An Empirical Study of the Use of Frankl-Weyuker Data Flow Testing Criteria to Test BPEL Web Services[*]

Lijun Mei

The University of
Hong Kong
Pokfulam, Hong Kong
ljmei@cs.hku.hk

W. K. Chan[†]

City University of
Hong Kong
Tat Chee Avenue, Hong Kong
wkchan@cs.cityu.edu.hk

T. H. Tse

The University of
Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

Fei-Ching Kuo

Swinburne University of
Technology
Melbourne, Australia
dkuo@ict.swin.edu.au

Abstract—Programs using service-oriented architecture (SOA) often feature ultra-late binding among components. These components have well-defined interfaces and are known as web services. Messages between every pair of web services dually conform to the output interface of a sender and the input interface of a receiver. Unit testing of web services should not only test the logic of web services, but also assure the correctness of the web services during input, manipulation, and output of messages. There is, however, little software testing research in this area. In this paper, we study the unit testing problem to assure components written in orchestration languages, WS-BPEL in particular. We report an empirical study of the effectiveness of the Frankl-Weyuker data flow testing criteria (particularly the all-uses criterion) on WS-BPEL subject programs. Our study shows that conventional data flow testing criteria can be much less effective in revealing faults in interface artifacts (WSDL documents) and message manipulations (XPath queries) than revealing faults in BPEL artifacts.

Keywords—*WS-BPEL; XPath; data flow testing*

I. INTRODUCTION

Programs using service-orientation as the architectural style often feature ultra-late binding among components. These components, known as web services, have well-defined *interfaces* to specify their input and output ports and their semi-structured messages to be passed. For instance, messages between every pair of web services should dually conform to the output interface of one web service and the input interface of the other. Thus, *unit testing of web services should not only test the logic of web services, but also assure the correctness of the web services with respect to the input, manipulation, and output of messages.*

Web Services Business Process Execution Language (WS-BPEL) [26] is an orchestration language that coordi-

nates web services from the perspective of individual web services. A typical application written in WS-BPEL usually comprises BPEL code, WSDL documents, and Web Services. Testing the correctness of WS-BPEL applications should, therefore, address the integration complexity arising from the presence of heterogeneous kinds of artifacts.

We propose to consider the functional testing of WS-BPEL applications at three levels, namely the BPEL level, the WSDL level, and the Web Service level. The BPEL level considers BPEL code only. This level is concerned with the correctness of the business logic specified in a process (written in BPEL). Many existing techniques, including process modeling and verification [21], test case generation [9][29], and exception handling [4], indeed focus on this level. The second level is the WSDL level. A WSDL document specifies the interface of a service. This interface specification supplies the syntactic quantifications of the semi-structured messages (such as in XML). An incorrect WSDL specification may result in integration failures. At this level, testing should consider both BPEL code and WSDL documents (as well as other semi-structured design-time artifacts). A few techniques such as unit testing criteria [19] have been proposed. At the Web Service level [23], the integration between BPEL code and web services should be taken into account. Techniques such as analysis [7][8] and testing of service composition [1] can be considered at this level.

What kinds of artifacts can traditional testing techniques be applied effectively to reveal failures in them? Knowing the answer to this question helps researchers propose new testing techniques. It also helps practitioners understand how well the conventional testing techniques that they may have mastered can be applied effectively to testing WS-BPEL applications at different levels.

Mei et al. [19] observe that XPath is important in testing the integration of BPEL code, WSDL documents, and Web Services. XPath Query (or XPath for short) [28] is a query language designed to express or retrieve data in XML format (that is, data stored in XML documents). In a WS-BPEL application, XPath is chiefly responsible for retrieving data from XML messages or BPEL variables [14]. Each XML message has its document model, such as a Document Type Definition (DTD), to govern its structure.

Depending on the structures of different XML messages, an XPath may retrieve different data sets and then update BPEL variables or enable an activity transition, thus modifying the dynamic state of the program execution. Many existing techniques, such as [7][8], simplify an XPath as an atomic functional operation. Almost all existing techniques [9][15] have not addressed the impact of XPath on the testing of service-oriented programs.

In this paper, we report an empirical study of the effectiveness of applying the Frankl-Weyuker data flow testing criteria [6] to service-oriented programs. In particular, we study the *all-uses* criterion in detail because, on average, it tends to be more effective than other coverage criteria and testing techniques. For instance, many previous studies such as [6][11] have compared the *all-uses* criterion more favorably than other structural testing criteria and random testing. Furthermore, mutation testing can be more comprehensive than data flow testing but incurs much more cost. Taking these into account, therefore, if the *all-uses* criterion turns out to be ineffective in revealing faults in a particular kind of artifact, our findings will also cast doubt on whether other rivaling techniques can be effective.

