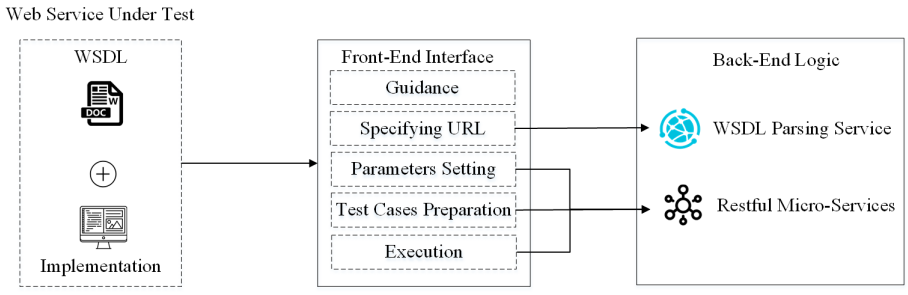
Table of contents

DRTester

The prototype tool *DRTester* supports dynamic random testing for web service testing. In this document, 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: (1) web service under test; (2) the front-end interface (*interface* for short) between the user and *DRTester*; (3) WSDL parsing service which is responsible for parsing the WSDL file of the web service under test; and (4) Restful micro-services that are used to divide the input domain, generate test cases, execute test cases, and send information to the *interface*.

We next examine each component in the framework individually.

Front-end interface

We developed three HTML pages using the Vue framework[^1]: https://cn.vuejs.org/, their source codes are accessible through the following link: https://github.com/phantomDai/DRTester.git.

This *interface* wraps the setting information in the HTTP messages, and sends them to several 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 the back-end logic, which comprises two parts: 1) a WSDL parsing web service; and 2) several Restful micro-services.

WSDL parsing service

We obtain the necessary information by parsing the WSDL file of the web service under test to generate test cases and automatically invoke interested methods of the web service under test. Accordingly, a web service has been developed to acquire information about the interested methods (such as names and types of return value), along with their parameter information (names and types). We have also made this web service publicly accessible through the following link: https://github.com/phantomDai/parseesdlws.git.

Restful Micro-Services

The Restful micro-services (For more details, please visit: https://github.com/phantomDai/drtAPI.git) are responsible for communicating HTTP messages to and from the front-end interface. 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: 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 codes from:
https://github.com/phantomDai/DRTester.git.
Next, the user needs to go the the root directory of the downloaded file, and create a directory named "node_modules". Then the user needs to execute the following command in DOS.

