults in the implementation under test, given that certain realistic assumptions hold [10, 9]. This method is described in more detail in section 5 and illustrated with the provided scenario. The executable tests are then run by a testing engine that communicates with the Web service via SOAP messages. If the test results are successful, i.e. the expected and produced outputs match, then the Web service implementation has been shown to be free of faults with respect to the behavioural specifications. If this is the case, an Advertisement Functional Profile (AFP) of the Web service is created and added to the UDDI registry. The benefit of performing this procedure at the broker site, as opposed to performing it at the consumer site upon discovery, is that it needs to be done only once. Since only successfully tested Web services are accepted by the broker, consumers are ensured that the Web services they discover have been verified with respect to their specification.

The *service consumer*, as a first step during discovery, makes a query to the service registry in the form of a Request Functional Profile (RFP). The discovery procedure may be based on any existing approach, but semantically-enabled service discovery is encouraged, since it is free of ambiguity, takes more information into consideration, and results in better matches. As a response, the service broker returns a set of annotated service descriptions, which match the request according to the matchmaking algorithm that the registry implements. The consumer can take advantage of the SXM behavioural model that is provided with each service candidate, with the final aim of validating that the service behaviour satisfies the consumer needs and performing service selection. This is achievable in a number of ways, such as through model animation, model checking of properties, and, if the consumer has a SXM model of the required service, through state and transition refinement [17].

4 Provider's Perspective

As described in the approach, the provider's role is to create the behavioural model of the Web service to be published, add it as an annotation to the WSDL document, and initiate the publication process. This section focuses on stream X-machines and illustrates through a shopping cart scenario the procedure of modelling a Web service.

4.1 Stream X-machines

Stream X-machines (SXMs) [15, 9] are a computational model capable of modelling both the data and the control of a system. SXMs are special instances of X-machines introduced by Samuel Eilenberg in 1974 [5], which also have input and output streams. They employ a diagrammatic approach of modelling the control by extending the expressive power of finite state machines. In contrast to finite state machines, SXMs are capable of modelling non-trivial data structures by employing a memory, which is attached to the state machine. Additionally, transitions between states are not labelled with simple input symbols but with processing functions. Processing functions receive input symbols and read memory values, and produce output symbols while modifying memory values. The benefit of adding the memory construct is that the state explosion is avoided and the number of states is reduced to those states which are considered critical for the correct modelling of the system's abstract control structure. A divide-and-conquer approach to design allows the model to hide some of the complexity in the transition functions, which are later exposed as simpler SXMs at the next level.

A stream X-machine is defined as an 8-tuple, $(\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- Σ and Γ is the input and output finite alphabet respectively;
- Q is the finite set of states;
- M is the (possibly) infinite set called memory;
- Φ, which is called the type of the machine SXM, is a finite set of partial functions (processing functions) ϕ that map an input and a memory state to an output and a new memory state, $\phi : \Sigma \times M \rightarrow \Gamma \times M$;
- F is the next state partial function that given a state and a function from the type Φ, provides the next state, $F : Q \times \Phi \rightarrow Q$ (F is often described as a transition state diagram);
- q_0 and m_0 are the initial state and memory respectively.

Apart from being formal as well as proven to possess the computational power of Turing machines [9], SXMs have the significant advantage of offering a testing method [9, 10] that ensures conformance of an implementation to a specification, as discussed in the next section. In order to allow the markup of stream X-machines the XMDL (X-Machine Definition Language) language has been introduced in [11]. XMDL serves as a common language for the development of tools supporting stream X-machines [12]. An extension of XMDL to support an object-based notation was suggested in [3]. The object-based extension, called XMDL-O, enables an easier and more readable specification of Stream X-machines.

4.2 Shopping Cart Scenario

In modelling the behaviour of a Web service, the provider goes through data-level and behavioural-level analysis to derive the SXM model constructs. Some parallels can be drawn between a stateful Web service and a stream X-machine, given that both accept inputs and produce outputs, while performing specific actions and moving between internal states. SXM inputs correspond to request messages, outputs correspond to response messages, and processing functions correspond to operation invocations in distinct contexts. In addition, the service provider has to define the memory structure, not only as a substitute for internal state, but also to supply genuine test data that can become part of the generated test sequences.

The example we use to illustrate our modelling and verification approach is a simplified version of a Web service that provides the backend functionality of a shopping cart to client applications, allowing the client to perform authentication, add items to a shopping cart or remove them, and proceed to checkout. It consists of four operations: `login`, `addToCart`, `clearCart`, and `checkout`. The resulting SXM model is illustrated diagrammatically in Figure 2. For a more detailed description of this modelling example, including the definition of inputs, outputs, memory and processing functions, please refer to our previous work in [4].