The main contribution of the paper is twofold: (i) We report the first empirical study of the effectiveness of the all-uses criterion at the WSDL level. (ii) The experimental results show that, on average, the *all-uses* criterion is more effective in revealing faults on BPEL code than revealing faults in WSDL or XPath artifacts. This difference in effectiveness is noticeable and may indicate that faults in non-executable artifacts are harder to be detected than executable counterparts if we apply conventional structural testing techniques. It provides empirical evidence on the need to develop new techniques that should also consider non-executable artifacts.

The rest of the paper is organized as follows: Section II outlines the technical preliminaries in testing WS-BPEL programs, and revisits the *all-uses* criterion. Section III presents the experiment and analyzes the experiment data. Section IV reviews related work. We conclude the paper in Section V.

II. PRELIMINARIES

This section introduces the key constructs of WS-BPEL, shows a motivating example, and revisits the Frankl-Weyuker data flow testing concepts.

A. WS-BPEL

A WS-BPEL program consists of three components, namely BPEL, XPath, and Web Services (defined by WSDL). We review each of them in this section.
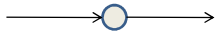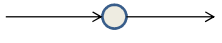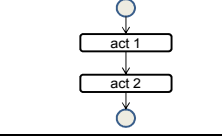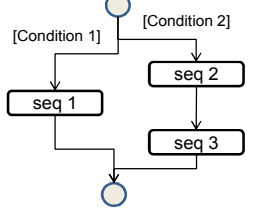
1) BPEL.

BPEL is a combination of two languages: the Web Services Flow Language (WSFL) of IBM and XLANG of Microsoft. In general, BPEL is a language to compose web services.

In conventional testing techniques (such as [7]), BPEL can be represented as a control flow graph (CFG), and we adopt their approaches to construct the CFG nodes and edges in our model. For illustration purpose, Table 1 shows sample BPEL code and the corresponding translation into CFG notation. Based on the CFG, one can then apply conventional CFG-based testing approaches (such as [6]) to test WS-BPEL programs.

A brief description according to the BPEL specification [26] is as follows: The <flow> construct specifies one or more activities to be performed concurrently. The <sequence> construct defines a collection of activities to be performed sequentially. The <invoke> construct allows the business process to invoke a service in a one-way or in a request-and-response operation using a "portType" offered by a partner. The <assign> construct updates the values of variables with new data. The <receive> construct allows the business process to wait for a matching message to arrive. Finally, the <variable> construct declares a variable. The full description can be found in [26].

TABLE 1. TRANSLATION FROM BPEL TO CFG

BPEL	Sample Code	Example in CFG Notation

assign	`<assign>`
	  `<copy>`
	  `<from variable="…"… />`
	  `<to variable="…"/>`
	  `</copy>`
	`</assign>`
		zipOnly.zipCode = XQ(Exp, I)

receive	`<receive …`
	  `operation="…"`
	  `variable="…" />`
		userName = Input()

invoke	`<invoke name="…"`
	  `partnerLink="…"`
	  `portType="…"`
	  `operation="…"`
	  `inputVariable="…"`
	  `outputVariable="…">`
	`</invoke>`
		zipInformation = City2geo(city)

sequence	`<sequence>`
	  `<…act1…>`
	  `<…act2…>`
	`</sequence>`
		act 1 → act 2

flow	`<flow>`
	  `< [condition1] seq1`
	  `< [condition2] seq2 seq3`
	`</flow>`
		[Condition 1] [Condition 2] seq 1, seq 2, seq 3

variable	`<variables>`
	  `<variable name="…"`
	  `messageType="…"/>`
	`</variables>`
		–

2) XPath.

In WS-BPEL programs, an XPath [28] specification is used as both the query language and expression language. In this paper, we focus on the use of XPath by BPEL to retrieve data from XML messages when invoking web services and receiving their responses. The structure of an XML message is defined by a WSDL specification.

Referring to [10][20], we present a fragment of XPath syntax to show how to manipulate XML messages via XPath. In the fragment, '*' denotes a label wildcard, and '.' denotes the current node. The constructs '/' and '//' mean child and descendant navigations, respectively, while '[]' denotes a predicate. An *XPath expression* is defined with the following grammar [20]:

$$q \rightarrow n \mid * \mid . \mid q / q \mid q // q \mid q[q]$$

As explained in [20], the following fragment specifies a representative part of XPath syntax and is sufficient to be used for research on XPath:

$$n(x) = \{y \mid (x, y) \in EDGES(t), LABEL(y) = n\}$$
$$*(x) = \{y \mid (x, y) \in EDGES(t)\}$$
$$.(x) = \{x\}$$
$$(q_1 / q_2)(x) = \{z \mid y \in q_1(x), (y, u) \in EDGES^*(t), z \in q_2(u)\}$$
$$q_1[q_2](x) = \{y \mid y \in q_1(x), q_2(y) \neq \varnothing\}$$

We list the fragment to help readers understand the motivating example in Section II-B. Owing to space limit, we will not explain the details.

