Department of Computer Science
San Marcos, TX 78666

Report Number TXSTATE-CS-TR-2009-13

Automatic test case generation for web service processes using a SAT solver

Karthikeyann Radhakrishnan
Rodion Podorozhny

2009-02-16

Automatic test case generation for web service processes using a SAT solver

Karthikeyann Radhakrishnan Rodion Podorozhny
Texas State University – San Marcos
Department of Computer Science
{rp31, kr1215}@txstate.edu

Abstract

Such useful properties of web services as access from any platform, great interoperability with other web services, ability to combine several web services into a larger application relatively quickly have made them an important category of software systems. One of the techniques used to increase the quality of software is testing. The adequacy of test cases and possible automation of the testing process greatly influence the quality of the produced software and timeliness of the software development process. Even though a great deal of work has been done in adapting test case generation techniques to the peculiarities of web services (e.g. [11][12][13]) we believe our work makes a useful contribution in this area.

This paper proposes a novel approach to generate test cases based on the process definition model of a web service. A process definition model defines a sequence of activities that can be performed by orchestrating the capabilities of a web service. A SAT solver (such as Alloy [10]) is used to extract the paths from the process definition model. These paths are used to generate test case specifications that will test all web service capabilities involved in a process.
In our opinion the main contribution of the work is an application of a static analysis method for generation of test cases for a web service guided by a goodness metric of process coverage.

1. Introduction

A great deal of attention has been paid to development and analysis of web services as they gain widespread use. Some of the reasons for web services popularity include their ability to provide access to widely distributed computational capabilities from any platform and provide fast and reliable integration of software systems not originally intended to interoperate with one another as compared to creation of a dedicated system of the same purpose.

Some of the high level goals of software engineering include invention of methods for software analysis that increase the software system's quality and increase the productivity of a software development process. Web services differ from traditional standalone software systems. Thus, a direct reapplication of analysis methods to web services is not always reasonable. At the same time there is still a great need in development of new web services and ways to integrate the existing ones. Thus, we need to find more cost effective methods for web service design and analysis that deliver highly reliable solutions.

One of the major stages of a software development is verification and validation. The automation of this step can bring multiple benefits: it is possible to reduce the amount of time for this stage and reduce the effect of human error. In particular, one of common software analysis methods is testing. Automation of test case generation can ensure adequacy of test case suits. The automation of this stage can be made possible due to systematic methods of test case generation and application. It is such a novel method that is suggested in this work.

2. Motivation

One of the common applications of web services that has a very strong influence on our society lies in the e-commerce problem domain. We would like to illustrate the suggested method on the motivating example of buying a book from an online provider such as amazon.com borrowed from [1][2]. The book buying process using web services based on the amazon.com site is illustrated in Fig. 1.

The figure uses a simple notation to represent the process. Nodes correspond to process steps and numbered arcs correspond to functional decomposition. Children of Choice process steps correspond to alternatives. The notions of process and process steps are used as in [3][4]. A process step can correspond to an action by a human, a human assisted with a web service capability or some (external) software system. Based on these notions, there are three types of process steps in this model. They are Choice steps, Sequence steps, and Condition process steps. Thus, arcs originating from a Choice step correspond to mutually exclusive outcomes of the Choice step. Sequence step correspond to sequentially going thru all the child process steps in a sequence from left to right. The arcs originating from a Condition step correspond to

mutually exclusive outcomes of this step based on satisfaction of a condition in the data values in a process. The data flow in the notation in Figure 1 occurs along

