Control Flow Analysis and Coverage Driven Testing for Web Services

Li Li, Wu Chou, Weiping Guo
Avaya Labs Research, Avaya Inc.
{lli5,wuchou,wguo}@avaya.com

Abstract

In this paper, we describe an approach to discover the control flow graph of web services for web services analysis, verification, and testing. For this purpose, three novel methods are proposed. First, we introduce a domain independent RDF Schemas for concise resource-oriented functional specification of web services operations. Secondly, we describe the use of RDF entailment to accurately derive the control flow from the functional specifications. We developed a transformation from RDF graph to SPARQL query to facilitate the RDF entailment which offers flexibility and extensibility over the direct graph matching approach. The third is a linkage based web services modeling and analysis framework, within which we apply an improved Google PageRank algorithm to efficiently calculate test coverage potential using the derived control flow. We justify that the proposed linkage based web services modeling and analysis framework is particularly suitable for testing web services. A prototype of the proposed methods has been implemented and tested on some standard based web services. Experimental results show that the control flow analysis is quite efficient and accurate, and the coverage based test results of the proposed approach are very promising.

1. Introduction

A control flow of a web service defines all legitimate sequences of successful web services operations. If a client follows these sequences with appropriate services messages, it should expect positive response from the service provider. If a client deviates from these sequences, it should expect negative response from the service provider and the flow usually terminates with a fault. Control flow of this kind is rather different from its conventional meaning in programming languages, where control flow means instruction execution paths specified explicitly by the program (code).

A control flow for web services can be represented by a directed graph in which vertices are operations and edges are control dependencies. Such graph answers the basic question: Should this operation be invoked successfully, what other operations can be invoked successfully next? The control flow graphs of a stateless web service, e.g. a calculator web service that performs add and subtract, are always fully connected, because a client can call any operation in any order as many times as necessary.

However, the control flow graphs for stateful and conversational web services can be very complex as constrained by the service semantics and stateful dependencies. Many web services are stateful or conversational, e.g. WS-Session [1], WS-Eventing [2], ECMA-348 [3], and WS-BaseNotification [6]. WS-Eventing is an example of stateful service, because a client must create a subscription resource using the subscribe operation before deleting it using the unsubscribe operation. Clearly, there is an edge from subscribe to unsubscribe and the other way around.

We need to make distinction between message flow (data flow) and control flow analyses for web services. Message flow analysis [16] reveals what data are transmitted between operations without any reference to internal service state. In WS-Eventing for example, the subscribe operation returns a subscription ID that must be used by subsequent unsubscribe operation.

It is possible, as in the case of WS-Eventing, that these two types of flows coincide. But in general, they do not need to overlap, and one cannot be deduced from the other, i.e. the existence of the message flow does not imply the existence of the control flow and vice versa. For example, in Parlay X Multimedia Conference Web Services [4], there is a message flow (conference ID) from operation createConference to operation disconnectParticipant. But this does not mean createConference will warrant disconnectParticipant, unless a participant actually exists. In conversational CSTA Routeing Services [5], a routeRequest message can be followed by routeReject, but there is no message flow between them at all.

These use cases illustrate an important distinction: unlike message flow analysis, which solely depends on input and output message relation between operations, control flow analysis is based on the *internal state* of the service. It captures an important aspect of the service semantics that is not covered by the message flow analysis, and this semantic aspect must be tested in order to claim the functional correctness of the service. We show, in this paper, the information from both message and control flows can be combined to enhance the test quality.

978-0-7695-3310-0/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWS.2008.104

For web services with small number of operations, the control flows could be manually determined by pairwise matching the operations. However, this process does not scale up to large or multiple web services where hundreds of operations are involved. To obtain consistent and meaningful results, even the manual process must presume the existence of some formal criteria, or semantic model, to pair and arrange these operations.

For the abovementioned reasons, we chose to adopt a model based approach to derive control flows from web services specifications. This approach is scalable as we can "tag" individual operations and let the algorithm uncover the control flows. Our model based approach uses Precondition and Effect to specify service states in terms of resources, and apply logical inference to determine control dependencies between operations. This approach is well adapted to the RDF [7], which is an attractive semantic framework designed for describing resources on the Web, upon which sound, consistent and efficient inferences can be made. In particular, we developed a RDF entailment mechanism and applied to RDF graphs to determine the control flow relation between operations.