3) Web Services.

Web Services is a kind of service-oriented architecture implementation that is both platform- and language-independent, and is accessible via standardized protocols. In WS-BPEL applications, they are invoked by BPEL similarly to external subroutine calls by traditional programs. In this paper, we are concerned about testing of messages returned by web services. We treat web services simply as invocations of external functions.

B. Motivating Example

We use an example adapted from the Apache WSIF project [27] to illustrate the testing challenges on a WS-BPEL application. The complete source code can be found in [3].

The motivating example is a Digital Subscriber Line service application that offers DSL query services. Let us discuss the DSL availability check service *isServiceAvailable*. This web service takes a user name as input, looks up the user address from the address book, then retrieves the corresponding zip information based on the given city name. The web service further verifies the given city name with the city name in the zip information, and finally determines whether the DSL service of the city is available.

The code excerpt consists of three parts: the BPEL process *isServiceAvailable* defined in `dslservice.bpel`, the web service *City2Geo* defined in `city2geo.wsdl`, and the schema of user address defined in `dslservice.wsdl`. *City2Geo* takes a city name as an input and returns the corresponding geographical information (including the zip). Figures 1, 2 and 3 show the annotated sample code extracted from these files to facilitate discussions.

```
     [Part 1]
1  <variables>
2  <variable name="zipInformation"
3      messageType="City2geo:GetLatLongSoapOut"/>
4  </variables>
     [Part 2]
5  <partnerLinks>
6  <partnerLink name="City2Geo"
7      partnerLinkType="tns:City2GeoPLT"/>
8  </partnerLinks>
     [Part 3]
9  <process name="dslservice" suppressJoinFailure="yes"…>
     [Part 3A]
10 <variables>
11 <variable name="userName" messageType=
12     "addressbook:GetAddressFromNameRequestMessage"/>
13 <variable name="userAddress" messageType=
14     "addressbook:GetAddressFromNameResponseMessage"/>
15 </variables>
     [Part 3B]
16 <invoke name="invokeAddressBookLookup"......
17     inputVariable="userName" outputVariable="userAddress"/>
18 <assign><copy><from variable="userAddress" part="address"
19     query="//city/"/><to variable="city"/></copy></assign>
20  <invoke name="invokeCity2GeoService" partnerLink=
21     "City2Geo" portType="city2geo:City2GeoSoap"
22     operation="GetLatLong" inputVariable="city"
23     outputVariable="zipInformation">
24   <source linkName="errorLink" transitionCondition=
25     "bpws:getVariableData('userAddress','address','//city/')!=
26     bpws:getVariableData('zipInformation','GetLatLongResult',
27     '//*[local-name()=&quot;City&quot;]')"/>
28 </invoke>
     [Part 4]
29 <assign><copy>
30 <from variable="userAddress" part="address" query="//zip/"/>
31 <to variable="zipOnly" part="zipCode"/>
32 </copy></assign>
33 <invoke name="invokeServiceAvailability"
34     partnerLink="ServiceAvailability"
35     portType="serviceavailability:CheckAvailabilityPortType"
36     operation="checkAvailability" inputVariable="zipOnly"
37     outputVariable="serviceAvailability"/>
38  <reply name="sendReply" partnerLink="User"
   portType="tns:DSLServicePT" operation="isServiceAvailable"
39     variable="serviceAvailability"/>
40 </process>
```

Figure 1. Example code in dslservice.bpel.

[Part 5]
```
41   <portType name="City2GeoSoap">
42    <operation name="GetLatLong">
43     <input message="s0:GetLatLongSoapIn" />
44     <output message="s0:GetLatLongSoapOut" />
45    </operation>
46   </portType>
```
[Part 6]
```
47   <message name="GetLatLongSoapOut">
48    <part name="parameters" element="s0:GetLatLongResponse"/>
49   </message>
```
[Part 7]
```
50   <s:element name="GetLatLongResponse">
51    <s:complexType><s:sequence>
52     <s:element name="GetLatLongResult"
53        type="s0:LatLongReturn" />
54    </s:sequence></s:complexType>
55   </s:element>
```
[Part 8]
```
56   <s:complexType name="LatLongReturn">
57    <s:sequence>
58    <s:element ... name="City" type="s:string" />
59    <s:element ... name="StateAbbrev" type="s:string" />
60    <s:element ... name="ZipCode" type="s:string" />
61    <s:element ... name="County" type="s:string" />
62    <s:element ... name="FromLongitude" type="s:decimal" />
63    <s:element ... name="ToLongitude" type="s:decimal" />
64    </s:sequence>
65   </s:complexType>
```

Figure 3. Example code in city2geo.wsdl.

