VeriWS: A Tool for Verification of Combined Functional and Non-functional Requirements of Web Service Composition

Manman Chen* Tian Huat Tan† Jun Sun† Yang Liu‡
Jin Song Dong*
*National University of Singapore, Singapore
†Singapore University of Technology and Design, Singapore
‡Nanyang Technological University, Singapore, Singapore
{chenman, dongjs}@comp.nus.edu.sg, {tianhuat_tan, sunjun}@sutd.edu.sg
yangliu@ntu.edu.sg

ABSTRACT

Web service composition is an emerging technique to develop Web applications by composing existing Web services. Web service composition is subject to two important classes of requirements, i.e., functional and non-functional requirements. Both are crucial to Web service composition. Therefore, it is desirable to verify combined functional and non-functional requirements for Web service composition.

We present VeriWS, a tool to verify combined functional and non-functional requirements of Web service composition. VeriWS captures the semantics of Web service composition and verifies it directly based on the semantics. We also show how to describe Web service composition and properties using VeriWS. The YouTube video for demonstration of VeriWS is available at `https://sites.google.com/site/veriwstool/`.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services

General Terms

Verification, Reliability, Performance

Keywords

Web service composition, verification, functional requirements, non-functional requirements

1. INTRODUCTION

Web service technologies enable dynamic inter-operability of heterogeneous and distributed Web-based platforms. Web service composition makes use of existing Web services to build new Web applications based on Service Oriented Architecture (SOA). The result of Web service composition is called a *composite service* and the Web services that constitute the composite service are called *component services*. The *de facto* standard for Web service composition is Web Services Business Process Execution Language (WS-BPEL) [5], an XML-based orchestration business process language for describing the behavior of a business process based on its interactions with its component services. It supports various compositional structures such as sequence, parallel composition, conditional choice, etc., to facilitate the composition of Web services.

There are two crucial classes of requirements for Web service composition, i.e., functional and non-functional requirements. Functional requirements are related to the conformance of Web service composition to the requirements on its functionality, whereas non-functional requirements are related to the quality of service (QoS), e.g., response time, availability, and cost. Non-functional requirements can determine the success or failure on Web service composition, as the Web service composition that is functionality correct but with poor performance is not likely to be adopted by the users. To guarantee the performance of Web service composition, the non-functional requirements are often noted down in service-level agreements (SLAs), which are a contractual basis between service consumers and service providers on the expected QoS level. Given a computer purchasing service (CPS), e.g., Dell.com, an example of functional requirements is that "the CPS always replies to users with the purchasing status", whereas an example of non-functional requirements is that "the CPS always responds within 3 seconds".

Concurrency has been frequently used in Web service composition. Nevertheless, concurrency often leads to subtle bugs as programmers have to deal with issues like multi-threads and critical regions. Yin *et al.* [19] reports 39% of concurrency bugs are not fixed correctly and concurrency bugs are the most difficult to fix among common bug types. Thus, it is desirable to apply automatic verification techniques on WS-BPEL, e.g., model checking [4]. Existing tools have provided verification for either functional requirements [7, 9, 2, 13, 16] or non-functional requirements [20], however, they could not support for combined functional and non-functional requirements. An example of combined functional and non-functional requirements is "CPS will always reply to the user, and when CPS replies to the user, the delay of CPS will not be larger than 3 seconds (from the points where it receives the request)".