Once a control flow is derived, we can generate the web services test plans based on various criteria. The linkage based web services testing framework described in this paper is based on a novel application of Google's PageRank algorithm to the linkage structure of the discovered control flow of web service operations. It calculates the linkage based "coverage score" for each operation. By doing so, we can maximize the test coverage with minimal test cases, and even optimize it by incorporating an appropriate service invocation usage model. As it will be shown later, the PageRank algorithm is particularly suitable for web services testing, and testing web services invocation is quite different from the conventional software testing for object-oriented programming languages, e.g. Java, C/ C++, etc.

The rest of this paper is organized as follows: Section 2 discusses some recent work related to our approach. Section 3 is dedicated to RDF based control flow analysis using RDF entailment. Section 4 describes our linkage based web services analysis and modeling framework including the linkage based operation coverage ranking using Google PageRank algorithm. Section 5 presents the implementation and experimental results, and we conclude with section 6.

2. Related Work

Web services testing is a fast moving research area as the complexity of web services increases. The prevalent approach in this area is model-based web service testing.

Sinha et al [8] proposed a model-based web service test framework where a WSDL operation is augmented with Preconditions and Effects using OWL ontology to specify its functional behavior. Extended FSM (finite-state machine) is then constructed from these specifications and used to generate test cases. The result EFSM always has a single state by definition. However, it is not clear how to determine if a precondition or effect is true from the service ontology. In our approach, we use domain independent RDF schemas. We also derive entailment relations between operations. And from that, we develop a PageRank based test coverage analysis paradigm from our linkage based web services analysis framework.

Narayanan et al [9] adopted a DAML-S based ontological representation of web services and proposed a Petri net formalism for encoding the service semantics for simulation and verification. In this approach, the "control flows" are already defined in DAML-S and are translated into Petri net. This falls short of our goal where we try to discover/derive the control flow graph with minimal human annotations.

Similarly, Wang et al [10] proposed a web services test generation approach by using Petri net translated from OWL-S process descriptions. Again, the "control flows" have been prescribed instead of being discovered as in our approach.

WSAT [11] is a formal analysis tool for web services composition developed by Fu et al. This tool takes some preexisting "control flows" in BPEL or Conversation Protocol and translated them into GFSA (Guarded Finite State Automata) which is subject to a series of formal analyses. Upon successful completion of these analyses, the GFSA is further translated to model checking language Promela to determine if the desired service can be composed. Our approach can be integrated with this tool by deriving the control flow graph, which can be transformed into different FSMs or Petri nets as needed.

Carroll [12] described a way to match RDF graphs based on graph isomorphism theory. It shows that RDF equality and equivalence can be resolved by testing graph isomorphism. Our RDF matching approach follows this line of thought. But instead of directly matching two RDF graphs through graph isomorphism, we translate one RDF graph into SPARQL [19] and transform the complicated graph matching process into RDF query with reduced complexity. This approach is more efficient, flexible and extensible over the approaches of direct graph isomorphism matching.

There are many open source and commercial tools that conduct various degrees of functional, performance, security and conformance tests on web services. A good review can be found in [13] and [14]. In terms of functional test, these tools can automatically create test cases from WSDL files but no control flow analysis is performed.

Code Coverage based software testing is an active research area. The conventional way to determine the coverage priority for program blocks is based on Dominator Analysis [15] which assigns priority score to each program blocks based on its coverage potentials. A high priority block means that in order to executing it, a program has to execute many blocks preceding it, thus more code is covered. However, testing web service interface is very different from testing a traditional program whose control flow is structurally fixed by the code itself. So far, we have not seen any service coverage driven algorithms and modeling directly aimed at web service testing.

3. RDF Based Control Flow Analysis

This Section describes the RDF based control flow analysis framework for web service operations.

3.1. RDF Schemas and Axioms

The functional specification of a web service consists of the axioms that define the precondition and effect for the successful invocation of each operation in the WSDL.