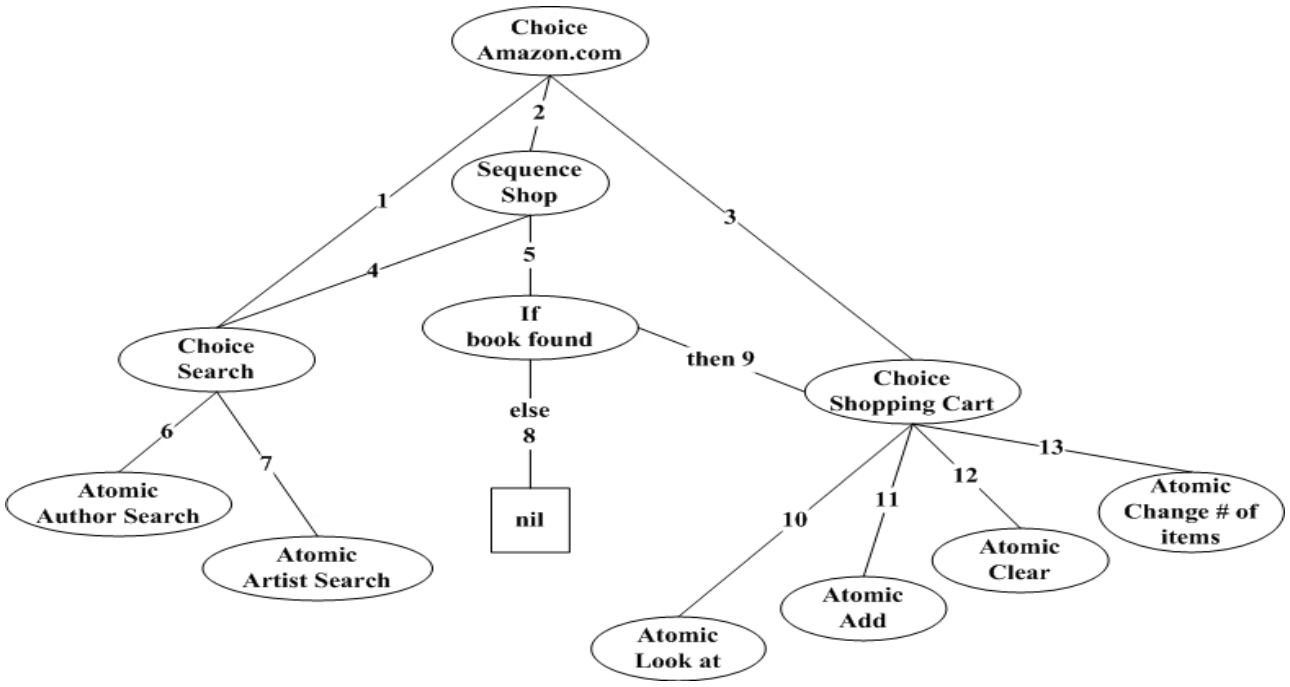

Figure 1: Amazon's book buying process model [1][2]

the arcs too. Input flows from a parent step to an immediate child step and output flows vice versa. Leaf nodes represent atomic process steps. Atomic process steps are fundamental units of action in the process definition that cannot be decomposed any further. Web service capabilities [5] of the amazon.com book buying web service are invoked at process steps by going along various paths in the process definition. A number of process instances can traverse the same path [3]. A path can have many process instances going through it because each process instance can have different data values. The amazon.com book buying process model shows the following capabilities of amazon.com book buying web service: (cf. Figure 1)

1. Search by author: Capable of searching for books by an author name.
   Capability correspond to step: *Author Search*
   Precondition: None
   Postcondition:
   a. List of books are either found or none are found

Definition of process instances that use this capability: Set of all process instances that are forced go through the path that contain Choice step *Amazon.com*, Choice step *Search*, and Atomic step *Author Search* in that order.

Execution constraints: None

2. Search by artist: Capable of searching for CD's by artist name
   Capability correspond to step: *Artist Search*
   Precondition: None
   Postcondition:
   a. List of CD's are either found or none are found

   Definition of process instances that use this capability: Set of all process instances that go through the path that contain Choice step *Amazon.com*, Choice step *Search*, and Atomic step *Artist Search* in that order.
   Execution Constraint: None

3. Add items (books or CD's) to shopping cart: Capable of adding items to a user's shopping cart.
   Capability correspond to step: *Add*
   Precondition:
   a. The user is signed in.
   b. There are sufficient quantities of the item in stock.

   Postcondition:
   a. The item is added to the cart.
   b. The stock counts in amazon.com for the item is reduced by however many items were added to the cart.

   Definition of process instances that use this capability: Set of all process instances that go through the path

that contain Choice step *Shopping Cart*, and Atomic step *Add* in that order.
Execution constraints: The Search by Author or Search by Artist capability must have been invoked at least once with a result containing non-empty list of items.