Although combined functional and non-functional requirements are important for Web service composition, currently there is no integrated tool support for these two classes of requirements. To facilitate the checking and improvement of functional and non-functional aspects of Web service composition, we have developed a toolkit called VeriWS. VeriWS is a tool designed to verify Web service composition for combined functional and non-functional

ICSE Companion'14, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2768-8/14/05...$15.00
http://dx.doi.org/10.1145/2591062.2591070

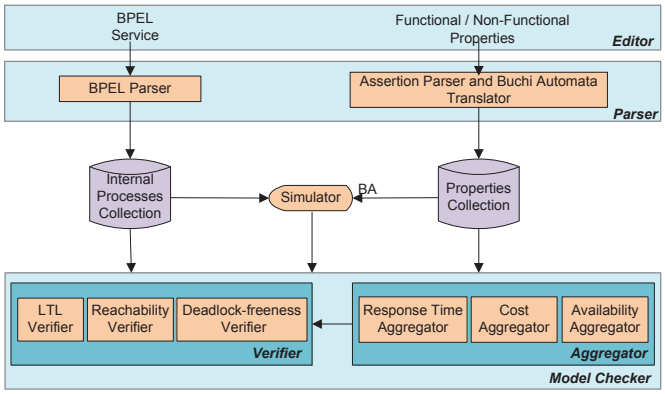

Figure 1: VeriWS Architecture

requirements, based on the QoS of participated component services. A counterexample will be provided when the violation of a requirement is detected. It also integrates with a simulator component to provide the simulation on behaviors of the composite service, as well as, to replay the counterexample that is reported.

In the following, we present the main features of VeriWS.

- It supports verification on different kinds of combined functional and non-functional properties of Web service composition, i.e., linear temporal logic (LTL) properties, reachability properties, and deadlock-freeness properties.
- It supports the simulation of Web service composition models and provides the counterexample in WS-BPEL, so that developers can easily locate the origin of the bug and subsequently fix it.
- It is easy to use for Web service modeling, testing and verification.

Our initial experiment was illustrated in [3], it has demonstrated the effectiveness of our method. To our knowledge, VeriWS is the first tool to provide verification on combined functional and non-functional requirements of Web service composition.

2. VERIWS

2.1 Architecture and Implementation

VeriWS is a self-contained toolkit that provides the state-of-the-art verifier for combined functional and non-functional requirements for Web service composition specific to WS-BPEL. Given WS-BPEL programs with QoS values for each component service, VeriWS enables integrated verification of both functional and non-functional requirements. Web service composition is verified directly based on its operational semantics. We adopt the formal operational semantics of WS-BPEL described in [6]. With the operational semantics, a WS-BPEL program can be treated as a *transition system*, which is subject to model checking. When the verification is violated, a counterexample in WS-BPEL will be provided to make developers easier to find and correct problems.

VeriWS is implemented in C# and uses a modular architecture. It provides powerful editor, simulator, and verifier. Figure 1 shows the architecture of VeriWS. To verify the composite service using WS-BPEL, a user first inputs the WS-BPEL service description and combined requirements to be verified using the editor. The editor is implemented using the text editor component of Sharp Develop framework[1], which supports multi-documents environment and is customizable in terms of syntax highlighting, code folding, etc.

Table 1: Aggregation Function

QoS Attribute	Sequential	Parallel	Loop	Conditional
Response Time	$\sum_{i=1}^{n} q(s_i)$	$\max_{i=1}^{n} q(s_i)$	$k * (q(s_1))$	$\max_{i=1}^{n} q(s_i)$

Subsequently, the WS-BPEL service description and the combined requirements are parsed into processes collection and properties collection respectively. The verifier is then used to check the WS-BPEL service against the combined requirements. The verifier checks the combined requirements based on the aggregated QoS values obtained from the aggregators. There are three kinds of verifiers, i.e., LTL verifier, reachability verifier and deadlock-freeness verifier, which are designed to check LTL properties, reachability and deadlock-freeness properties respectively. VeriWS offers extensible software architecture, such that new verifiers and aggregators could be plugged in easily.

The simulator can be used to visualize the behavior of a WS-BPEL service and the combined requirements. The simulator can also be used to replay the counterexample returned by the verifier. We illustrate the details of verifier, aggregator and simulator in the following.

