Design and Implementation of Comprehensive Test Automation Method for API Adapter in C-Plane and U-Plane

Sho Kanemaru
NTT Network Service Systems Laboratories
NTT Corporation
Tokyo, Japan
sho.kanemaru.pw@hco.ntt.co.jp

Tomoki Ikegaya
NTT Network Service Systems Laboratories
NTT Corporation
Tokyo, Japan
tomoki.ikegaya.ya@hco.ntt.co.jp

Kensuke Takahashi
NTT Network Service Systems Laboratories
NTT Corporation
Tokyo, Japan
kensuke.takahashi.gm@hco.ntt.co.jp

Tsuyoshi Toyoshima
NTT Network Service Systems Laboratories
NTT Corporation
Tokyo, Japan
tsuyoshi.toyoshima.dk@hco.ntt.co.jp

Abstract— **The Business-to-Business-to-X (B2B2X) model has increased the importance of orchestrators that build and operate services that consists of various wholesale services. To develop RESTful API-based services, service providers need to catch up on the specifications of new wholesale services and specification changes of existing services quickly and inexpensively. Therefore, it is important to develop software called an API adapter that absorbs API differences from various services quickly and inexpensively. In this paper, we propose a method for comprehensively automating testing of not only C-Plane signals such as service provisioning/change/abolishment but also U-Plane signals such as communication between user devices and servers. We implemented our proposal using open source software and found that it improved test efficiency in an actual software development project.**

Keywords—API Adapter, Test automation, Orchestrator

I. Introduction

The spread of the Business-to-Business-to-X (B2B2X) model is increasing the importance of integrated management software called an "orchestrator", which supports the provisioning and operation of multiple wholesale services provided as representational state transfer application programming interfaces (RESTful APIs; hereinafter simply called APIs). However, the specifications of published APIs vary for each service and are frequently modified, so service providers are required to catch up on the specifications of new wholesale services and specification changes of existing services quickly and inexpensively. Thus, service providers need to develop API adapters, which convert the various wholesale APIs (Southbound APIs in Fig. 1) into unified APIs (Internal APIs in Fig. 1) and absorb the API differences in various wholesale services.

In the service development that combines the wholesale partner service, the API adapter needs to be able to be added/changed quickly. Therefore, technology is needed to develop an API adapter efficiently [1]. In the test process in network service development, it is necessary to test that not only the control signals (C-Plane signals) such as the provisioning/changing/abolishing operations of the service but also the data communication signals (U-Plane signals) are correctly exchanged in order to confirm that the C-Plane signals are correctly reflected in network equipment and the customer terminals.