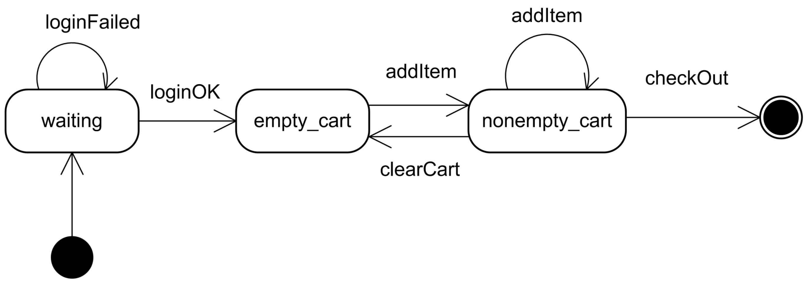

Fig. 2. Stream X-machine model of the shopping cart Web service

5 Broker's Perspective

This section describes the test cases generation method and the testing procedure to be performed by the broker on a Web service before being published. We aim to achieve automation of all broker activities, since the latter should be relieved of manual tasks.

5.1 Test Cases Derivation

A main benefit of modelling systems with SXMs is the existence of a test generation method, which under certain assumptions [10] is proven to find all faults in the implementation. The testing method is a generalization of the W-method [2]. It works on the assumption that the system specification and the implementation can be both represented as stream X-machines with the same type Φ (i.e. both specification and implementation have the same processing functions) and Φ satisfies the following design for test conditions: completeness with respect to memory (all processing functions can be exercised from any memory value using appropriate inputs) and output distinguishability (any two different processing functions will produce different outputs if applied on the same memory/input pair). More details about the derivation of the test sequences in the shopping cart scenario are given in [4].

An intermediate result of the test derivation method is a test set that consists of sequences of processing functions (labels), which have to be converted into sequences of inputs that will drive the application under test. This is achieved by the fundamental test function as described in [9]. For instance, the sequence of operations $\langle$`loginOK`, `addItem`, `addItem`$\rangle$ is converted into the sequence of inputs:

$$\begin{aligned}\langle\ &\texttt{loginRequest(“usr1”, “pwd1”)},\\ &\texttt{addToCartRequest(“912”)},\\ &\texttt{addToCartRequest(“875”)}\rangle\end{aligned}$$

Feeding these inputs to the SXM model and running a simulation produces the expected outputs, which are to be compared with the results of running the tests on the implementation.

5.2 Testing approach

With the availability of the test set from the SXMT method, the broker has to go through additional steps to test the behavioural conformance of the Web service implementation. The main steps are: *(i)* mapping the abstract input/output messages in the test cases to concrete messages in a Web service implementation under test, *(ii)* feeding the test cases to the respective Web service operations via a SOAP interface, and *(iii)* evaluating the service responses with respect to the expected output.

Since the inputs and expected outputs produced by the SXMT testing method are at an abstract level, they have to be mapped to concrete executable test cases that can be processed by a testing engine, in order to interact with the Web service under test and provide the results. A number of commercial and non-commercial Web service testing tools are available, such as SOAtest by Parasoft [16], and Coyote, described in [19].

For our testing approach to be applicable, we assume that the operations of the Web service under test follow the request-response pattern, i.e. they accept a request (input) message from the invoker and return a response (output) message. This makes it possible to fulfil the output distinguishable design for test condition, i.e. any two different processing functions should produce different outputs on each memory/input pair. As a result it is possible to tell from the outputs which processing functions have been activated during an execution path. We don't consider this assumption to be too restrictive, since request-response patterns are the normal case for Web service operations.

6 Consumer's Perspective

The consumer is the participant that benefits most from the described approach, since the consumer is able to discover and use Web services that match his own needs and that have a tested implementation with respect to behaviour. The consumer's role is to discover Web services from a service registry offered by the broker, using a discovery query, and to select a suitable one from a potential set of candidates. The whole procedure has to take into consideration, among other things, the behavioural facet, which is of high importance in ensuring correct matchmaking, binding, and interoperability. It is the provided SXM behavioural model that assists in this process.

There are two mechanisms that would allow a consumer to select the service candidate whose behaviour satisfies the consumer requirements. In one of them, the broker's service registry incorporates a behavioural matchmaking algorithm, in order to return service candidates with a matching behaviour during discovery. The consumer, on the other hand, includes behavioural constraints in the service request. The other mechanism, which is manual and involves only the consumer, is based on validating the behavioural models of returned service candidates against consumer requirements. The results of behavioural validation are used to filter service candidates and to perform service selection.

A method that enables behavioural validation is model animation with tool support. During animation, the user feeds the SXM model with sample inputs while observing the current state, transitions, processing functions, memory values, and last but not least, the outputs. One prolog-based tool supporting the animation of stream X-machine models is X-System [11], while a graphical Java interface on top of X-System, named JXSystem, is also available. In addition to animation, model checking techniques may be employed on the SXM model to check for desirable or undesirable properties, which are specified in a temporal logic formula. Research on X-machines already offers a model-checking logic, called XmCTL, which extends Computation Tree Logic (CTL) with memory quantifiers in order to facilitate model-checking of temporal properties in X-machine models [6]. Alternatively, if the consumer has a SXM model of the required service, this can be validated by state and transition refinement against the published SXM of the provided service [17].