4. View shopping cart: Capable of viewing items in the user's shopping cart
   Capability correspond to step: *Look At*
   Precondition:
   a. The user is signed in.
   Postcondition:
   a. All the items in the shopping cart are displayed
   Definition of process instances that use this capability: Set of all process instances that go through the path that contain Choice step *Amazon.com*, Choice step *Shopping Cart*, and Atomic step *Look At* in that order.
   Execution Constraints: None

5. Clear shopping cart:: Capable of clearing specific or all items from the shopping cart
   Capability correspond to step: *Clear*
   Precondition:
   a. The user is signed in.
   b. The shopping cart is not empty
   Postcondition:
   a. The items cleared are not in the shopping cart.
   b. The stock count of cleared items in amazon.com is incremented with the number of items that are cleared from the shopping cart.
   Definition of process instances that use this capability: Set of all process instances that go through the path that contain Choice step *Amazon.com*, Choice step *Shopping Cart*, and Atomic step *Clear* in that order.
   Execution constraints: The *Add items to shopping cart* capability has been invoked at least once before invoking this capability.

6. Change number of items: Capable of editing items in the shopping cart.
   Capability correspond to step: *Change # of items*
   Precondition:
   a. The user is signed in.
   b. The shopping cart is not empty
   Postcondition:
   a. The items cleared are not in the shopping cart.
   b. If the count of an item in the cart is increased,
      i. There are sufficient quantities of the item in stock
      ii. And, the count of the item in the inventory decreases by that amount
   c. If the count of an item in the cart is decreased, the count of this item in stock is increased by that much.
   Definition of process instances that use this capability: Set of all process instances that go through the path that contain Choice step *Amazon.com*, Choice step *Shopping Cart*, and Atomic step *Change # of items* in that order.
   Execution constraints: The *Add items to shopping cart* capability step has been invoked at least once before invoking this capability.

7. Shop for books and CD's (i.e., Search items and Manage shopping cart): This capability is further decomposed into capabilities in steps that already exist. It is a sequence of steps where the first step is a capability from the capability set {1, 2}, followed by the Condition step *If*, followed by a step having a capability from the capability set {3, 4, 5}. A capability set is a set containing some or all capabilities 1 to 6 mentioned above.
   Capability correspond to step: *Shop*
   Precondition:
   a. The user is signed in.
   Postcondition:
   a. The post condition of the process instances after implementing this capability will be the same as the capability in the last step that was implemented in the sequence
   Definition of process instances that use this capability: Set of all process instances that traverse through a sequence of desired capability steps chosen such that one capability at a step has been picked from capability set {1, 2}, followed by the Condition step *If*, followed by a step containing another capability picked from the set {3, 4, 5}.
   Execution constraint: For a sequence of capabilities selected, every invocation of capability 5 at a step is preceded by a step invoking 3; 3 is preceded by a step invoking 1 or 2.

3. Use scenarios of motivating example

The execution constraints of a process for a capability described above allow the process to prescribe paths a user must take to invoke these capabilities. For example, the following process definition corresponds to given execution constraints.

First, a user invokes the *Choice Amazon.com*. Next, the process prescribes the execution of *Choice Search* which has two alternatives; either by *Author Search* or *Artist Search*. One possible process definition that satisfies the execution constraints for this capability is search by author as depicted in Figure 1. Let us say, a user intends to search for a book by author. The user has to do a series of steps to perform this search. The actor in a step is a user who is a human assisted with a web

service capability. First, the user goes to the Amazon.com website. The user at this step is assisted by a web service capability *Choice Amazon.com*. Then, the user chooses to do a search and reaches a step containing *Choice Search* web service capability. At this step, the step prescribes a list of alternatives to the user to choose. The alternatives are search by author or search by artist. The user chooses search by author from the list. The user is directed to a step containing the capability *Atomic Author Search* where he provides the name of the author to search and executes the search. The user is returned with a non-empty list of books if there are books associated with the author or an empty list if no books exist for the given author.

Since a possibly infinite number of process instances can go through a single path even a large number of test cases do not guarantee that various capabilities of a web service are tested. To achieve a greater confidence in the assurance level of testing, the test cases should be selected in a systematic way. There also must be a way to evaluate the "goodness" of the selected set of test cases. One possible approach to achieve this for web services is to select test cases based on the coverage of the process that orchestrates application of web service capabilities to accomplish a certain goal. The internal details of web service are quite often unavailable so direct approach of coverage criteria to source code is not reasonable. In addition the sheer size of web service implementations can make such a direct approach infeasible. Process definitions for web services are usually rather small (under 20 activities). Thus application of coverage criteria to them to derive test cases is feasible. Also, such testing is beneficial because it can detect critical failures of a web service.