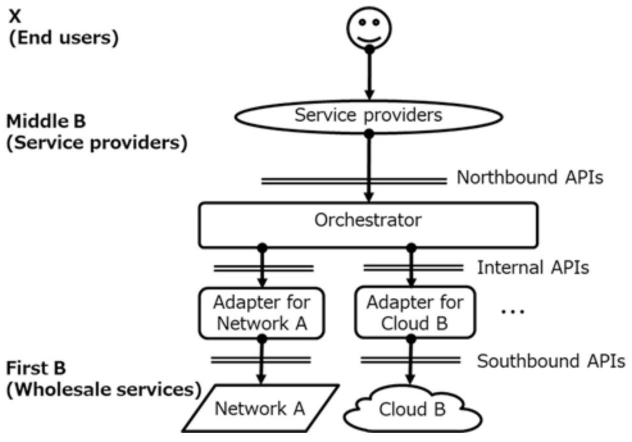
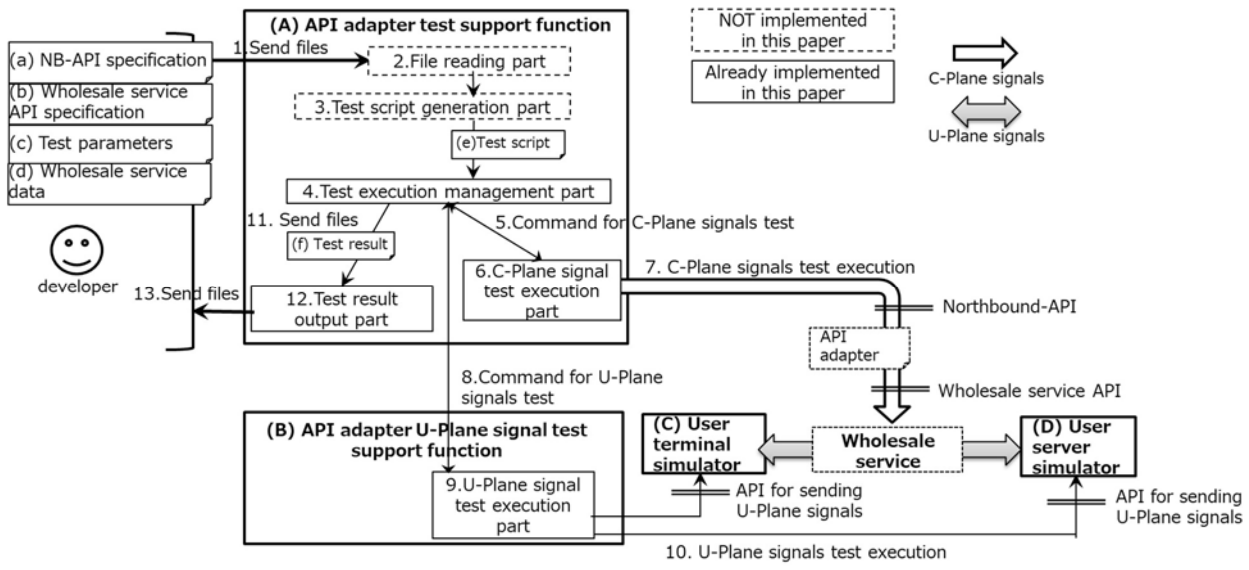


Fig. 1 Orchestrator and API adapters

As a method to deliver services in a short time, agile development practices are frequently used. When developing network services in agile, as the "sprint" (a development unit in agile) progresses, the number of items that need to be tested per sprint tends to increase. This is because test items in sprint N include not only items developed in the current sprint but also regression testing items developed in the previous sprint (i.e., sprint N-1). However, the test period is constant in each sprint. Therefore, development cannot proceed on schedule unless the test efficiency of the C-Plane and U-Plane is improved. Techniques have been proposed to simplify the design process and the implementation process of the API adapter [1] and a part of the testing process [2], but no technique has been proposed to comprehensively automate the testing of the C-Plane and U-Plane signals.



(A) API adapter test support function
(a) NB-API specification
(b) Wholesale service API specification
(c) Test parameters
(d) Wholesale service data
1.Send files
2.File reading part
3.Test script generation part
(e)Test script
4.Test execution management part
11. Send files
(f) Test result
5.Command for C-Plane signals test
6.C-Plane signal test execution part
12.Test result output part
developer
13.Send files
7. C-Plane signals test execution
Northbound-API
API adapter
Wholesale service API
8.Command for U-Plane signals test
(B) API adapter U-Plane signal test support function
9.U-Plane signal test execution part
(C) User terminal simulator
Wholesale service
(D) User server simulator
API for sending U-Plane signals
API for sending U-Plane signals
10. U-Plane signals test execution
NOT implemented in this paper
Already implemented in this paper
C-Plane signals
U-Plane signals

Fig. 2. Procedure of proposed method

In this paper, we propose a method to comprehensively automate testing of C-Plane and U-Plane signals in the testing process of an API adapter, which is a key element in the development of network services in the B2B2X model. This proposal consists of (1) a mechanism that can carry out a test combining C-Plane and U-Plane signals by combining the API adapter test support function, API adapter U-Plane signal test support function, user terminal simulation device, and user server simulation device and (2) a test scenario execution management technique that executes APIs of C-Plane and U-Plane signals on the basis of the test scenario and analyzes the test results.

II. Related Work

A. API specification method

Numerous methods to test RESTful API-based software have been proposed. As software development and testing are often done by teams, API specifications should be shared by teams, and a format that defines API specifications is useful. Swagger [3] is a standard format defined by the OpenAPI Initiative [4] for describing the interface of RESTful API, and its corresponding document is called Swagger Spec. Swagger Spec is expressed in a file format such as JSON or YAML, and API specifications can be discussed while visualizing documents using editing tools such as Swagger Editor.