[Part 9]
```
66   <xsd:complexType name="address">
67    <xsd:sequence>
68    <xsd:element name="state" type="xsd:string"/>
69    <xsd:element name="city" type="xsd:string"/>
70    <xsd:element name="streetNum" type="xsd:int"/>
71    <xsd:element name="streetName" type="xsd:string"/>
72    </xsd:sequence>
73   </xsd:complexType>
```

Figure 4. Example code in dslservice.wsdl.

BPEL Code. Figure 1 shows a BPEL code fragment. Part 1 defines the variable *zipInformation*. Part 2 defines a partner link named *City2Geo*. Part 3 shows a process that consists of two subparts. First, Part 3A defines two local variables *userName* and *userAddress*. Then, Part 3B shows a web service invocation. Web service *AddressBookLookup* is invoked, using *userName* as input and *userAddress* as output. Part 3B further queries the name of the city from the *address* part of the *userAddress* variable using XPath `//city/`, and assigns the query result to the variable *city*. The assignment is followed by the invocation of a web service *City2GeoService* using *city* as input and *zipInformation* as the variable to receive the output from the web service. Further verification is carried out to check the consistency between the city name in *userAddress* and that in *zipInformation*.

XPath1: '//*[local-name()='City']'
XPath2: '//city/'
XPath3: '//zip/'
XQ(Variable, XPathExp): XPath Query with input variable and XPath expression
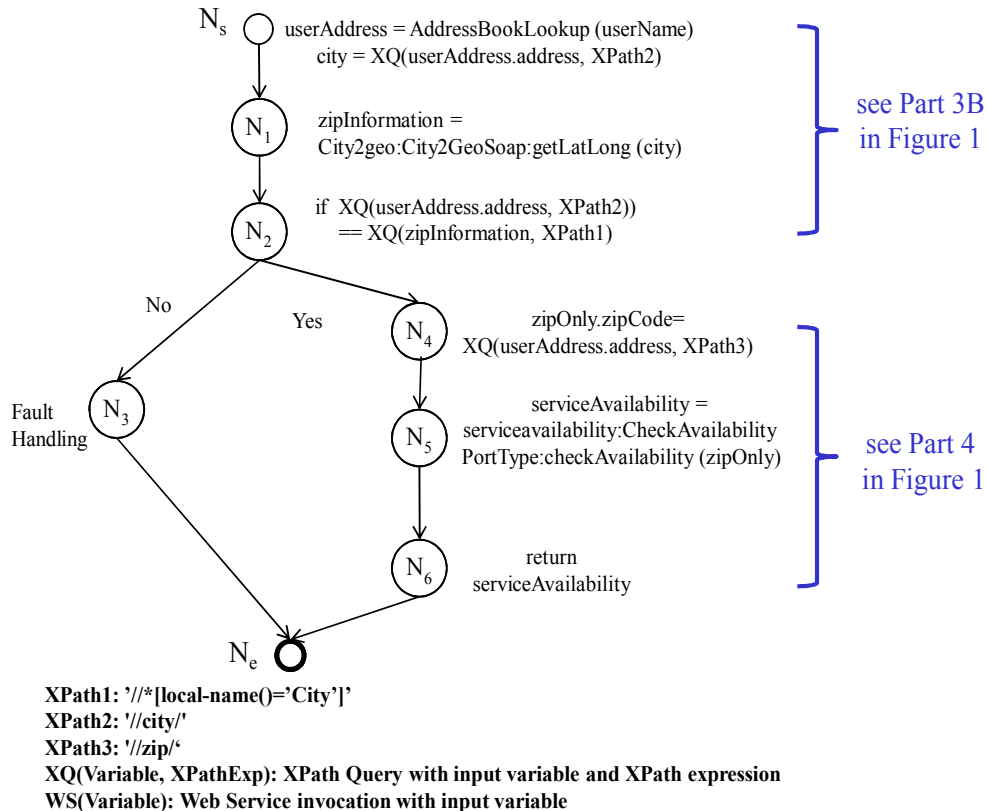WS(Variable): Web Service invocation with input variable

Figure 2. CFG for IsServiceAvailable.

Part 4 extracts the zip code from the address and keeps it in the variable *zipOnly.zipCode*. The code also invokes the web service *ServiceAvailability* to check whether the DSL service of the zip code is availability.

WSDL Documents. Figure 2 shows the WSDL content for the web service *City2Geo*, in which Parts 5, 6, 7, and 8 define a port type, a message type, a definition for the method *GetLatLongResponse*, and the type of the return data, respectively. Figure 3 shows the document model of the XML message of the DSL service, in which Part 9 defines the schema for type *address*.