Each Precondition and Effect of an axiom contains a RDF graph that describes the service state *before* and *after* invoking the operation. In order to derive the control flow for operations within or across web services, we developed a three-layer ontology representation. The top layer is the generic RDF classes that can apply to all web services of interest. The middle layer is the service specific RDF classes, defined as subclasses of the generic top layer RDF schemas. And the bottom layer consists of the operation specific RDF instances.

The generic RDF Schema models the service states as connected resources, where each *node* class represents a resource with properties *state*, *role*, and *arc* linking to other nodes.

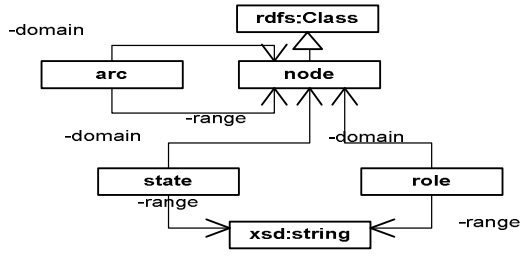

Figure 1: Generic RDF classes and properties for Axioms

A concrete example of middle layer is the RDF subclasses for ECMA-348 Call Control services, which include Call, Connection and Device, as shown below.

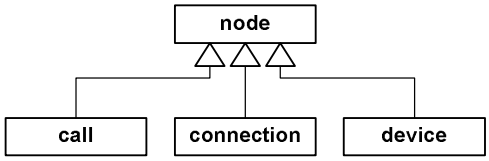

Figure 2: RDF classes (device, call, connection) for ECMA-348 Web Services

To specify the axioms for the operations, we consult the service specification. The following diagram from ECMA-269 describes the Accept Call operation, which transits an incoming call between two devices from "alerting" to "ringing" mode.

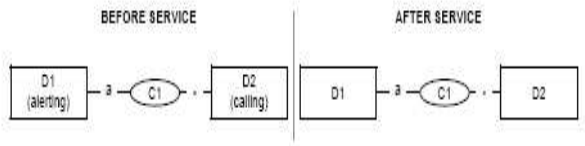

Figure 3: Resource diagrams for CSTA Accept Call service

From this information, we create the RDF graph for the precondition of this operation as follows:

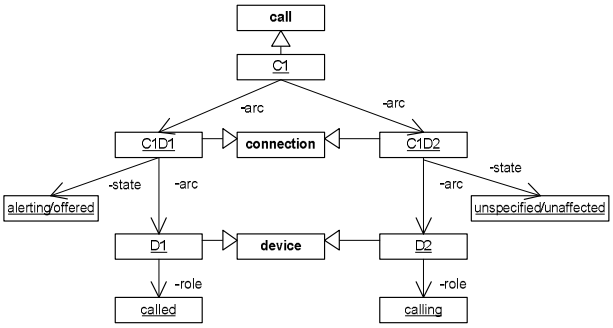

Figure 4: RDF graph representing the precondition of CSTA Accept Call service

For brevity, this paper encodes such RDF graphs by a set of paths whose nodes can have attributes in square brackets. The above RDF graph will be encoded as two paths:

C1/C1D1[alerting/offered]/D1[alerting]
C1/C1D2[unspecified/unaffected]/D2[calling]

The relation and state of the resources are necessary in discriminating operation functions. This phenomenon is prevalent in CSTA services where many call control operations depend on the subtle difference of connection states. Consider the following resource diagram also from ECMA-269 [5] for the Answer Call operation and compare it with the resource diagram for Accept Call (Figure 3).

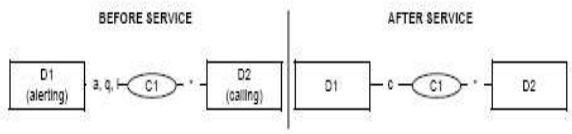

Figure 5: Resource diagram for CSTA Answer Call service

Clearly, both operations involve one call (C1), two connections (C1D1 and C1D2) and two devices (D1 and D2). Only the states of the connection C1D1 in these two operations are different.