7 Conclusions and Further Work

This paper describes an approach for reliable Web service publication and discovery through behavioural verification and validation, based on the stream X-machine formalism. It involves all three main participants in a SOA environment, i.e. the service provider, the service broker, and the service consumer. The service provider augments the WSDL document with a SXM behavioural model. The service broker is able to use the SXM model to derive a complete set of test cases that can verify behavioural conformance of the Web service implementation before it is accepted (checked-in) in the registry. On the other hand, the service consumer can use the provided behavioural model to perform service selection based on behavioural properties, using a number of techniques, such as model animation, model-checking, and, in case the consumer has a formal model of the requested service, model refinement techniques can be used.

The described approach is supported in fragments by a number of existing tools, which have been developed during previous research. However, several gaps exist in the required supporting infrastructure, and future research will address the consolidation of techniques and tools into a comprehensive application framework with industrial applicability. The main focus will be on the broker infrastructure, which requires more substantial work to support fully-automated Web service testing and, possibly, behavioural matchmaking. We have already developed a semantically-enabled UDDI service registry supporting the SAWSDL semantic framework, as part of the EU-funded STREP project FUSION [14]. The semantic UDDI registry is readily extensible to allow XMDL annotations in SAWSDL. In order to support automated Web service verification, we are planning to integrate the semantic registry with tools for test cases generation based on XMDL, and with capabilities for runtime testing of a Web service implementation under test. Additionally, in order to support behavioural matchmaking, we are planning to define an abstract query language for the service consumer and extend the current matchmaking algorithm of the UDDI registry to match the behavioural query with the advertised SXM models.

References

1. A. Bertolino, I. Frantzen, A. Polini, and J. Tretmans. Audition of web services for testing conformance to open specified protocols. *Architecting Systems with Trustworthy Components*, LNCS 3938, 2006.
2. T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.
3. D. Dranidis, G. Eleftherakis, and P. Kefalas. Object-based language for generalized state machines. *Annals of Mathematics, Computing and Teleinformatics (AMCT)*, 1(3):8–17, 2005.
4. D. Dranidis, D. Kourtesis, and E. Ramollari. Formal verification of web service behavioural conformance through testing. In D. Dranidis and I. Sakellariou, editors, *3rd South East European Workshop on Formal Methods*, pages 112–125. SEERC, November 2007.

5. S. Eilenberg. Automata, languages and machines. *Academic Press, New York*, A, 1974.
6. G. Eleftherakis, P. Kefalas, and A. Sotiriadou. Xmctl: Extending temporal logic to facilitate formal verification of x-machines. *Matematica-Informatica*, 50:79–95, 2002.
7. J. Farrell and H. Lausen. *Semantic Annotations for WSDL and XML Schema*. W3C Candidate Reccomendation, January 2007. Available at: http://www.w3.org/TR/sawsdl/.
8. R. Heckel and L. Mariani. Automatic conformance testing of web services. In *FASE 2005*, pages 34–48. Springer, 2005.
9. M. Holcombe and F. Ipate. *Correct Systems: Building Business Process Solutions*. Springer Verlag, Berlin, 1998.
10. F. Ipate and M. Holcombe. An integration testing method that is proven to find all faults. *International Journal of Computer Mathematics*, 63:159–178, 1997.
11. E. Kapeti and P. Kefalas. A design language and tool for X-machine specification. In *Proceedings of the 7th Panhellenic Conference on Information Techology, Greek Computer Society, Ioannina*, 1999.
12. P. Kefalas, G. Eleftherakis, and A. Sotiriadou. Developing tools for formal methods. In *Proceedings of the 9th Panhellenic Conference in Informatics*, pages 625–639, November 2003.
13. C. Keum, S. Kang, and I. Y. Ko. Generating test cases for web services using extended finite state machine. In *TestCom 2006*, pages 103–117. Springer, 2006.
14. D. Kourtesis and I. Paraskakis. Web service discovery in the fusion semantic registry. In *Proceedings of the 11th International Conference on Business Information Systems (BIS 2008)*, May 2008.
15. G. Laycock. *The Theory and Practice of Specification-Based Software Testing*. PhD thesis, Dept of Computer Science, Sheffield University, UK, 1993.
16. Parasoft. *SOATest Data Sheet*. www.parasoft.com.
17. A. J. H. Simons. A theory of regression testing for behaviourally compatible object types. *Software Testing, Verification, and Reliability*, 16(3):133–156, August 2006.
18. A. Sinha and A. Paradkar. Model-based functional conformance testing of web services operating on persistent data. In *TAV-WEB'06*, pages 17–22, Portland, Maine, USA, 2006. ACM.
19. W. T. Tsai, R. Paul, S. Weiwei, and C. Zhibin. Coyote: an XML-based framework for web services testing. In *Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02)*, 2002.