It even might be feasible to test all paths in the process definition, but it may not be useful to test all paths because some paths may be impractical. For example, generating test cases that try to test the negative outcomes of a search might be meaningless. If there is no systematic way to test the paths in the web service process model, a great number of test cases can hit the same path and never exercise the capabilities that occur in other paths. For example, we might have test cases that test *clearing the cart* capability or *adding to a cart* capability. These test cases might show that both adding to cart and clearing the cart work as expected when the capabilities are tested independently. But, we might never have generated a test case that would try to test a sequence of capability invocations corresponding to clearing the shopping cart and adding items into the shopping cart and clearing them again. In the absence of such a test case, we will not be able to check the correctness of clearing the cart in different use scenarios where adding items to the cart in some other path might affect the way a cart is cleared. Hence, we need a more systematic way to select test cases that can be targeted along specific paths to test a set of functionalities in an organized way. The benefits of systematically generating test cases are:

1. Higher level of assurance that all the web service capabilities and the combinations of their invocations that correspond to typical usage patterns are exercised.
2. Higher level of automation in both creation and application of test cases.

The higher level of automation is especially useful in systems that mash-up and/or compose web services dynamically in response to user queries. Such an automated test case generation method can be used to evaluate functional correctness of a web service composition promptly.

4. Test case generation using SAT solver

The approach for generating test cases to test web service capability processes is as shown in Figure 2. The steps of this approach for manually generating test cases for amazon.com book buying process model is as explained below. The test cases may then be used in a keyword-driven based web service testing framework based applications for book buying web services.

Let us describe an example of application of this approach for searching a book on Amazon's book buying model. A set of test cases is generated using a pair of execution constraint and a path constraint. Execution constraints specify the constraints on all process instances along a path. The path constraint mentions the start and end process steps along a path. The steps for generating test cases using a set of execution constraints and path constraints are as follows:

Step 1: The amazon.com book buying web service capability process definition is obtained in a well known specification format like OWL-S or WSMO [5]. Example of a pre-condition for using the process definition is that the user must be authenticated/ to buy a book.

Step 2: The following are some execution constraints and the start and end of paths that we desire to examine:

Execution constraint (EC1): Choice step *Search* precedes Atomic step *Author Search*. Choice step *Amazon.com* precedes Choice step *Search.*

Path constraint (PC1): Start at process step *Amazon.com*. End at process step *Search book*

We will generate test cases using the pair EC1 and PC1.

Step 3: Model the execution constraint EC1 using Alloy tool.

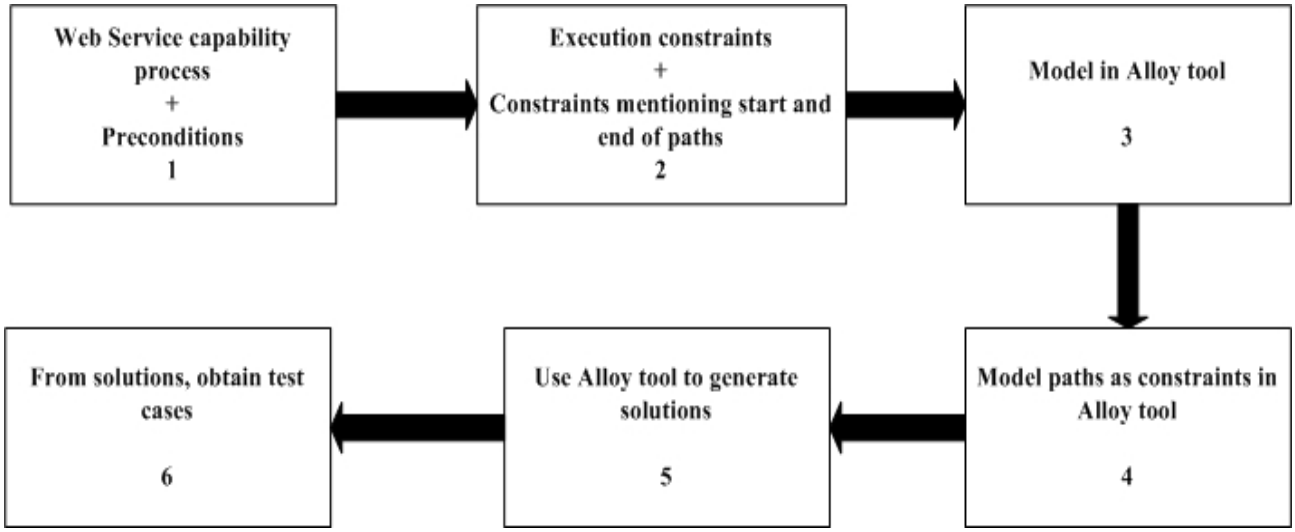

Figure 2: Test case generation for web service capability process using Alloy SAT solver