By exploiting the resource-oriented aspect of web services [22], such RDF graphs can directly model service states for a variety of stateless, stateful and conversational web services that we investigated. Another benefit of the RDF graph representation is that the resource nodes can be linked with the XML Schemas elements. And based on that, the positive and negative XML messages can be generated by conforming to or violating the relations and states of the described resources respectively.

3.2. Control Flow Analysis

Using the short-hand RDF representation from the previous subsection, we focus on four operations from Parlay X Conference Web Services. These four operations are listed and described below:

- createConference (CC): the Precondition is always true. The Effect states that a conference "C1" is created.
- getParticipants (GP): the Precondition states that there must be a conference "C1", the Effect is the same as Precondition.
- inviteParticipant (IP): the Precondition states that there must be a conference resource "C1". The Effect states that the conference "C1" will contain a participant resource "P1" with media type "M1".
- disconnectParticipant (DP): the Precondition states that conference "C1" must contain a participant "P1" with media "M1". The Effect states that the conference "C1" remains.

The Precondition and Effect of these operations are listed below:

Operation	Precondition	Effect
CC	{}	{C1}
GP	{C1}	{C1}
IP	{C1}	{C1/P1/M1}
DP	{C1/P1/M1}	{C1}

By using RDF "Open World Assumption", these axioms make minimal assertions about the resources. For instance, the assertion {C1} does not mean conference C1 has no participant. Instead, it means other assertions, true or false, are irrelevant to this operation. The RDF existential semantics also implies that if an entailment is true for one set of resources, it is true for any set of resources with the same properties. This semantics is particularly useful in modeling web services that can create many resources of the same type. The RDF graph entailment is well-defined by RDF Semantics [18]. In our approach, graph *G1* entails *G2* if *G2* is a subgraph of *G1*. By visual inspection of the two complementary operations IP and DP, we can find the following entailment relations:

1. the Effect of IP entails the Preconditions of IP and DP;
2. the Effect of DP entails the Precondition of IP;

This leads to the following links between these two operations:

$$IP \rightarrow IP,\ IP \rightarrow DP,\ DP \rightarrow IP$$

This intuition can be described formally by the definition below:

Definition 1

Given a set of web services *W,* define the following relations:

Operation Entailment: Let *Axiom(OX)={(PX,EX)}* denote the axioms of operation *OX* in *W,* where *PX* is the precondition, and *EX* is the effects defined in Section 3.1. Similarly, let *Axiom(OY)={(PY,EY)}* denote the axioms of operation *OY*. Then *op_entail(OX,PY,OY)* denote that *OX entails OY by PY*. The predicate is true if and only if there exists *(PX,EX)* in *Axiom(OX)* and *(PY,EY)* in *Axiom(OY)* such that *rdf_entail*(*EX,PY)*, which denotes RDF entailment defined in [18].

Operation Linkage: There is a link from operation *OX* to *OY* under precondition *C*, denoted by *OX→OY{C}*, if an only if *op_entail(OX,C,OY)* is true.

The control flow analysis amounts to deciding entailment between two RDF graphs. In general, direct model-theoretic interpretation of RDF entailment is not realistic as we have to test truth condition in all possible worlds. Instead, we adopt a graph-theoretic interpretation of entailment motivated by graph isomorphism [12]. Furthermore, we take advantage of recently ratified RDF query language SPARQL [19] to facilitate the graph matching process.

It turns out that our RDF axioms can be readily translated into SPARQL so that we can reduce RDF graph matching problem into well-defined RDF query problem. A more formal description of this idea is given below.

Definition 2

RDF Entailment: Let *Q(V,PY)* denote a SPARQL query equivalent to RDF *PY* with a set of query variables *V*. *rdf_entail*(*EX,PY)* is true if and only if there exists a query result *S=query(EX, Q(V,PY))* such that $|S| = |V|$, i.e. all query variables in *V* are matched.

Translation of RDF graph into SPARQL can be carried out on the RDF graph by judiciously replacing RDF blank nodes with variables and adding proper filters to ensure distinct blank nodes do not match the same node. To facilitate axiom representation, we also support regular expressions in literal nodes, useful to expressive alternative states. In this sense, our translation based approach is more flexible and extensible than direct graph isomorphism without sacrificing the simplicity of RDF.