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

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

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

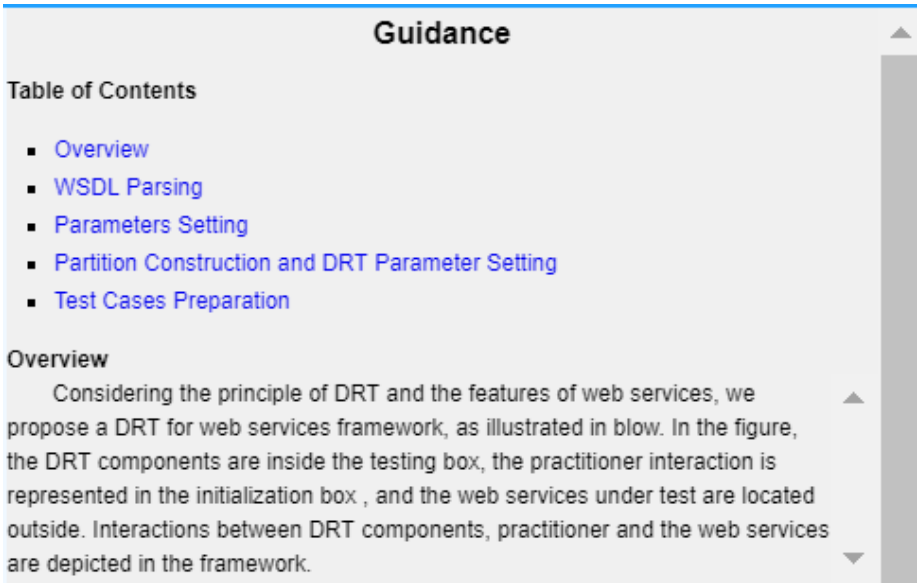
with

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

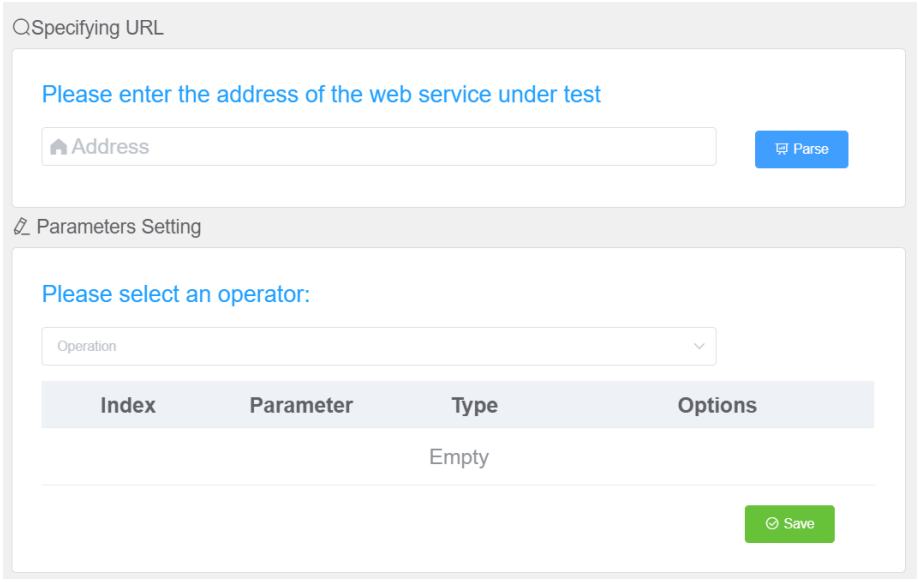
Finally, the user executes the following command in DOS and inputs "http://localhost:8080" in the browser.

