# Towards Automated WSDL-Based Testing of Web Services*

Cesare Bartolini[1], Antonia Bertolino[1], Eda Marchetti[1], and Andrea Polini[1,2]

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche
Via Moruzzi 1 - 56124 Pisa, Italy
[2] Dipartimento di Matematica ed Informatica
University of Camerino
Via Madonna delle Carceri, 9 - 62032 Camerino, Italy
{cesare.bartolini, antonia.bertolino, eda.marchetti,
andrea.polini}@isti.cnr.it

**Abstract.** With the emergence of service-oriented computing, proper approaches are needed to validate a Web Service (WS) behaviour. In the last years several tools automating WS testing have been released. However, generally the selection of which and how many test cases should be run, and the instantiation of the input data into each test case, is still left to the human tester.

In this paper we introduce a proposal to automate WSDL-based testing, which combines the coverage of WS operations with data-driven test case generation. We sketch the general architecture of a test environment that basically integrates two existing tools: soapUI, which is a popular tool for WS testing, and TAXI, which is a tool we have previously developed for the automated derivation of XML instances from a XML Schema.

The test suite generation can be driven by basic coverage criteria and by the application of some heuristics, aimed in particular at systematically combining the generated instance elements in different ways, and at opportunely varying the cardinalities and the data values used for the generated instances.

## 1 Introduction

Service-oriented Architecture (SOA) is the emerging paradigm for the development of distributed applications that are easy to integrate and flexible to fast changes of the environment and of user needs.

The escalation of Web Service (WS) technology is now evident to everyone. All major IT vendors, such as IBM, Tibco, Software AG, Oracle, just to cite the top competitors, have made huge investments into SOA in the last years.

Moreover service providers from virtually any domain, banks, governments, hospitals, academies, travel agencies, and so on, are progressively shifting towards the on-line service-market.

* The authors wish to thank Antonino Sabetta for his help in defining the test cases. This work was supported by the TAS[3] Project (EU FP7 CP n. 216287).

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 524–529, 2008.

The net effect of this IT technology trend is that unavoidably business and social welfare are more and more depending on the proper functioning of services delivered over the Net. WS trustworthiness is a modern buzzword to qualify those characteristics that allow a client to put justified reliance on a provided service, against accidental or intentional faults. Because of their pervasive distribution, WSs must offer very strict guarantees in this regard, even for services that are not dealing with safety-critical or money-critical applications.

For this reason, it is imperative that WSs are thoroughly tested before deployment. Essentially, a WS collects a set of functions, whose invocation syntax is defined in the associated WSDL document. The adoption of open standard specifications for the WS interface has been instrumental to achieve interoperability and is at the basis of several available testing tools. The WSDL formalized description of service operations and of their input and output parameters can be in fact taken as a reference for black box testing at the service interface. In the last years a wealth of WSDL-based WS test tools has been developed [5,8]. In general such tools can automatically derive skeletons of WS test cases and provide support for their execution and result analysis. Nevertheless they do not provide support for input data selection, for which they still rely on the human tester's intervention.

To us, it is somewhat surprising that till today WS test automation is not pushed further than this, since in principle the XML-based syntax of WSDL documents could support fully automated WS test generation by means of traditional syntax-based testing approaches. In this direction, we have defined a framework for "turn-key" generation of WS test suites[1], in which we combine coverage of WS operations (as provided by soapUI) with data-driven test case generation.

In this paper we illustrate the feasibility of the idea by means of a proof-of-concept implementation that integrates soapUI and TAXI. The latter is a tool we have previously developed [9] for the automated derivation of XML instances from a XML Schema. The idea, in comparison with soapUI and other existing WS test tools, is to derive from the WSDL interface of a WS, in a completely automated way, a test suite that thoroughly exercises the WS operations by systematically varying the possible input message structures and values.

The paper is structured as follows. In the next section we present the envisaged approach to fully automated WSDL-based testing; in Sec. 3 we then illustrate some feedbacks from our preliminary hands-on experience. Related work is briefly surveyed in Sec. 4 and conclusions are drawn in Sec. 5.

## 2 Approach

Our methodology aims at generating a set of SOAP messages sufficient to cover the whole interface provided by a WSDL file. Specifically, the tasks which must be carried out are:

[1] To be precise the test suites and test cases we derive only refer to input messages and data; the test oracle has still to be defined by the tester.