The following is the SPARQL query automatically translated from the Precondition of operation disconnectParticipant (formatted for readability):

```
SELECT DISTINCT ?x1 ?x3 ?x2
WHERE {
?x1 <http://research.avayalabs.com/dialog/ws-stargaze/axiom#arc> ?x2
.
?x1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.csapi.org/wsdl/parlayx/multimedia_conference/v2_0/interfa
ce#participant> .
?x2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.csapi.org/wsdl/parlayx/multimedia_conference/v2_0/interfa
ce#media> .
?x3 <http://research.avayalabs.com/dialog/ws-stargaze/axiom#arc> ?x1
.
?x3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.csapi.org/wsdl/parlayx/multimedia_conference/v2_0/interfa
ce#conference> .
FILTER(?x1 != ?x3) }
```

The solution of this query against the Effect of operation inviteParticipant is *S={x1=P1,x2=M1,x3=C1}*, which justifies the operation entailment and linkage:

$$IP \rightarrow DP\{C1/P1/M1\}.$$

For a set of *N* web services operations, the algorithm to construct the control flow graph is to make N^2 comparisons between the operations according to Definition 1 and Definition 2. The algorithm thus has time complexity $O(N^2)$.

The following diagram illustrates the control flow (left) derived from the four operations, where the preconditions for entailment are omitted for clarity. For reference, we include the derived message flow on the right.

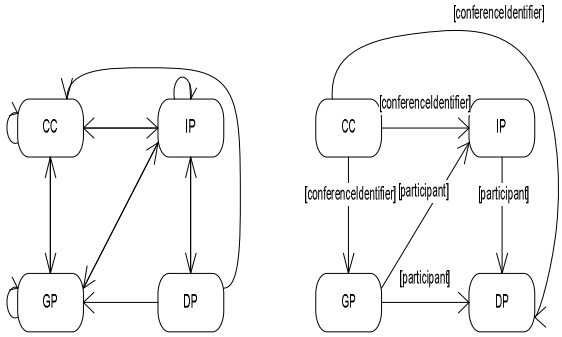

Figure 6: Control flow (left) and message flow (right)

3.3. Control Flow Applications

To construct the test flows, the control flow graph can be interpreted as Extended Finite-State Machine (EFSM) whose vertices represent operations and transitions represent entailment links. The configuration *M* of EFSM is initialized with a particular precondition. The flow generator starts with the initial *M* and iteratively updates *M* and selects operations from the control flow graph whose precondition is entailed by *M* (Definition 2).

For example in Figure 6, starting with *M={C1}*, we can generate such test flows: IP{n},DP{n} and (IP,DP){n} that will result in positive responses, and test flows: IP{n},DP{n+1} for $n \geq 1$ that will result in negative (fault) responses.

The test flow examples suggest that we could use resource counters to replace RDF graphs in functional specification. However, this simplification has few problems, because control flow depends on more than just the number of resources. For example, the seemingly correct precondition of DP: *C>0* and *P>0* (number of conferences and number of participants are greater than zero), can lead to some false positive states, such as {C1/P1/M1, C2}, in which the true precondition of DP is violated because conference C2 does not have any participant.

Control flow can be used to detect false negative message flows [16] by verifying them against control flow as follows: let *OX* and *OY* denote operations. There is a message flow *OX* to *OY* only if there is a control flow from *OX* to *OY*. It is evident that if a client can never reach operation *OY* from *OX* following any control flow, the message flow between them must be useless and spurious.

4. Test Coverage Analysis Using PageRank

Since the number of test flows from a control flow graph grows exponentially with their length, we need further guidance on generating testing plans. One software testing strategy is to maximize code coverage with

minimal set of test cases [15]. This notion can be carried over to web service testing to maximize the number of operations being tested with minimal number of test cases. In order to test an operation, a test case must have an input message, an expected output or a fault message. Constructing such test cases is a time consuming work and the number of test cases should be minimized as much as possible.