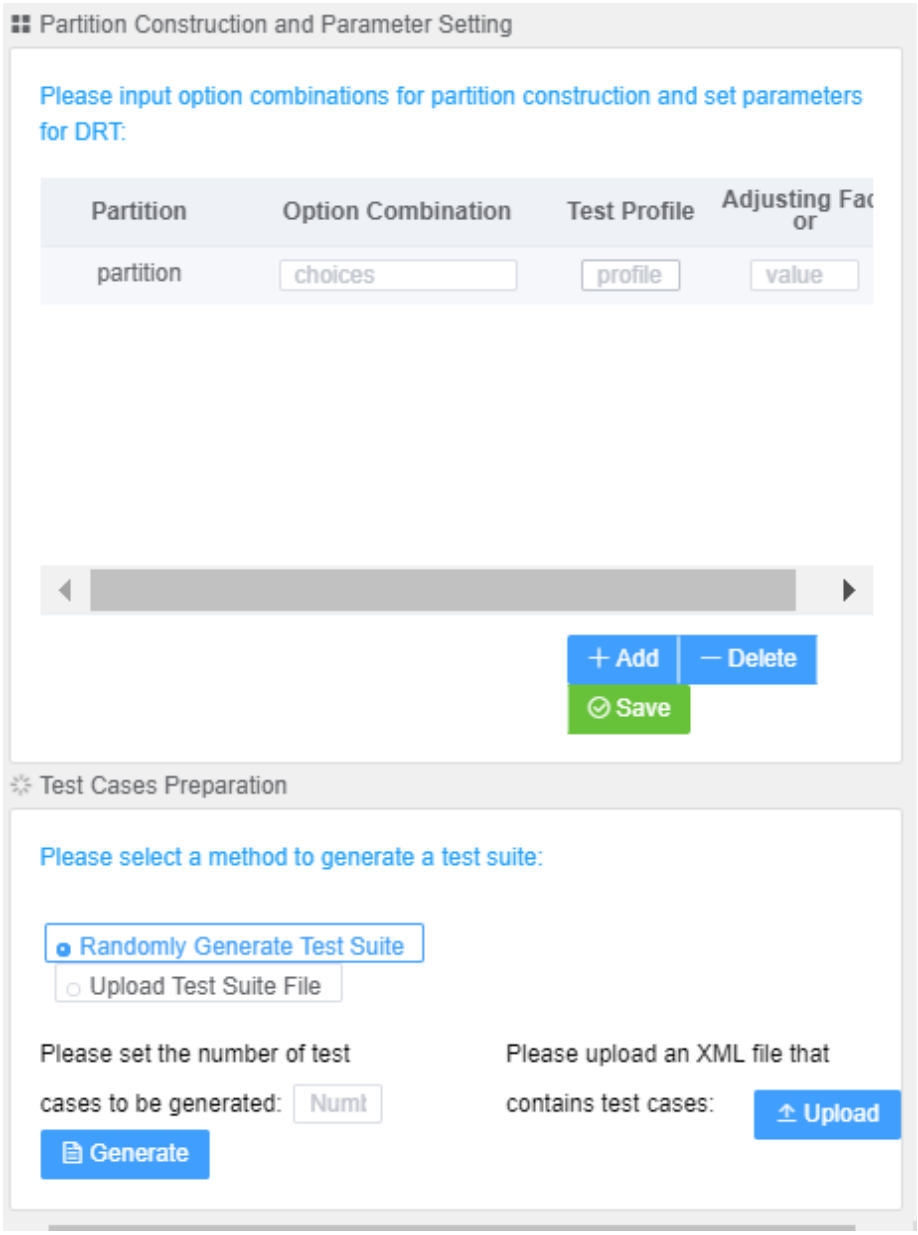
```
npm run dev
```

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

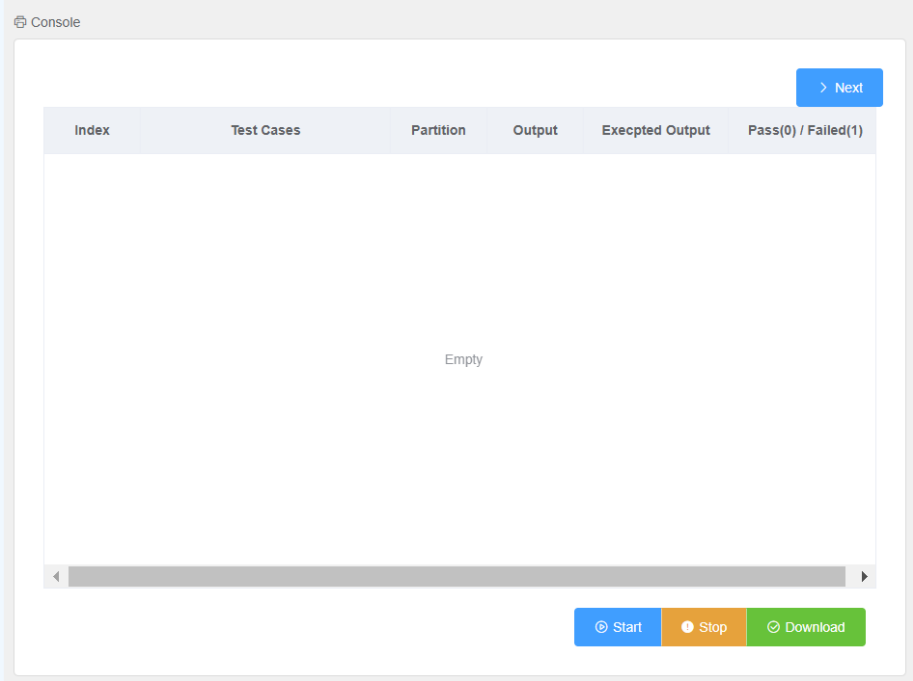


The second page of the front-end interface is *Configuration* (as shown in following figures), *where the user needs to provide some information to partition the input domain and generate test cases*.





The third page of the front-end interface is *Execution* (as shown in the following figures), *where the user controls the execution of the test cases and downloads the test report.*



Configuration for back-end logic

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

1. Tomcat 9.06 (available in the repository:

   https://github.com/phantomDai/parseesdlws.git)
2. JDK 1.8.0_161 (available from:

   https://www.oracle.com/technetwork/java/javase/downloads/index.htm)
3. IntelliJ IDEA (available from:

   http://www.jetbrains.com/)

the source codes of the WSDL parsing service are available from the following link:
https://github.com/phantomDai/parseesdlws.git.