Based on Table 1, we can translate a BPEL program (e.g., lines 1 to 40 in Figure 1) into a CFG (e.g., Figure 2). In Figure 2, N_i is a CFG node representing a BPEL statement. (Owing to the page limit, we do not list all the source code in Figure 1 for the CFG in Figure 2.)

For instance, N_s models the statements on lines 16 and 20. Similarly, N_2 models the state on lines 20 to 23. The edge $\langle N_1, N_2 \rangle$ models the state transition from N_1 to N_2 as coded. The state transitions on lines 24 to 27 are defined by two edges, namely $\langle N_2, N_3 \rangle$ and $\langle N_2, N_4 \rangle$, representing whether the error transition condition (line 24) is satisfied. Other nodes and edges can be interpreted in the same way.

Fault. A program fault occurs in Part 3B (on lines 24 to 27) and an illustration of the fault is as follows: Different cities may share the same name. For example, there are two cities known as *HuangShan* in Anhui, China. For ease of discussion, we refer to these two cities as *HuangShan_A* and *HuangShan_B*. When one uses the WS-BPEL application targeting to look for the zip code of *HuangShan_A* by inputting the city name "*HuangShan*", it may return the zip code of *HuangShan_B* instead. This is because the web service *City2Geo* may return a message containing these two cities, and yet the XPath in the BPEL code may select only one of them and assign it to the variable *zipOnly.zipCode*.

A quality application may provide a city list for users to pick the target city [19]. However, given the above motivating example, and without revealing a relevant failure, identifying the fault is hard in the first place.

C. Frankl-Weyuker Data Flow Testing

In this section, we revisit the definitions in [6] for the concepts related to conventional data flow testing.

Let n, m_1, …, m_r, n' be nodes in a CFG. A path $\langle n, m_1, …, m_r, n' \rangle$ is said to be *def-clear* with respect to the variable x when none of m_1, …, m_r defines x. A *def-use association* is a triple $\langle x, n_d, n_u \rangle$ such that the variable x is defined at node n_d and used in node n_u, and there is a def-clear path with respect to x from n_d to n_u. We refer readers to [6] for more details.

Based on the CFG in Figure 2, one can compute the test requirements of the *all-uses* criterion [6]. A test suite is said to satisfy the *all-uses* criterion if and only if the test suite executes every def-use association of the program unit at least once.

III. EMPIRICAL STUDY

This section reports our empirical study that evaluates the effectiveness of applying the Frankl-Weyuker data flow testing criteria to WS- BPEL programs. We have explained in Section I why the *all-uses* criterion has specifically been chosen for experimentation.

A. Experimental Setup

In the experiment, we use 8 open-source WS-BPEL programs that are publicly available on the Internet [2], namely atm [3], buybook [22], dslservice [3], gymlocker [3], loanapproval [3], marketplace [3], purchase [3], and TripHandling [3]. For instance, loanapproval and buybook are used in the sample projects to introduce WS-BPEL programs by IBM and Oracle, respectively. These programs have also been frequently used in previous WS-BPEL research.

To create faulty versions, we invited developers, who are non-authors and have experience in developing service-oriented applications, to seed faults to create multiple single-fault versions of each WS-BPEL application. We did not impose any restriction to the developers on the nature of faults injected. In total, 60 faults were injected in these subject applications.

We developed a prototype tool to randomly generate a test pool of 1000 test cases for each application. We adopted the following process to generate a test suite for the all-uses criterion: For each target version of an application, the tool randomly selects a test case from the appropriate test pool and executes it on the target version. The tool adds the test case to the test suite under construction only if this test case increases the coverage (with respect to the all-uses criterion) [6][12][15] of the test suite. After a number of tryouts, we set the process to terminate if either 100% coverage of the criterion had been attained, or an upper bound of 200 trials had been reached.

The tool used the outputs of test case executions on the original programs as test oracles. In other words, for each test case, it compared the output of a faulty version with that of the original program to tell whether a failure has been detected.

We repeated the test suite construction process 100 times to obtain 100 test suites for each benchmark application. We executed the experiment on Dell GX260, Pentium 4 CPU 2.26 GHz, 512 M RAM.

To collect various statistics information for evaluation, we developed a simple WS-BPEL simulation engine for WS-BPEL programs. The simulator was designed to test WS-BPEL programs and to meet the BPEL specification requirements in executing the subject programs. We referred to the basis path testing approach [31] to implement our simulation engine.

Finally, WS-BPEL programs may, in general, have internal concurrency, and their data associations among data flow entities are different from the sequential counterparts when programs are executed concurrently [30]. In the ex-

periment, we applied the notion of forced deterministic testing for concurrent programs [12] to conduct the evaluation.

