

Table of contents

Table of contents

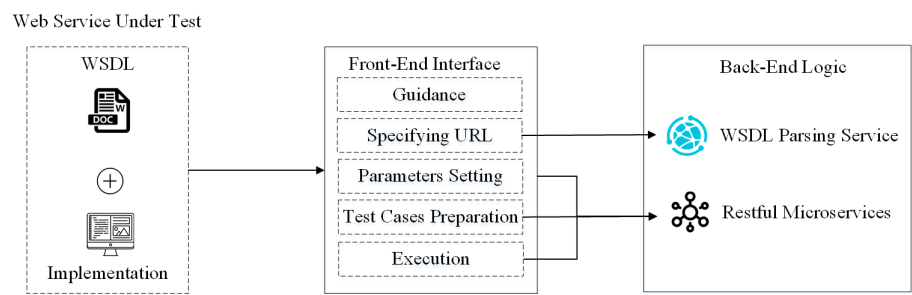
DRTester

- Framework
 - Front-end interface
 - Back-end logic
 - WSDL parsing
 - Restful Micro-Services
- Configuration
 - Configuration for front-end interface
 - Configuration for back-end logic
- An example
 - The specification of web service under test
 - Step 1: Specifying url and setting parameters
 - Step 2: Partition construction and parameter setting
 - Step 3: Test case preparation
 - Step 4: Test case execution

DRTester

The prototype tool *DRTester* supports dynamic random testing technique for web service testing. We describe the implementation and usage of the tool in detail.

Framework



The above figure illustrates the *DRTester* framework comprising four main components, corresponding to web service under test; the *interface* between the user and *DRTester*; WSDL parsing service is responsible to parse WSDL file of web service under test; and Restful micro-services that are used to divide input domain, generate test cases, execute test case, and send information to *interface*.

We next examine each component in the framework individually.

Front-end interface

We developed a HTML page by using the Vue framework (<https://cn.vuejs.org/>), its source codes are available by visiting <https://github.com/phantomDai/DRTester.git>.

This *interface* wraps the setting information in the HTTP messages, and sends them to the Restful micro-services that are not only responsible for communicating with this *interface* but also wrap the selected test cases in SOAP messages, and send them to the web service under test.

Back-end logic

In this section, we describe the implementation of back-end logic, which comprises two parts: 1) a web service that is responsible for parsing WSDL file of web service under test; 2) several Restful micro-services.

WSDL parsing

We obtain the necessary information by parsing WSDL of web service under test to generate test cases and automatically invoke interested methods of web service under test. Accordingly, a web service has been developed to acquire information about the names and types of interested methods of web service under test, along with their parameters information (name and type). Besides, we also made this web service publicly accessible (<https://github.com/phantomDai/parseesdlws.git>).

Restful Micro-Services

The Restful micro-services (For more details, please visit linkage: <https://github.com/phantomDai/drtAPI.git>) are responsible for communicating HTTP messages to and from the front-end interface. Besides, these micro-services need to update the test profile according to the test results, and select test cases from the partitions. The selected test cases then are wrapped in SOAP messages and sent to the web service under test through the proxy class.

Configuration

This section describes the configuration of the front-end interface and back-end logic.

Configuration for front-end interface

The users need to set up the local environment as follows:

1. download and install *node.js* (please visit linkage: <https://nodejs.org/en/>)
2. execute the following command in DOS (if not in China, please ignore this step):

```
npm install -g cnpm --  
registry=https://registry.npm.taobao.org
```

3. execute the following command in DOS:

```
npm install vue -g
```

4. execute the following command in DOS:

```
npm install vue-cli -g
```

After the front-end environment is configured, the user can download the source code by visiting the linkage: <https://github.com/phantomDai/DRTester.git>. Next, users need to go to the root of the downloaded file, and create a directory named "node_modules". Then please execute the following command in DOS.

```
npm install (if in China, please execute command: cnpm install)
```

Finally, users need to find all *post* or *get* methods in *BaseTable.vue* and *Tabs.vue*, and change the values of *url* to replace the IP address with user's IP. For instance, the user replace the current value of *url* (on the line 266 of *BaseTable.vue*)

```
url: 'http://202.204.62.171:8082/api/parse/wsd1' (current  
IP address)
```

with

```
url: 'http://xxx.xxx.xxx.xxx:8080/api/parse/wsd1' (user's  
IP address)
```

Finally, the user executes the following command and input "<http://localhost:8080>" in her browser.

```
npm run dev
```

The first page of front-end interface is *Guidance* (as shown in following figure), where we describe the steps and rules the tester should follow when testing a web service.