1. **WSDL Analysis:** A parser reads the WSDL specification and extracts information on operations, messages and data structures.
2. **SOAP Envelope Derivation:** Some tool is used to generate a skeleton of the SOAP message.
3. **Message Parts Definition:** For each data structure in the WSDL specification, different message instances are generated.
4. **Envelopes Composition:** The bogus data in the envelope skeletons are replaced with the actual derived instances.
5. **Messages Sending and Results Analysis:** The tester, or a batch script, sends the envelopes to the WS under test and collects the outputs for future inspection.

This methodology is eligible for combining different components to perform some of the above mentioned activities; in particular, we have selected soapUI and TAXI as two of them. The proposed architecture is sketched below.

The soapUI tool is responsible of the SOAP envelope skeleton derivation. TAXI is in charge of the actual message definition, and to do this it must extract the XML Schema data from the WSDL file (a modified version of the software or a preproduction script can be used for this purpose). The XML instances derived by TAXI and the envelope skeletons generated by soapUI can be assembled and sent to the WS. The results of the WS invocation are presented to the tester, or checked against provided expected output annotations (as done by soapUI). The whole process can be automated via a wrapping tool and the incorporation of suitable test strategies. We envisage that the generation of the SOAP messages can be carried out with various coverage criteria such as Operation Coverage, Message Coverage and so on, producing different degrees of detail.

### 2.1 soapUI

soapUI [5] is a tool developed by Eviware Software AB, available both in free and improved commercial versions. It assists programmers in developing SOAP-based web services. In particular, within the proposed methodology it allows the developer to generate stubs of SOAP calls for the operations declared in a WSDL file. Additionally, it is possible to use soapUI to send SOAP messages to the web service and display the outputs; this can be used for preliminary testing purposes.

Alternatively, for the purposes of this research it is possible to use any other tool capable of generating SOAP envelopes from WSDL files, such as Altova XMLSpy [1].

### 2.2 TAXI

TAXI (Testing by Automatically generated XML Instances) [3,4,9] is a tool able to generate XML instances compliant with a given XML Schema. It has been conceived so as to cover all interesting combinations of the schema by adopting a systematic black-box criterion. For this reason, TAXI applies the

well-known Category Partition (CP) technique [6] to the XML Schema. CP provides a stepwise intuitive approach to identify the relevant input parameters and environment conditions and combine their significant values into an effective test suite.

TAXI activity starts with the analysis of an input XML Schema. The implementation of CP requires the analysis of the XML Schema and the extraction of the useful information.

***choice*** elements are processed by generating instances with every possible child. Multiple *choice* elements produce a combinatorial number of instances. This ensures that the set of sub-schemas represents all possible structures derivable from *choice*.

**Element occurrences** are analyzed, and the constraints are determined, from the XML Schema definition. Boundary values for *minOccurs* and *maxOccurs* are defined.

***all*** elements result in a random sequence of the *all* children elements for generating the instance. This new sequence is then used during the values assignment to each element.

Exploiting the information collected so far and the structure of the (sub)schema, TAXI derives a set of intermediate instances by combining the occurrence values assigned to each element.

The final instances are derived from the intermediate ones by assigning values to the various elements. Two approaches can be adopted: values can be picked from an associated database or generated randomly if no value is associated to an element in the database. Since the number of instances with different structures could be huge, in the current implementation TAXI only selects one value per element for each instance.

## 3 Preliminary Evaluation

To measure the feasibility and strength of the proposed approach, our methodology has been trialed for testing a WS which queries a publications database.

Using the WSDL available description we derived systematically a test suite along the steps presented in Sec. 2, and we compared it against a manually generated one, mimicking a human tester using the soapUI tool.

Both test suites consisted exclusively of XSD-compliant messages, and included both data actually taken from the publications database, and fictitious names or keywords. The two test suites have been used for testing the web services and the results have been collected. The first obtained feedbacks made clear that the manual test suite completely ignored certain classes of problem of the tested WS, while they evidenced a good performance of our approach in finding more problems.

In particular this experiment gave us the opportunity to detect bugs which had not popped out before. These errors were related to some parameters which were not passed to the search function.

The in-use version of the web service software had been thoroughly tested and is "bug-free enough" for ordinary use, while the version used for this experiment contains several improvements which still have to be integrated into the in-use application. Several manual tests had been previously run against the new features, but they were not sufficient to highlight the errors we found with a proof-of-concept of our methodology.