B. Data Analysis

We present the empirical results in this section. We first summarize the coverage percentage of *all-uses* associations for each benchmark program in Table 2. The table shows the minimal, average, and maximal coverage that can be achieved by the 100 test suites on each program. We observe that, on average, these test suites achieve high coverage.

TABLE 2. TEST SUITE COVERAGE STATISTICS

Application	Min.	Avg.	Max.
atm	0.941	0.969	1.000
buybook	0.923	0.923	0.923
dslservice	0.824	0.824	0.824
gymlocker	1.000	1.000	1.000
loanapproval	1.000	1.000	1.000
marketplace	1.000	1.000	1.000
purchase	0.833	0.833	0.833
triphandling	1.000	1.000	1.000
Overall	0.940	0.944	0.947

We further analyze the fault-detection capability of the testing criterion. We partition the 60 faults into three categories (in-BPEL, in-XPATH, and in-WSDL) according to the types of artifacts that the individual faults reside. There are 21 faults in BPEL programs, 21 faults on XPath Query, and 18 faults in WSDL documents, respectively. The fault-detection rate [11] of a test suite for a category of faults is defined by X/Y, where X is the number faults detected by the test suite and Y is the total number of faults in the category.

Hutchins et al. [11] have concluded that a test set with full def-use coverage (100%) is much more valuable than that with lower def-use coverage (such as 90% or 95%). Therefore, we follow [11] to use the full all-uses coverage (100%) test sets to examine their fault-detection rates. The results are listed in Table 3.

TABLE 3. FAULT-DETECTION RATES BY CATEGORIES
(ON TEST SUITES WITH 100% ALL-USES COVERAGE)

Category	Fault-Detection Rate		
	Min.	Avg.	Max.
in-BPEL	0.727	0.916	1.000
in-XPath	0.547	0.737	1.000
in-WSDL	0.500	0.688	1.000
Overall	0.600	0.790	1.000

The table shows that the *all-uses* criterion has a much higher fault-detection rate in revealing failures due to the faults on the BPEL artifacts than revealing those in either the XPath or WSDL artifacts when using test sets with 100% all-uses coverage. The former is better than the latter two by 18% and 23%, respectively. As the all-uses criterion is generally considered to be effective, the results show that the performance difference on detecting different categories of faults is huge. It shows that data flow testing criteria that covers BPEL code without considering other artifacts is not highly effective in detecting failures owing to the presence of faults in the latter kinds of artifacts.

C. Threats to Validity

This section discusses the threats to validity of the experiment.

We use a set of open-source programs as subjects. This enables us to access the source code for applying data flow testing. On the other hand, the number of subject programs and their sizes are not large. In the future, we will gain more insights by finding larger subject programs to study how the fault-detection effectiveness of testing criteria may vary with the increase of program size.

We also observe from Table 2 that the maximum coverage achieved by some test suites may not reach 100%. As explained in Section III-A, we use 200 trials as the upper bound for adding more test cases to the test suite under construction. This parametric value (200) is chosen after several tryouts during initial experimentation. We observe that the overall average coverage is already high. Therefore, we do not further increase the number of trials. However, the results of fault-detection rates are likely to change when higher coverage can be achieved. At this stage, we do not have effective means to construct such test cases. In the future, we should study the effect of different levels of coverage.

In addition, in this paper, we are mainly concerned with three categories of faults, namely BPEL faults, XPath faults, and WSDL faults. Other faults such as web service faults have not been studied in the experiment. We thus assume flawless messages (related to interactions with other web services) in the experiment. Also, the CFG model and the execution of WS-BPEL are based on the perspective of sequential programs, so that we use forced deterministic testing to execute programs. The use of other execution models may change the experimental results. In the future, we should investigate how the relaxation of the above assumptions may affect the testing of WS-BPEL applications.

IV. RELATED WORK

This section reviews the literature related to the testing of WS-BPEL programs.

First, we review the research related to WS-BPEL in general. WS-BPEL is an area with active research studies. Such research studies can be grouped briefly into several interesting subareas, including WS-BPEL modeling and verification, interoperability analysis between BPEL programs and web services, and WS-BPEL test case generation.

The subarea of WS-BPEL modeling and verification has been studied by many researchers. Take the work of Mongiello et al. [21] as an example. They propose to use formal methods to construct a model and formalize the correctness properties about the reliability of business process design methods. Since WS-BPEL programs are

often too complex to be formalized adequately, a lot of research just focuses on the finite state machine (FSM) of the WS-BPEL specification. However, such techniques may be ineffective in revealing faults related to the WSDL and XPath artifacts. Our experiment verifies this point.

The modeling of service composition has been studied in [5][7]. They analyze the interactions between BPEL programs and web services using WS-BPEL as the specification. Analysis tools such as WSAT [25] have been developed to conduct formal analysis of web services. For instance, Mandell and McIlraith [18] propose a bottom-up analysis approach that describes the interactions among web services. However, they have not studied XPath and WSDL documents.