2.1.1 Aggregator

Different aggregators are used to aggregate different QoS based on the their aggregation functions. Table 1 shows the aggregation functions that *response time aggregator* used for different compositional structures. *Response time aggregator* sums up the response time of component services in sequential structure, i.e., the response time of the composite service is calculated by summing up the response time of the n participated component services. The response time of the composite service composed by parallel structure is decided by the maximum response time among the n participated component services. As for loop structure, the response time of the composite service is obtained by summing up the response time of the participating component service for k times, where k is the number of maximum iteration of the loop. While for the conditional composition, the response time of the composite service is the maximum response time of n participating component services since at the design phase it is unknown which guard will be satisfied. Cost and availability aggregators work similarly, where their aggregation functions could be found in [3].

2.1.2 Verifier

The verifier model checks combined functional and non-functional requirements. If a counterexample is found, it can be replayed using the simulator. Several verifiers are implemented to cater for different kinds of combined requirements. There are three categories of properties that are currently supported by the verifier: deadlock-freeness, reachability, and LTL. The verifier integrates the aggregated QoS values from the aggregators, into the transition system that represents the WS-BPEL process using approaches in [3], and offers verification for the following properties:

LTL Property. An LTL property checks whether the property specified in LTL holds. To verify the LTL formulae, we adopt the automata-based on-the-fly verification algorithm [15], i.e., by firstly translating a formula to a Büchi automaton (BA) and then checking emptiness of the product of the system and the automaton.

[1] `http://www.icsharpcode.net/OpenSource/SD/Default.aspx`

Table 2: Web Service Verification Tools

Tool	Requirement	Input	Intermediate
WSEngineer [7]	Functional	BPEL	FSP
WSAT [9]	Functional	BPEL	GFSA
VERBUS [2]	Functional	BPEL	Promela
WOMBAT [13]	Functional	BPEL	Petri nets
AgFlow [20]	Non-functional	Statecharts	–
VeriWS	Combined	BPEL	–

Reachability Property. A reachability property asks whether there exists a state that fulfills a given property. The properties are specified using the constraint on a set of verification variables. The verification variables are manipulated by the WS-BPEL service using the extended WS-BPEL attributes [14]. To verify reachability, Depth First Search (DFS) or Breath First Search (BFS) is applied on the transition system of the WS-BPEL process to search for a state that fulfilled the given property.

Deadlock-freeness. Checking deadlock-freeness is to check whether a WS-BPEL service contains a deadlock. The WS-BPEL service starts with receiving the message from the user, and ends with reply the user with the desired result. A state is deadlocked if it does not have any outgoing transitions, and the user has not yet been replied. To verify deadlock-freeness, standard graph traversal algorithm (e.g., DFS or BFS) is applied on the transition system of the WS-BPEL process to search for a deadlock state.

2.1.3 Simulator

The simulator could be used to visualize the behaviors of a WS-BPEL service in the form of a transition system. The simulator provides various simulation functions for users, e.g., complete generation of the transition system – where the user could generate entire state space of the WS-BPEL program; interactive exploration of the transition system – where the user could view the subset of the transition system by exploring on the actions of their interest; random simulation - where an example trace is automatically generated for the user. This allows users to have an in-depth understanding on the behavior of the WS-BPEL service through the simulation interface. The simulator is also used to visualize BA generated from the negation of a LTL property. In addition, the simulator could also allow the user to replay the counterexample returned by the verifier, when a property is violated, in order to aid the user on finding out the origin of the problem.

2.2 Comparison with Existing Tools

Table 2 shows the comparison of VeriWS with existing tools. Existing tools can either verify only functional requirements or non-functional requirements as shown in the *Requirement* column. Existing functional verification tools (WSEngineer, WSAT, VERBUS, WOMBAT) takes WS-BPEL as input, and translate WS-BPEL into an intermediate formal language (e.g., FSP, Petri nets) and use verification techniques and tools for the intermediate formal language (e.g., LTSA[2] tool is used for FSP) for verification. Their counterexamples are in their respective intermediate formal language (e.g., counterexample of WSEngineer is in FSP). Existing non-functional verification tool (AgFlow) requires the user to provide corresponding statecharts [10] as input to provide the non-functional analysis for the composite service.