Guidance

Table of Contents

- Overview
- WSDL Parsing
- Parameters Setting
- Partition Construction and DRT Parameter Setting
- Test Cases Preparation

Overview

Considering the principle of DRT and the features of web services, we propose a DRT for web services framework, as illustrated in blow. In the figure, the DRT components are inside the testing box, the practitioner interaction is represented in the initialization box , and the web services under test are located outside. Interactions between DRT components, practitioner and the web services are depicted in the framework.

The second page of front-end interface is *Configuration* (as shown in following figures), where users need to provide some information to partition input domain and generate test cases.

Q Specifying URL

Please enter the address of the web service under test

Address

Parse

P Parameters Setting

Please select an operator:

Operation

Index	Parameter	Type	Options
Empty			

Save

Partition Construction and Parameter Setting

Please input option combinations for partition construction and set parameters for DRT:

Partition	Option Combination	Test Profile	Adjusting Factor
partition	<input type="text" value="choices"/>	<input type="text" value="profile"/>	<input type="text" value="value"/>

+ Add

− Delete

✔ Save

Test Cases Preparation

Please select a method to generate a test suite:

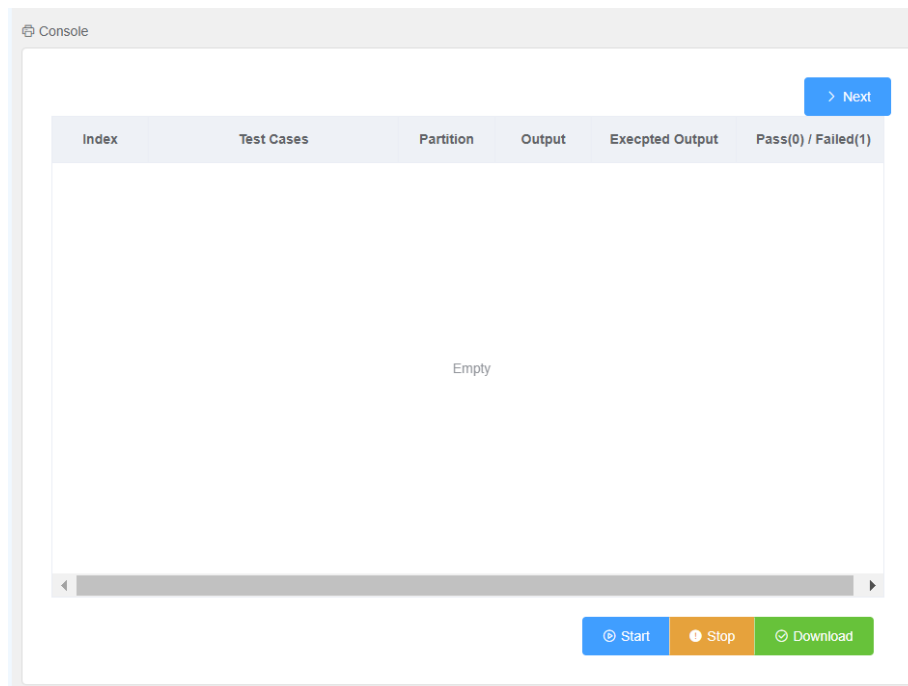
☒ Randomly Generate Test Suite

☐ Upload Test Suite File

Please set the number of test cases to be generated:

Please upload an XML file that contains test cases:

The third page of front-end interface is *Execution* (as shown in the following figures), where users can control the execution of test cases and download test report.



Configuration for back-end logic

The user needs to set up the local environment as follows:

1. Tomcat 9.06 (that is available in the repository: <https://github.com/phantomDai/parseesdlws.git>)
2. JDK 1.8.0_161 (that is available in the linkage: <https://www.oracle.com/technetwork/java/javase/downloads/index.html>)
3. IntelliJ IDEA (that is available in the linkage: <http://www.jetbrains.com/>)

Source code of this service can be downloaded from the provided linkage.

The default port of this web service is 8085. If changing the port of this web service, the user needs to change the value of parameter *endpoint* in *ParseWSDL* script that is available in linkage: <https://github.com/phantomDai/drtAPI.git>.

In the *parseWSDL* class, there is a member variable named *endpoint*. Users need to change the value of this variable according to their IP and above port. For instance, we set *endpoint* value according to our IP and port:

```
private static String endpoint =  
"http://202.204.62.171:8085/services/parser?wsdl"
```

As for Restful APIs, the users need to set up the local environment as follows:

1. Maven (that is available by visiting the linkage: <http://maven.apache.org/>)

Then, the users can execute the following command in the drtAPI directory.

```
mvn clean package -Dmaven.test.skip=true
```

Finally, users can go to the "target" directory and execute the following command.

```
java -jar ./drt-0.01-SNAPSHOT.jar
```

Congratulations! you have finished the configurations of environments.