The default port of this service is *8085*. If changing the port of this web service, the user needs to change the value of the variable *endpoint* in the *ParseWSDL* script that is available from:
https://github.com/phantomDai/drtAPI.git.
For instance, the user may replace the current value of *endpoint*

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

with

```
private static String endpoint =
"http://202.204.62.171:XXXX/services/parser?wsdl" (user's
port)
```

As for the Restful micro-services, the user needs to set up the local environment as follows:

1. Maven (available from: http://maven.apache.org/)

Then, the user executes the following command in the root directory of the drtAPI file that is available from: https://github.com/phantomDai/drtAPI.git.

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

Finally, the user goes to the "target" directory and executes the following command in DOS.

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

Congratulations! You have finished configuring the environment.

An example

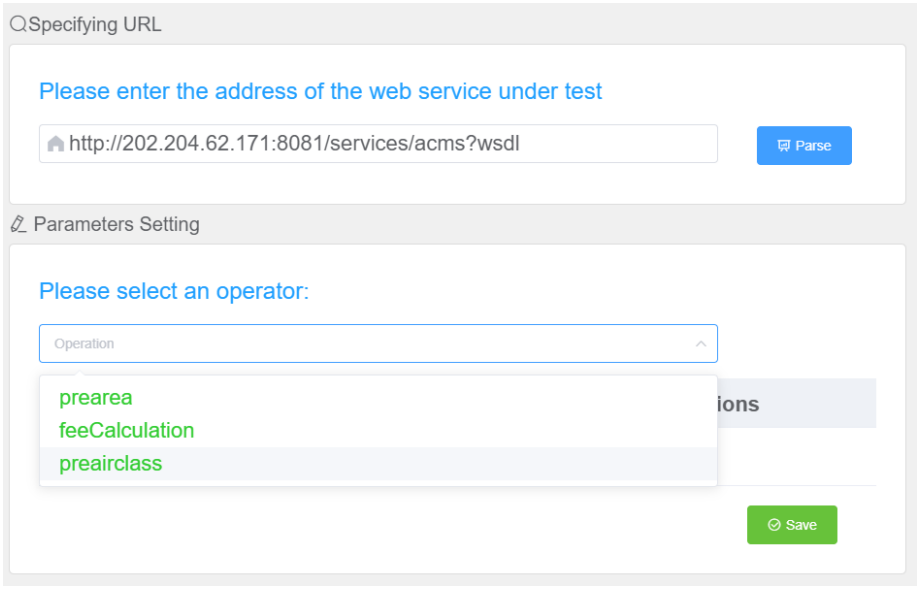
We use an example to demonstrate web service testing using our prototype tool.

Specification of the web service under test

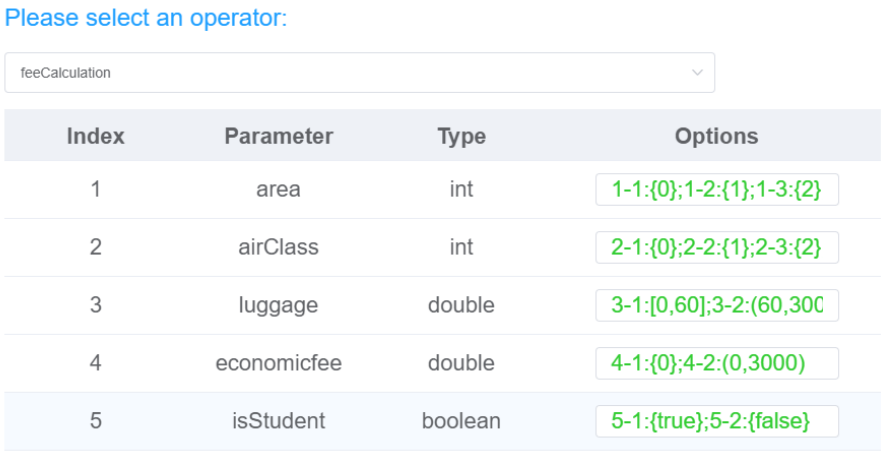
Aviation consignment management service (ACMS) (available from: https://github.com/phantomDai/drt4ws) helps airline companies check the allowance (weight) of free baggage, and the cost of additional baggage. Based on the destination, flights are categorised as either domestic or international. For international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

Step 1: Specifying the url and setting parameters

The user first enters the address of the WSDL of the web service under test, clicks the "Parse" button, and then selects the interested method of the web service under test from the following drop-down menu (as shown in the following figure).