In [18], we have developed the tool on verification of computation orchestration language, nevertheless the tool is only focused on functional requirement of Orc language [11]. In [17, 12], we have developed tools in analyzing the time requirement, which are only focused on the non-functional requirement.

Compared to existing tools, VeriWS is distinguished by several features. First, VeriWS supports efficient combined functional and non-functional verification which could not be achieved by any existing tools. In addition, for non-functional verification, AgFlow only provides for the non-functional analysis for the composite service as a whole, e.g., “the CPS will always be response within 3 seconds". In contrast, VeriWS could support more “fine-grained" non-functional requirement such as “when invoking the shipping service, CPS will not be delayed for more than 3 seconds". Second, VeriWS does not translate WS-BPEL to an intermediate formal language; therefore it could provide the counterexample in WS-BPEL language. Another advantage is that, this also provides a more natural handling of data semantics in XML, where formalism like XPath[3] is normally used to retrieve particular data elements in an XML document. WSAT is the only existing tool that supports on the XML data manipulation, and to support a single line of XPath operation, it requires to translate to 56 lines of Promela codes (excluding the comments) [8]. The translated code is hardly comprehensible. While in our approach, we could directly manipulate the XML data based on the semantics of XPath operation. This will in turn provide the user with a more pleasant experience to understand the behaviors of WS-BPEL services using the simulation tool.

```
<process xmlns:bpel="http://VeriWS/" ... >
  ...
  <sequence>
    <receive ... />
    <if>
      <condition>$CustomerType = 'Corporate'</condition>
      <invoke partnerLink="CBS" ... />
      <else>
        <invoke partnerLink="PBS" ... />
      </else>
    </if>
    <flow>
      <invoke partnerLink="MS" ... />
      <invoke partnerLink="SS" ... />
    </flow>
    <reply ... />
  </sequence>
</process>
```

Figure 2: WS-BPEL Description for CPS

3. DEMONSTRATION

The section is to complement with the video demonstration to illustrate the models and requirements to be verified. We use the Computer Purchasing Service (CPS), a service that allows users to purchase a computer online using credit cards.

3.1 Computer Purchasing Service (CPS)

The WS-BPEL program of CPS is described in Figure 2. In the following we illustrate the workflow of CPS. Upon receiving the request from the customer with his personal information and the computer he wishes to buy, if the type of the customer is corporate, Corporate Billing Service (CBS) is invoked synchronously (i.e., waiting for the reply from CBS before moving on) to bill the customer, otherwise, Personal Billing Service (PBS) is invoked synchronously to bill the customer. Manufacture Service (MS) and Shipping Service (SS) are triggered concurrently once receiving the billing confirmation message. MS is invoked synchronously to in-

[2] www.doc.ic.ac.uk/ltsa/

[3] www.w3.org/TR/xpath/

form the manufacture department with the purchased computer. SS is invoked synchronously to arrange the shipment for the customer. Finally, the purchasing result will be replied to the customer.

3.2 Requirements for Verification

We provide verification on five requirements, which are listed as follows:

1. CPS $\models$ deadlockfree
2. CPS $\models \Box$ (ReplyUser $\Rightarrow$ ResponseTime ≤ 6)
3. CPS $\models \Box$ Availability ≥ 0.95
4. CPS $\models \Box$ Cost$\leq$5
5. CPS $\models$ reach (invokeCBS $\wedge$ Cost<1)

The first property is the deadlock-freeness property which is used to check whether CPS is deadlock-free. The second property (i.e., $\Box$ (ReplyUser $\Rightarrow$ ResponseTime ≤ 6)) is an LTL property, which is to check whether the CPS always replies to users within 6 seconds. The third property (i.e., $\Box$ Availability ≥ 0.95) is an LTL property which is used to check whether the availability of CPS is always greater or equals to 0.95. The fourth property (i.e., $\Box$ Cost$\leq$5) is an LTL property which is used to check whether the cost incurred by CPS is always less than or equal to 5 dollars. The fifth property is a reachability property (reach invokeCBS $\wedge$ Cost<1) is a reachability property, which is used to find out whether there is a possibility that the accumulated cost is less than 1 dollar at the time when CBS is invoked. Both the second and fifth properties check the combined functional and non-functional requirements.