However, web services testing is very different from the traditional software testing, because web services operations, unlike fixed program blocks, can be entered from anywhere by client. Similar to web surfing, the client can choose to jump to any other operations at will. From this respect, web service operations behave more like web pages and the control flows indicate the linkages between them. A well-behaved client would most likely follow the flows to "visit" each operation in the right order, but clients at large may deviate from the prescribed flows and jump around.

For flow based coverage test, we have to rank operations such that an operation with a high rank score would imply more operations must be "visited" before reaching this operation. Those "visited" operations are thus "covered" by this operation. Consequently, this operation would lead to high test coverage.

To achieve this, we model each web service operation as a special web page with the linkage derived from the control flow analysis described in Section 3. We make a novel use of an improved Google PageRank algorithm [17] that models the linkage between web services operations by a Markov Chain and assigns coverage ranking score to each operation based on its linkage. This model is described mathematically as follows.

We represent the control flow graph of n nodes (operations) by an $n \times n$ row-normalized adjacency matrix H where $H_{ij} = |P_i|^{-1}$ if there is a link from node i to node j and $|P_i|$ is the total number of outgoing links from node i; $H_{ij}=0$ otherwise.

Define matrix $S = H + a(1/n)e^T$ where a is the dangling node indicator vector with $a_i = 1$ only if node i is a dangling node (no outgoing links) and e^T is the row identity vector. The PageRank Google matrix G is given by the following equation: $G = \alpha S + (1-\alpha)ev$, which models a random web surfer with probability α of following hyperlinks in the pages and probability $1-\alpha$ of jumping to other preferred nodes (say the favorite pages) according to v, a row probability distribution vector called the personalization vector that can be used to model client usage patterns. Given an initial probability distribution vector $\pi^{(0)}$ on n nodes, it calculates $\pi^{(k)} = \pi^{(k-1)}G = \pi^{(0)}G^k$ until some convergence criterion is met, and the rank order is given by the values of $\pi^{(k)}$ on each node.

From this linkage based Markov Chain model, the rank score of a web service operation indicates the amount of "accumulated" links into the operation and consequently the coverage potential of the operation. Once operations are ranked, the coverage driven test would select the highest ranked operation and test operations "covered" by it. If there are uncovered operations, the next highest ranked uncovered operation will be selected and the process repeats until all operations are tested.

5. Implementation and Experiments

We have implemented the RDF based control flow analyses in Java using open source RDF package Jena [20] and WSDL parser WSDL4J [21]. The framework of our web services flow analyzer (WS-StarGaze) is illustrated below. The flow analyzer takes a WSDL file, a patch file that fixes bugs in WSDL, and a RDF Axiom file as input, and produces both the control flow and message flow in adjacency matrix and RDF format. The linkage based PageRank algorithm is then applied to obtain the linkage based coverage ranking of each operation. The PageRank algorithm takes a control flow graph adjacency matrix as input and outputs the page rank scores.

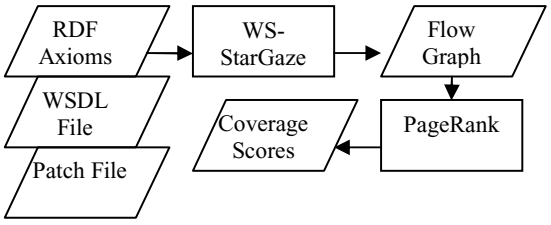

Figure 7: Architecture of WS-StarGaze

Two web services, Parlay X Conference Web Service and CSTA Routeing Service, are annotated with axioms and analyzed for test coverage. In our experiment, self-loops in the control graph were removed to be consistent with coverage based code analysis. They were modeled as links with probability $1-\alpha$ in the PageRank algorithm.

The performance of flow analyses is measured on a Windows 2003 Server with 3GHz CPU and 2GB RAM. The results are summarized below in Table 1 where the time includes:

1) loading and parsing input files;
2) parsing and translating RDF graphs;
3) computing both message and control flows; and
4) saving results to output file.