An example

We use an example to demonstrate web service testing using our prototype tool.

The specification of web service under test

Aviation consignment management service (ACMS) (that is available by visiting the linkage: <https://github.com/phantomDai/drt4ws>) helps airline companies check the allowance (weight) of free baggage, and the cost of additional baggage. Based on the destination, flights are categorised as either domestic or international. For international flights, the baggage allowance is greater if the passenger is a student (30kg), otherwise it is 20kg. Each aircraft offers three cabins classes from which to choose (economy, business, and first), with passengers in different classes having different allowances.

Step 1: Specifying url and setting parameters

The user first needs to enter the address of the WSDL of web service under test, and click "Parse" button, and then select a method of web service under test in the following drop-down menu (as shown in following figures).

The screenshot displays a web application interface for configuring a web service test. It is divided into two main sections: "Specifying URL" and "Parameters Setting".

Specifying URL: This section contains a text input field with the placeholder "Please enter the address of the web service under test". The field contains the URL "http://202.204.62.171:8081/services/acms?wsdl". To the right of the input field is a blue button labeled "Parse".

Parameters Setting: This section contains a text input field with the placeholder "Please select an operator:". Below this field is a dropdown menu that is currently open, showing three options: "prearea", "feeCalculation", and "preairclass". To the right of the dropdown menu is a green button labeled "Save".

The user divides each parameter into disjoint options, and describe them according to predefined rules that are introduced in *Guidance* page (as shown in the following figure). After the "Save" button is clicked, parameters and corresponding options are sent to some Restful API.

Please select an operator:

feeCalculation			
Index	Parameter	Type	Options
1	area	int	1-1:{0};1-2:{1};1-3:{2}
2	airClass	int	2-1:{0};2-2:{1};2-3:{2}
3	luggage	double	3-1:[0,60];3-2:(60,300
4	economicfee	double	4-1:{0};4-2:(0,3000)
5	isStudent	boolean	5-1:{true};5-2:{false}

Step 2: Partition construction and parameter setting

The user divides input domain by combining options with different parameters of selected method (as shown in following figure). Besides, the user need to set the selecting probability for each partition, and the value of probability adjusting factor. After clicking the "Save" button, all provided information will be sent to some Restful API, which is responsible for initializing test profile, setting the value of *epsilon*, and dividing input domain.

Partition Construction and Parameter Setting

Please input option combinations for partition construction and set parameters for DRT:

Partition	Option Combination	Test Profile	Adjusting Factor
partition	2-1:{0};5-1:{true}	0.125	0.05
partition	2-2:{1};5-1:{true}	0.125	
partition	2-3:{2};5-1:{true}	0.125	
partition	2-4:{3};5-1:{true}	0.125	
partition	2-1:{0};5-2:{false}	0.125	
partition	2-2:{1};5-2:{false}	0.125	

+ Add

- Delete

Save

Step 3: Test case preparation

We provide two ways to generate test cases: 1) Randomly generate test cases; 2) Upload Json file that include test cases. Note that there are rules about the format of the uploaded Json file, which are described in *Guidance* page.


⚙️ Test Cases Preparation

Please select a method to generate a test suite:


☒ Randomly Generate Test Suite

☐ Upload Test Suite File

Please set the number of test cases to be generated:

 Generate

Please upload an XML file that contains test cases:

 Upload

Step 4: Test case execution

After performing all the steps above, all necessary stuff for testing are ready. The user first clicks *start* button, then the table in the middle of page shows the information of testing. If test cases are generated using random strategy, users must decide whether the last test case detected a fault. If a test case detected a fault, the user needs to change the value of the last row and last column in the table to "1". Then, next test case is executed by click "Next" button. The "Stop" button is responsible for sending a signal which means testing task is finished. Users also can download testing report that records the information of testing process.

📄 Console

> Next

Index	Test Cases	Partition	Output	Exepected Output	Pass(0) / Failed(1)
1	{"area":1,"airClass":0,"luggage":2527.7,"economicfee":0.0,"isStudent":true}	0	0	NA	<input type="text" value="0"/>
2	{"area":3,"airClass":1,"luggage":2313.0,"economicfee":0.0,"isStudent":false}	5	0	NA	<input type="text" value="0"/>

⏪

⏩

🔄 Start

⏹ Stop

📄 Download