4. CONCLUSION AND FUTURE WORK

For Web service composition, both functional and non-functional requirements are important. Therefore, it is crucial to verify functional and non-functional requirements of composite services at design time so that it could detect the problem before deployment. With VeriWS, we provide a tool to check the satisfiability of combined functional and non-functional requirements of composite services directly based on their semantics.

As future work, we are going to enhance VeriWS by applying state reduction techniques, and provide visualization for the workflow structure of the WS-BPEL model. In addition, we also plan to integrate VeriWS toolkit with other WS-BPEL IDEs, such as Eclipse BPEL Designer [1], to provide the integrated verification for these IDEs.

5. ACKNOWLEDGMENTS

This work is supported by “Formal Verification on Cloud" project under Grant No: M4081155.020 and “Verification of Security Protocol Implementations" project under Grant No: M4080996.020.

6. REFERENCES

[1] Eclipse bpel designer project. `http://www.eclipse.org/bpel/`.

[2] J. Arias-Fisteus, L. S. Fernández, and C. D. Kloos. Formal verification of BPEL4WS business collaborations. In *EC-Web*, pages 76–85, 2004.

[3] M. Chen, T. H. Tan, J. Sun, Y. Liu, J. Pang, and X. Li. Verification of functional and non-functional requirements of web service composition. In *ICFEM*, pages 313–328, 2013.

[4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[5] O. W. S. B. P. E. L. W. T. Committee. Web Services Business Process Execution Language Version 2.0. http://www.oasis-open.org/specs/#wsbpelv2.0, Apr 2007.

[6] H. Foster. *A rigorous approach to engineering web service compositions*. PhD thesis, Citeseer, 2006.

[7] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Ws-engineer: A model-based approach to engineering web service compositions and choreography. In *Test and Analysis of Web Services*, pages 87–119. 2007.

[8] X. Fu, T. Bultan, and J. Su. Model checking xml manipulating software. In *ISSTA*, pages 252–262, 2004.

[9] X. Fu, T. Bultan, and J. Su. Wsat: A tool for formal analysis of web services. In *CAV*, pages 510–514, 2004.

[10] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[11] D. Kitchin, A. Quark, W. R. Cook, and J. Misra. The orc programming language. In *FMOODS/FORTE*, pages 1–25, 2009.

[12] Y. Li, T. H. Tan, and M. Chechik. Management of time requirements in component-based systems. In *FM*, 2014. to appear.

[13] A. Martens and S. Moser. Diagnosing SCA components using wombat. In *Business Process Management*, pages 378–388, 2006.

[14] OASIS Web Services Business Process Execution Language (WSBPEL) Technical Committee. Web services business process execution language version 2.0, Apr. 2007.

[15] J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *CAV*, pages 709–714, 2009.

[16] J. Sun, Y. Liu, J. S. Dong, G. Pu, and T. H. Tan. Model-based methods for linking web service choreography and orchestration. In *APSEC*, pages 166–175, 2010.

[17] T. H. Tan, É. André, J. Sun, Y. Liu, J. S. Dong, and M. Chen. Dynamic synthesis of local time requirement for service composition. In *ICSE*, pages 542–551, 2013.

[18] T. H. Tan, Y. Liu, J. Sun, and J. S. Dong. Verification of orchestration systems using compositional partial order reduction. In *Formal Methods and Software Engineering*, volume 6991 of *Lecture Notes in Computer Science*, pages 98–114, 2011.

[19] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. N. Bairavasundaram. How do fixes become bugs? In *SIGSOFT FSE*, pages 26–36, 2011.

[20] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004.