Step 4: Model the path constraint PC1 as constraints in Alloy tool
Step 5: With the above constraints, Alloy will generate solutions within the scope of the constraints where a solution is a path that satisfies constraints EC1 and PC1.
Step 6: We obtain the test cases from this set of solutions because a number of test cases can force a process instance to traverse the same path

In the case of testing a web service process, a test case is a sequence of interactions between a user and web service capabilities rather than an input to the first step of a path. These interactions correspond to a series of request/response between the user and process instance steps along the generated paths. For example, a test case Testcase1 generated for searching a book capability from a solution for EC1 and PC1 based on Figure 1 might correspond to a set of request/response between the user and the web service as below:

1. The user performs a **request** to access *Amazon.com* web service.
2. The process instance step *Choice Amazon.com* **responds** by granting access to the user.
3. The user performs a **request** to make a search. The process instance directs the user to step *Choice Search* along arc 1.
4. The process instance step *Choice Search* **responds** by prescribing a list of search options.
5. The user performs a **request** to search by an author.
6. he process instance **responds** by directing the user to step *Atomic Author Search* along arc 6.
7. The user makes a search **request** by entering a particular query bookByAuthorQuery1. Assuming that the website's database does contain entries that correspond to that query (i.e. books by the author whose name was entered), the process instance step *Author Search* **responds** with a result containing a non-empty list of books published by the author.

In the above interaction, the process steps *Amazon.com, Search, and Author Search* along a path is defined according to PC-EC1 constraint. The query bookByAuthorQuery1 is a designation given to a set of entries into the fields of the *Author Search* webpage. The user interacts at each process instance step before reaching the next process instance step in the path. The interaction proceeds to the next step only if the current interaction resulted in a successful response. The test cases can be annotated with the expected results before using them in a testing application. Below, we have two more test cases Testcase2 and Testcase3 that test the manipulation of the shopping cart and their expected results.

We make the following assumptions:

1. Only one book can be selected and added to the shopping cart at a time.
2. The books for the authors we are looking for already exist in Amazon.com stock and will be always found in the search. We assume the results of the search queries will contain these 3 books: Book1 with ISBN

T

1234, Book2 with ISBN 5678 and Book3 with ISBN 8765.

Expected results for Testcase2 and Testcase3:

We expect Book1 with ISBN 1234 and Book3 with ISBN 8765 to be in the shopping cart after executing these test cases. This expectation is based on the intended behavior of the website capabilities.

Testcase2 for checking cart manipulation capabilities is presented below (cf. Figure 1 for a process definition):

1. The user performs a **request** to access *Amazon.com* web service.
2. The process instance step *Choice Amazon.com* **responds** by granting access to shopping options and cart manipulation to the user.
3. The user performs a **request** to start shopping. The process instance directs the user to step *Sequence Shop* along arc 2.
4. The process instance step *Sequence Shop* **responds** by directing the user to step *Choice Search* along arc 4.
5. The user performs a **request** to make a search.
6. The process instance step *Choice Search* **responds** by prescribing a list of search options.
7. The user performs a **request** to search by an author. The process instance **responds** by directing the user to step *Atomic Author* Search along arc 6.
8. The user makes a search **request** by querying for books by an author name (bookByAuthorQuery1). The process instance step *Author Search* **responds** with a result containing a list of books (Book1, Book2, Book3) published by the author.
9. The user **requests** to select a particular book, book1 with ISBN 1234.
10. The process instance step *Author Search* selects book1 and the process instance directs the user to step *Choice Shopping Cart*.
    [The process instance directs the user by returning to step *Sequence Shop* by traversing along arc's 6 and 4. Then, process instance traverses to step *If book found*. Since, the user found a book, process instance traverses to step *Choice Shopping Cart* along arc 9.]
11. The process instance step *Choice Shopping Cart* **responds** by prescribing a list of options to manipulate the shopping cart.
12. The user **requests** to add book1 with ISBN 1234 to shopping cart.
13. The process instance step *Choice Shopping Cart* **responds** by directing the user to step *Atomic Add* along arc 11.
14. The user submits a **request** to add book1 to shopping cart.
15. The process instance step *Atomic Add* **responds** by adding book1 to the shopping cart. The process instance returns to step *Choice Amazon.com by traversing along* arc's 11 and 3.