In conclusion, the experiments showed out that our systematic automated approach can provide a test suite which is more effective than the one which is created manually. In particular, having a test suite which covers such a wide range of variability in the structure and the values of the data would require a huge effort if done manually, even starting from a basis of automatically generated skeletons such as those provided by soapUI. Even though we have not yet performed a formal benchmark evaluation, the effort and time required appear drastically reduced using this methodology.

## 4 Related Work

There is today a list of good tools that can be used and have been adapted to test web services. Just to mention a few interesting ones, there are soapUI [5], PushToTest [8], SOATest [7]. Tipically such tools are extremely effective in supporting the various testing activities and in increasing the productivity of testers. Nevetheless they mainly focus on management and execution of test cases and none of them tries to automatically provide test design and generation. Such a step is still mainly on the shoulder of testers and is strongly related to their ability. In particular none of the tools we analyzed take advantage of the availability of service models expressed in computer readable format suitable for automatic manipulation. In particular we refer here to the availability of XML-based description of service operation data models.

To the best of our knowledge, the only work which addresses issues similar to ours is [2], which also proposes XML-based test data generation and test operation generation. However, the work only outlines the possible perspectives of WSDL-based testing, but does not provide a tool, nor does it rely on standard test approaches. In our approach, instead, we intend to offer a "turn-key" tool which, by exploiting the existing TAXI tool, focuses on a systematic generation of test cases based on the Category Partition algorithm.

Finally, automatic generation of instances from XML Schemas its nowadays a feature of some, even commercial, tools [1,10]. Therefore our approach could be pursued even using other XML instance generation tools.

## 5 Conclusions and Future Work

Testing of Web Services is a challenging activity. Many characteristics (runtime discovery, multi-organization integration) of this new paradigm and its related technologies certainly contribute to make testing much more difficult. Nevertheless there are other characteristics that could be fruitfully exploited

for testing purposes. Among these, the representation of data in a computer readable format (typically XML-based) facilitates the automatic derivation of data instances to be used for testing invocations.

Starting from this consideration we presented a methodology to automatically derive test messages from WSDL descriptions. Such messages include data representing possible values that a real implementation of the service should be able to handle. We proposed that the generated data instances are encapsulated in correct SOAP envelopes that can be used to invoke a service implementation. Furthermore, by use of our tool TAXI, we proposed to exploit the characteristics of an XML Schema-based data description to automatically apply well known testing methods such as Category Partition and boundary value selection. This would result in the derivation of a test suite of messages that are representative of the space of possible messages.

The described methodology is still undergoing development and validation. Main tasks for the future include: To perform focussed and extensive evaluations in order to identify fault categories that are easily discovered and better define the usage scope; to extend the embedded TAXI functionalities so as to also generate non-compliant test cases that can support robustness testing of the invoked service; to complete the approach implementation and make it available as a free tool to the community for download and experimentation.

## References


1. Altova. XML Spy, http://www.altova.com/products/xmlspy/xml_editor.html
2. Bai, X., Dong, W., Tsai, W.-T., Chen, Y.: WSDL-based automatic test case generation for web services testing. In: Proc. of IEEE Int. Work. SOSE, Washington, DC, USA, pp. 215–220. IEEE Computer Society, Los Alamitos (2005)
3. Bertolino, A., Gao, J., Marchetti, E., Polini, A.: Systematic generation of XML instances to test complex software applications. In: Guelfi, N., Buchs, D. (eds.) RISE 2006. LNCS, vol. 4401, pp. 114–129. Springer, Heidelberg (2007)
4. Bertolino, A., Gao, J., Marchetti, E., Polini, A.: Automatic test data generation for XML Schema based partition testing. In: Proc. Int. Work. on Automation of Software Test (ICSE 2007 companion), Minneapolis, Minnesota, USA (May 2007)
5. Eviware. soapUI; the Web Services Testing tool (accessed May 30, 2008), http://www.soapui.org/
6. Ostrand, T.J., Balcer, M.J.: The category-partition method for specifying and generating fuctional tests. Commun. ACM 31(6), 676–686 (1988)
7. Parasoft. SOATest (accessed June 3, 2008), http://www.parasoft.com/jsp/products/home.jsp?product=SOAP
8. PushToTest. PushToTest TestMaker (accessed June 3, 2008) http://www.pushtotest.com/Docs/downloads/features.html
9. TAXI. Testing by automatically generated XML instances (2007), http://labse.isti.cnr.it/index.php?option=com_content&task=view&id=94&Itemid=49
10. Toxgene. Toxgene (2005), http://www.cs.toronto.edu/tox/toxgene/