The relations column shows the number of entailment relations and number of message flows in parentheses. The computed control flow graphs have been verified for

correctness against the standards upon which these services are defined.

service	operation	relations	time (sec)
Conference	9	44 (17)	3
Routeing	10	34 (0)	10

Table 1: Performance of WS-StarGaze flow analyses

The derived CSTA Routeing Service Control Flow is illustrated below (Figure 8). The following table (Table 2) is the operation coverage scores calculated with α=0.85 based on the above control flow graph. The algorithm correctly ranks the operations by their coverage potentials. In particular, the PageRank ranks operation routeEnd as the highest and the rest according to their coverage potentials.

Figure 9 is the control flow for Parlay X Conference Web Services. The PageRank scores of the Conference Service are displayed in Table 3. The PageRank algorithm in this case correctly assigns the highest score to operation endConference which happens to be "last" operation of the service with the highest coverage potential. Ranking for other operations also indicates the similar feature.

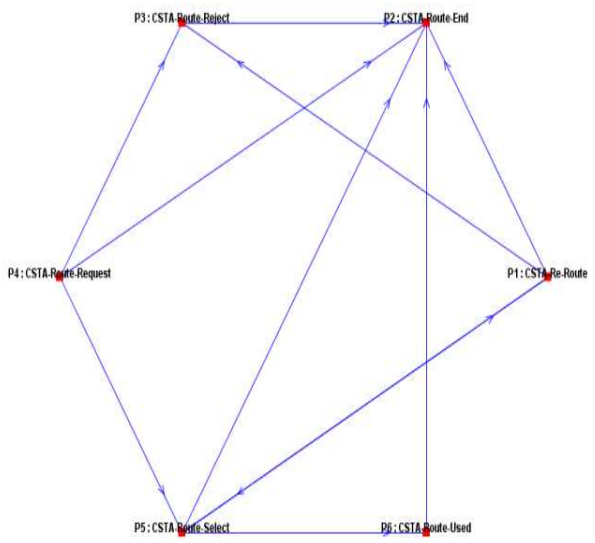

Figure 8: Control flow for CSTA Routeing Service

operation	Rank score
CSTA-Route-End	0.4091
CSTA-Route-Reject	0.1372
CSTA-Route-Select	0.1372
CSTA-Route-Used	0.1192
CSTA-Re-Route	0.1192
CSTA-Route-Request	0.0780

Table 2: PageRank output of CSTA Routeing Service

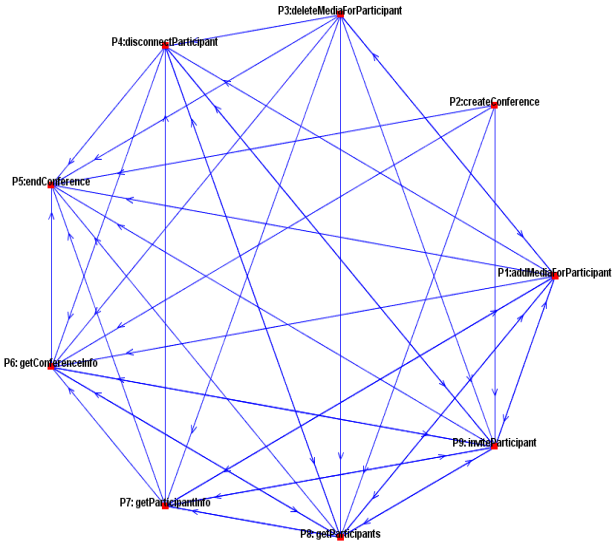

Figure 9: Control flow for Parlay X Conference Service

operation	Rank score
endConference	0.1715
createConference	0.1655
inviteParticipants	0.1491
getParticipants	0.1491
getConferenceInfo	0.1319
disconnectParticipant	0.0773
addMediaforParticipant	0.0685
getParticipantInfo	0.0672
deleteMediaforParticipant	0.0199

Table 3: PageRank output of Parlay X Conference Service

6. Summary

This paper presented a RDF based method to derive control flow graphs for web services operations and a linkage based web services modeling and analysis framework. We proposed generic RDF Schemas designed for concise functional specification of web service operations. We developed the entailment algorithm for accurately deriving control flow graphs of web services from their functional specification. The entailment is based on an efficient, flexible and extensible SPARQL query based RDF graph matching. We described a novel application of Google PageRank algorithm on the derived control flow graph to calculate coverage potentials of operations. The performance of control flow analyses is provided and a prototype system was developed. It was applied to stateful web services scenarios of ECMA-348 and Parlay. Experimental results of the proposed approach and the linkage based coverage web services testing framework are very promising.