Steps 1 thru 14 are repeated to add another book , book3 with ISBN 8765, to the shopping cart. A different query is used this time, bookByAuthorQuery3. We can add a few more interactions to this test case to lookup the shopping cart.

16. The user **requests** to manipulate the shopping cart. (Note that at this point, the user is at process instance step *Choice Amazon.com*). The process instance step *Choice Amazon.com* directs the user to step *Choice Shopping Cart* along arc 3.
17. The process instance step *Choice Shopping Cart* **responds** by prescribing a list of options to manipulate shopping cart.
18. The user **requests** to look at the shopping cart. The process instance step *Choice Shopping Cart* directs the user to process step *Atomic Look at* along arc 10.
19. The process step *Atomic Look at* **responds** by showing the list of books in the shopping cart.

We expect to see book1 with ISBN 1234 and book3 with ISBN 8765 in the shopping cart as expected result for this test case.

Similarly, we can get the same expected result by a different sequence of manipulations of the shopping cart using another test case, Testcase3:

Include steps 1 to 15 from Testcase2 (we do not repeat them here for space concerns). Repeat steps 1 to 15 three times to add Book1 with ISBN 1234, Book2 with ISBN 5678 and Book3 with 8765 to the shopping cart using the corresponding search queries for different authors (named bookByAuthorQuery1, bookByAuthorQuery2, bookByAuthorQuery3). Include these steps:

16. The user **requests** to manipulate the shopping cart. (Note that at this point, the user is at process instance step *Choice Amazon.com*). The process instance step *Choice Amazon.com* directs the user to step *Choice Shopping Cart* along arc 3.
17. The process instance step *Choice Shopping Cart* **responds** by prescribing a list of options to manipulate shopping cart.
18. The user **requests** to clear items in the shopping cart. The process instance step *Choice Shopping Cart* directs the user to process step *Atomic Clear* along arc 12.
19. The process step *Atomic Clear* **responds** by showing the list of books in the shopping cart.
20. The user **requests** to clear Book2 with ISBN 5678 from the cart. The process instance step *Atomic*

Clear **responds** by clearing Book2 from the shopping cart and return to the process step *Choice Shopping Cart* along arc 12.

21. The process instance step *Choice Shopping Cart* **responds** by prescribing a list of options to manipulate shopping cart.
22. The user **requests** to look at the shopping cart. The process instance step *Choice Shopping Cart* directs the user to process step *Atomic Look at* along arc 10.
23. The process step *Atomic Look at* **responds** by showing the list of books in the shopping cart.

We expect to see book1 with ISBN 1234 and book3 with ISBN 8765 in the shopping cart as expected result for this test case which is the same as the expected result of Testcase2.

5. Related work

While previously there have been attempts to generate test cases for webservices automatically [7][8][9], to our knowledge it is the first approach that suggests the use of a SAT solver. In [7], Tsai et. al proposes test case generation based on web service specifications only, while their focus has been in establishing trustworthiness of webservices using Boolean expression analysis. Hanna & Munro in [8] generate test cases based on XML schema data types found in input data parameters in WSDL specifications of webservices. They do this by performing boundary value analysis on the data ranges of data types in WSDL specifications. In [9], Deutsch et al claim to show the effectiveness of verifying properties like soundness of specifications (like what web page should be displayed next) and semantic properties (like payment cannot be made when cart is empty) of data-driven web applications in general by combining model checking and database optimization techniques.

From a review of a number of approaches to the problem of webservice verification ([2][7][8][9]) it is possible to summarize that these approaches range from static analysis techniques such as model checking to automatic generation of test cases either based on specification or webservice definition. The goodness metrics usually fall into categories of coverage of behavior or coverage of specification. The technical feasibility of the reviewed static analysis methods is achieved by bounding the sets of values assigned to variables of the corresponding webservices definitions. The feasibility of the dynamic analysis methods is achieved by using constraints on input values as described in the requirements. Quite often constraints are derived from boundary value analysis.