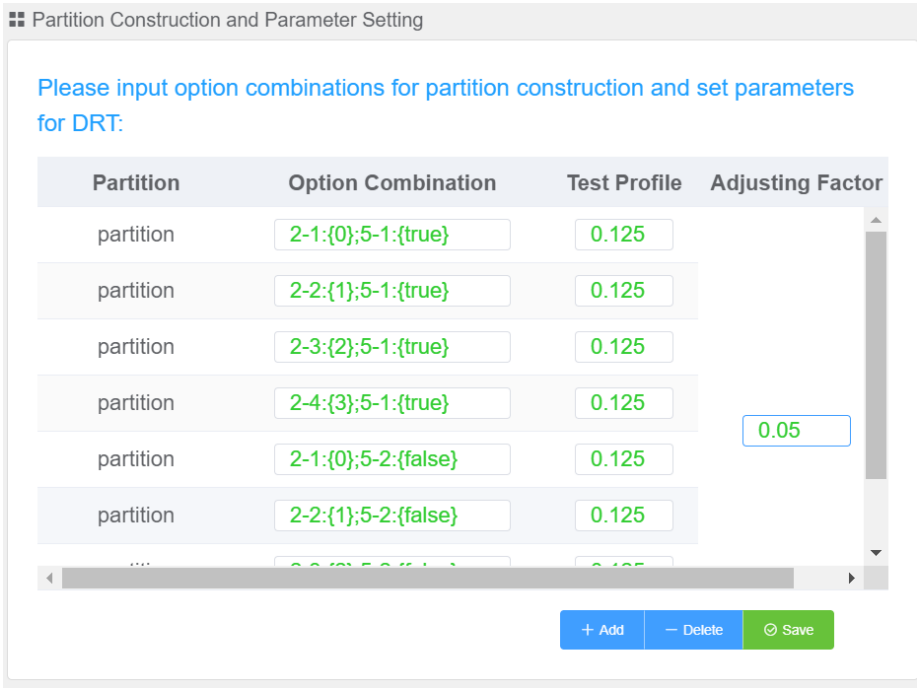


The user divides each parameter into disjoint options, and describes them according to predefined rules that are introduced in *Guidance* page (as shown in the following figure). Once the "Save" button is clicked, parameters and corresponding options are sent to some Restful micro-service.



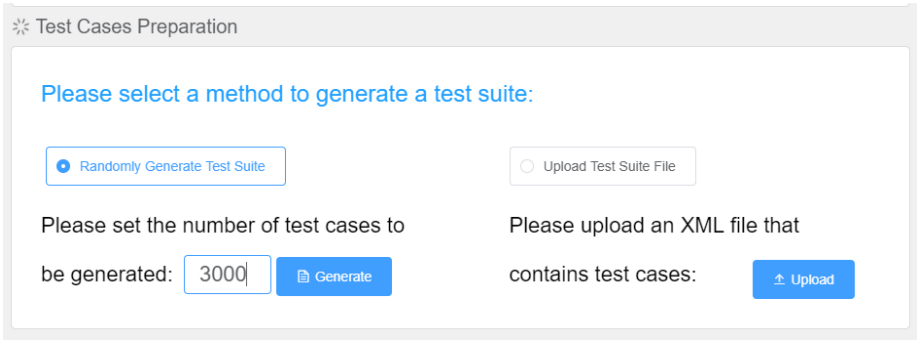
Step 2: Partition construction and parameter setting

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



Step 3: Test case preparation

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



Step 4: Test case execution

After performing all the steps above, all necessary preparation for testing are complete. The user next clicks the *Start* button, then the table in the middle of the page shows the information for the testing. If test cases are generated randomly, then the user needs to decide whether or not the last test case detected a fault. If the test case detected a fault, the user changes the value of the last row and last column in the table to "1". Then, the next test case is executed by clicking the "Next" button. The "Stop" button is responsible for sending a signal to the back-end logic, which means that the testing task is finished. The function of downloading the test report is also supported by

clicking the "Download" button.