The selection of test cases for WS-BPEL programs is another valuable research area for testing WS-BPEL programs. Some researchers (such as García-Fanjul et al. [9]) treat a WS-BPEL program as a finite state machine and use mutation analysis to generate faulty versions. In [9], they check each mutated program against a given temporal property using SPIN, and transform the counterexample (containing violations of the property) thus generated to specify test cases. Some researchers consider the testing of the concurrency aspect of WS-BPEL programs [29]. They model WS-BPEL as a set of concurrent finite state machines, and use a heuristic approach to conduct reachability analysis to find concurrent paths for the set of FSMs, and use such concurrent paths as test cases.

In the rest of this section, we review related data flow testing research. Souter and Pollack [24] propose a way to construct contextual def-use associations, in which each definition and use of an object is associated with contexts. The effect of environment information in pervasive context-aware software is studied in [16]. They take the environment information relevant to an application program as contexts, and then propose context-aware data flow associations and testing criteria. Their approach is verified in an experiment on an RFID-based location sensing software. In this paper, we study the capability of conventional data flow testing (particularly the *all-uses* criterion) in testing service-oriented business (WS-BPEL) applications.

A family of test adequacy criteria that are used to assess the quality of test suites for database-driven applications is proposed by [13]. Their test adequacy criteria also cover data associations between database-driven applications and the environment (that is, the database). However, unlike our approach in constructing a CFG to parse the domain structure, they do not consider how to parse a query in their database domain model.

Other related work on test adequacy problems has been reviewed in [16][32]. In particular, Lu et al. [17] studies how to test programs when its contexts may be affected by connected services. Mei et al. [19] presents a new data structure to quantify XPath and new testing criteria to assure WS-BPEL programs. The effectiveness of conventional data flow testing criteria has not been reported.

V. CONCLUSION

In service–oriented testing, many existing proposals conjecture that conventional techniques are ineffective and aim to develop new ones. There is, however, little empirical evidence in the literature to show that existing techniques are indeed ineffective.

In this paper, we report an empirical study of how well the Frankl-Weyuker data flow testing criteria (particularly the *all-uses* criterion) reveal the presence of faults in service-oriented implementations. We use a set of WS-BPEL programs to evaluate the *all-uses* criterion at the WSDL level, which considers both BPEL code and WSDL artifacts. Our experiment finds that the *all-uses* criterion is noticeably less effective in revealing the faults in XPath and WSDL artifacts than revealing those in BPEL code. This preliminary evidence provides a solid justification to develop new techniques to assure service-oriented applications. More generally, our experiment indicates that testing techniques that are developed on top of executable artifacts may be ineffective in revealing the faults in non-executable artifacts in general applications.

For future work, we plan to study the test case generation problem for WS-BPEL applications. We also plan to study the relationship between the level of coverage and the fault-detection rate for such applications.

REFERENCES

[1] B. Benatallah, R. M. Dijkman, M. Dumas, and Z. Maamar. Service composition: concepts, techniques, tools and trends. In *Service-Oriented Software System Engineering: Challenges and Practices*, Stojanovic Z. and Dahanayake A., Eds., pages 48–66. Idea Group, Hershey, PA, 2005.

[2] BPEL Code Samples. Available at `http://www.bpelsource.com/resources/code.html`. (Last access on April 30, 2009.)

[3] *BPWS4J: a Platform for Creating and Executing BPEL4WS Processes*. Version 2.1. IBM, 2002. Available at `http://www.alphaworks.ibm.com/tech/bpws4j`.

[4] F. Curbera, R. Khalaf, F. Leymann, and S. Weerawarana. Exception handling in the BPEL4WS language. In *Business Process Management*, W. M. P. van der Aalst et al., Eds., volume 2678 of Lecture Notes in Computer Science, pages 276–290. Springer, Berlin, Germany, 2003.

[5] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of Web service compositions. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering* (*ASE 2003*), pages 152–161. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[6] P. G. Frankl and E. J. Weyuker. An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering*, 14 (10): 1483–1498, 1988.

[7] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL Web services. In *Proceedings of the 13th International Conference on World Wide Web* (*WWW 2004*), pages 621–630. ACM Press, New York, NY, 2004.

[8] X. Fu, T. Bultan, and J. Su. Model checking XML manipulating software. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA 2004*), pages 252–262. ACM Press, New York, NY, 2004.

[9] J. Garcia-Fanjul, J. Tuya, and C. de la Riva. Generating test cases specifications for BPEL compositions of Web services using SPIN. In *Proceedings of the International Workshop on Web Services: Modeling and Testing* (*WS-MaTe 2006*), pages 83–94. Palermo, Sicily, Italy, 2006.