6. Conclusion

Thus, we have illustrated an example of generating test cases according to the suggested method. The paths for the test cases to traverse can be determined automatically by a SAT-solver such as Alloy. A coverage metric can be calculated to show the "goodness" of the set of paths. Initially, the test cases can be generated manually in a systematic manner based on these paths. One of the future work directions can focus on automated techniques such as symbolic execution so that to generate test cases automatically (based on the paths). Even without automatic generation of test cases based on a path, this approach is beneficial because it delivers paths satisfying a certain coverage criterion. The benefit of automation, even at the stage of path generation, is likely to reduce testing time and make this approach useful for testing webservice compositions. Furthermore, we suggest combining this approach with an existing automatic webservice testing framework. This framework generates test cases based on the key word approach [16] that is not guided by a coverage metric. A prototype of this framework was developed during summer 2008 internship at Akamai Technologies.

The main contribution of the work compared to the related approaches described in the related work is an application of a static analysis method for generation of test cases for a webservice guided by a goodness metric of process coverage. We believe this work is the first application of such a test case generation approach to webservices even though similar approaches have been used for other kinds of software systems [14][15].

7. References

[1] DAML Services: http://www.daml.org/services/swsl/straw-proposals/OWL-S-Straw-Amazon.ppt . Retrieved: November 25, 2008.

[2] Ankolekar, A., Paolucci, M., Sycara, K. (2004). Spinning the OWL-S Process Model: Toward the Verification of the OWL-S Process Models. *In Proc. of ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of BusinessApplications, 2004*. http://www.ai.sri.com/SWS2004/final-versions/SWS2004-Ankolekar-Final.pdf . Retrieved: November 25, 2008.

[3] L. Osterweil. (1987). Software *Processes are Software Too*. *ICSE*.

[4] Cass, Aaron G., Lerner, Barbara S., McCall, Eric K., Osterweil, Leon J., Sutton, Stanley M., Wise, Alexander. (2000). Little-JIL/Juliette: A process definition language and interpreter. *ICSE*.

[5] Dumitru Roman et. al. (2005). Web Service Modeling Ontology. *Applied Ontology archive, Volume 1 , Issue 1, pp 77-106.*

[6] The ASTOOT approach to testing object-oriented programs, 1994.

[7] Tsai, Wei-Tek., Chen, Yinong., Paul, R. (2005). Specification-based verification and validation of Web services and service-oriented operating systems. *International Workshop on Object-Oriented Real-Time Dependable Systems*. pp. *139- 147*.

[8] Hanna, S., Munro, M. (2007). An approach for specification-based test case generation for Web Services. *International conference on Computer Systems and Applications*. *pp. 16-23.*

[9] Deutsch, Alin., Marcus, Monica., Sui, Liying., Vianu, Victor., Zhou, Dayou. (2005). A verifier for interactive, data-driven web applications. *International Conference on Management of Data. pp539-550.*

[10] Alloy Analyzer: *http://alloy.mit.edu/community/ Retrieved: February 2nd 2009.*

[11] Tsai, W.T., Wei, Y. Chen., Xiao, B., Paul, R. 2005. Developing and assuring trustworthy web services. *International Symposium on Autonomous Decentralized Systems, 2005. pp 43-50.*

[12] Tsai, W.T., Wei, Y. Chen., Paul, R., Xiao, B. 2005. Swiss Cheese Test Case Generation for Web Services Testing. *IEICE - Transactions on Information and Systems. pp 2691 – 2698.*

[13] Tsai, W.T., Yu, L., Zhu, F., Paul, R. 2003. *Rapid Verification of Embedded Systems Using Patterns. COMPSAC. pp 466 – 471*

[14] Cyrille Artho et al., 2005. Combining test case generation with runtime verification. *The Journal of Theoretical Computer Science 336 (2005) pp. 209-234*

[15] Sarfraz Khurshid and Darko Marinov, 2004. TestEra: Specification-based Testing of Java programs using SAT. *The Journal of Automated Software Engineering, vol. 11, issue 4 (October 2004) pp. 403-434*

[16] Nagle, Carl. Software Automation Framework Support: http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm *Retrieved: February 13th 2009.*