For future research, we plan to speed up the annotation process by developing some graphical interface for axiom composition so we can annotate large web services. We also plan to investigate the test flow and test case generations by combining the derived control flow and message flow graphs.

7. References

[1] Standard ECMA-366, WS-Session, http://www.ecma-international.org/publications/standards/Ecma-366.htm

[2] Web Services Eventing (WS-Eventing), W3C Member Submission 15 March 2006, http://www.w3.org/Submission/WS-Eventing/

[3] Standard ECMA-348, Web Services Description Language (WSDL) for CSTA Phase III, 3rd edition (December 2006), http://www.ecma-international.org/publications/standards/Ecma-348.htm

[4] Parlay X Web Services Specification, Version 2.1, part 12, Multimedia Conference, http://www.parlay.org/en/specifications/pxws.asp

[5] Standard ECMA-269, Services for Computer Supported Telecommunications Applications (CSTA) Phase III, 7th edition (December 2006), http://www.ecma-international.org/publications/standards/Ecma-269.htm

[6] Web Services Base Notification 1.3 (WS-BaseNotification), OASIS Standard, 1 October 2006, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

[7] Dave Beckett (ed), "RDF/XML Syntax Specification (Revised)", W3C Recommendation, 10 February 2004.

[8] Sinha, A. and Paradkar A., "Model Based Functional Conformance Testing of Web Services Operating on Persistent Data", *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications TAV-WEB '06*, Portland, Maine, July 17 2006, pp. 17-22.

[9] Srini Narayanan, Sheila A. McIlraith, "Simulation, Verification and Automated Composition of Web Services", *Proceedings of WWW 2002*, May 7-11, 2002, Honolulu, Hawaii, USA, pp 77-88.

[10] Yongbo Wang, Xiaoying Bai, Juanzi Li, Rubo Huang, "Ontology-Based Test Case Generation for Testing Web Services", Proceedings of 8th IEEE International Symposium on Autonomous Decentralized Systems (ISADS'07), 2007.

[11] Xiang Fu, Tevfik Bultan, Jianwen Su, "WSAT: A Tool for Formal Analysis of Web Services", Proc. CAV, LNCS 3114, pages 510--514. Springer, 2004.

[12] Jeremy J. Carroll, "Matching RDF Graphs", Proceedings of the First International {Semantic Web} Conference, pages 5-15, 2002.

[13] Rick Grehan, "Three open source Web service testing tools get high marks", http://www.infoworld.com/article/07/05/11/19TCwebservicetest_2.html, May 11 2007.

[14] Rick Grehan, "Clean up your SOAP-based Web services", http://www.infoworld.com/article/07/11/26/48TC-web-services-test-tools_1.html, November 26, 2007.

[15] H. Agrawal, "Dominators, super blocks and program coverage", Proceedings of the 21st Symposium on Principles of Programming Languages, pages 25-34, January 1994.

[16] Li Li, Wu Chou, "Automatic Message Flow Analyses for Web Services Based on WSDL", Proceedings of 2007 IEEE International Conference on Web Services (ICWS 2007), page 880-887, Salt Lake City, July 2007.

[17] Amy N. Langville and Carl D. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, 2006.

[18] Patrick Hayes (ed), "RDF Semantics, W3C Recommendation", 10 February 2004.

[19] Eirc Prud'hommeaux, "SPARQL Query Language for RDF", W3C Recommendation, 15 January 2008.

[20] Jena – A Semantic Web Framework for Java, http://jena.sourceforge.net/

[21] WSDL4J, http://sourceforge.net/projects/wsdl4j

[22] David Booth, et al (eds), "Web Services Architecture", W3C Working Group Note, 11 February 2004