[10] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and XML typing. *Journal of the ACM*, 52 (2): 284–335, 2005.

[11] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th International Conference on Software Engineering* (*ICSE '94*), pages 191–200. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[12] G. H. Hwang, K.-C. Tai, and T. L. Huang. Reachability testing: an approach to testing concurrent software. *International Journal of Software Engineering and Knowledge Engineering*, 5 (4): 246–255, 1995.

[13] G. M. Kapfhammer and M. L. Soffa. A family of test adequacy criteria for database-driven applications. In *Proceedings of the Joint 9th European Software Engineering Conference and 11th ACM SIGSOFT International Symposium on Foundation of Software Engineering* (*ESEC 2003/ FSE-11*), pages 98–107. ACM Press, New York, NY, 2003.

[14] C. Koch. On the role of composition in XQuery. In *Proceedings of the 8th International Workshop on the Web and Databases* (*WebDB 2005*), pages 37–42. Baltimore, Maryland, 2005.

[15] Z. Li, W. Sun, Z. B. Jiang, and X. Zhang. BPEL4WS unit testing: framework and implementation. In *Proceedings of the IEEE International Conference on Web Services* (*ICWS 2005*), pages 103–110. IEEE Computer Society Press, Los Alamitos, CA, 2005.

[16] H. Lu, W. K. Chan, and T. H. Tse. Testing context-aware middleware-centric programs: a data flow approach and an RFID-based experimentation. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (*SIGSOFT 2006/FSE-14*), pages 242–252. ACM Press, New York, NY, 2006.

[17] H. Lu, W. K. Chan, and T. H. Tse. Testing pervasive software in the presence of context inconsistency resolution services. In *Proceedings of the 30th International Conference on Software Engineering* (*ICSE 2008*), pages 61–70. ACM Press, New York, NY, 2008.

[18] D. J. Mandell and S. A. McIlraith. Adapting BPEL4WS for the semantic Web: the bottom-up approach to Web service interoperation. In *Proceedings of the 2nd International Semantic Web Conference* (*The Semantic Web—ISWC 2003*), volume 2870 of Lecture Notes in Computer Science, pages 227–241. Springer, Berlin, Germany, 2003.

[19] L. Mei, W. K. Chan, and T. H. Tse. Data flow testing of service-oriented workflow applications. In *Proceedings of the 30th International Conference on Software Engineering* (*ICSE 2008*), pages 371–380. ACM Press, New York, NY, 2008.

[20] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51 (1): 2–45, 2004.

[21] M. Mongiello and D. Castelluccia. Modelling and verification of BPEL business processes. In *Proceedings of the 4th Workshop on Model-Based Development of Computer-Based Systems and the 3rd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software* (*MBD-MOMPES 2006*), pages 144–148. IEEE Computer Society Press, Los Alamitos, CA, 2006.

[22] Oracle BPEL Process Manager. Oracle Technology Network. Available at `http://www.oracle.com/technology/products/ias/bpel/`. (Last access on April 30, 2009.)

[23] C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36 (10): 46–52, 2003.

[24] A. L. Souter and L. L. Pollock. The construction of contextual def-use associations for object-oriented systems. *IEEE Transactions on Software Engineering*, 29 (11): 1005–1018, 2003.

[25] Web Service Analysis Tool. University of California, Santa Bababra, CA. Available at `http://www.cs.ucsb.edu/~su/WSAT`. (Last access on April 30, 2009.)

[26] *Web Services Business Process Execution Language Version 2.0*. OASIS, 2007. Available at `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

[27] Web Services Invocation Framework: DSL Provider Sample Application. Available at `http://svn.apache.org/viewvc/webservices/wsif/trunk/java/samples/dsl provider/README.html?view=co`. (Last access on April 30, 2009.)

[28] *XML Path Language* (*XPath*) *Recommendation*. World Wide Web Consortium, 2007. Available at `http://www.w3.org/TR/xpath20/`.

[29] J. Yan, Z. Li, Y. Yuan, W. Sun, and J. Zhang. BPEL4WS unit testing: test case generation using a concurrent path analysis approach. In *Proceedings of the 17th International Symposium on Software Reliability Engineering* (*ISSRE 2006*), pages 75–84. IEEE Computer Society Press, Los Alamitos, CA, 2006.

[30] R.-D. Yang and C.-G. Chung. Path analysis testing of concurrent programs. *Information and Software Technology*, 34 (1): 43–56, 1992.

[31] G. Zhang, R. Chen, X. Li, and C. Han. The automatic generation of basis set of path for path testing. In *Proceedings of the 14th Asian Test Symposium* (*ATS 2005*), pages 46–51. IEEE Computer Society Press, Los Alamitos, CA, 2005.

[32] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29 (4): 366–427, 1997.