B. Test Driver/Stub generation method

As an API adapter converts the Southbound APIs into the Internal APIs, the API adapter alone cannot be used for testing, and a stub or driver that simulates the corresponding system is required. The driver sends an API request to the API adapter and receives an API response (i.e., it simulates the orchestrator in Fig. 1). The stub receives the API request from the API adapter and sends the API response (i.e., it simulates the wholesale services in Fig. 1). Cucumber [5] is an open source software (OSS) for automating the testing of RESTful APIs and acts as a driver by executing a test scenario defined by a grammar called Gherkin. In addition, since Cucumber has a function to judge whether the response returned from the API adapter is as expected or not, the work of manually verifying the API response can be coded, and it is useful for improving test efficiency. Postman [6] is an OSS that enables HTTP clients and, like Cucumber, provides a mechanism for automating the tests of drivers and APIs. There is also a tool called Newman [7], which is a CLI version of Postman. As stub generation technologies, there is Swagger-Codegen [8]. Swagger-Codegen is a tool that automatically generates drivers and stubs on the basis of the API specification defined in the Swagger Spec. These driver/stub generation technologies enable software necessary for testing API adapters to be prepared quickly.

C. Test automation technology

In the related research [2], the test of C-Plane in the API adapter test process was automated. On the other hand, the test automation of the U-Plane is not mentioned, so it is not possible to verify whether the content of the C-Plane signal is correct even if the C-Plane signal is communicated. For example, if the API response of the C-Plane signal is normal, whether the parameters stored in the API request are neither excessive nor deficient can be verified, but whether the contents of the parameters (e.g., IP address values) are actually correct cannot be verified. Procedures such as login to various network devices and communication confirmation commands of U-Plane signals are necessary to check the accuracy of parameters, and this confirmation takes time.

III. Proposed method

A. Components

The proposed method consists of *(A) an API adapter test support function*, *(B) an API adapter U-Plane signal test support function*, *(C) a user terminal simulator,* and *(D) a user server simulator* (see Fig. 2). (A) The API adapter test support function is responsible for managing the execution of the whole test scenario and executing the test of the C-Plane. It generates test scripts from (a) the Northbound API specification, (b) wholesale service API specification, (c) test parameters, and (d) wholesale service data. After generating the test scripts, (A) runs the tests of C-Plane signals on the basis of the test scripts and outputs the test results. (B) The API adapter U-Plane signal test support function tests the U-Plane signal on the basis of a test script generated by (A) the API adapter test support function. Specifically, the

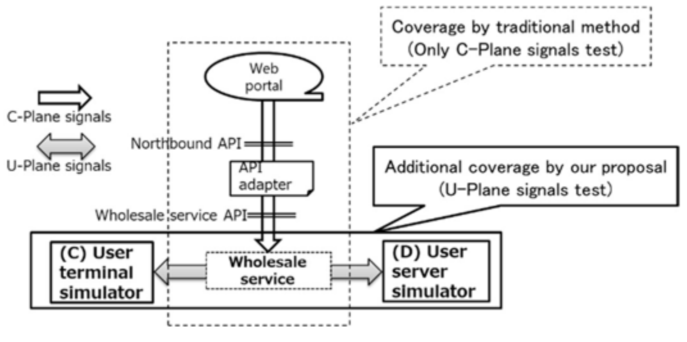

Coverage by traditional method (Only C-Plane signals test)
C-Plane signals
U-Plane signals
Web portal
Northbound API
API adapter
Wholesale service API
Additional coverage by our proposal (U-Plane signals test)
(C) User terminal simulator
Wholesale service
(D) User server simulator

Fig. 3. Test coverage expansion by our proposal

communication of the U-Plane signal can be verified by executing the API of the device group ((C) the user terminal simulator, and (D) the user server simulator). (C) The user terminal simulator and (D) the user server simulator are used to verify the communication of the U-Plane signal. For example, after an operation that activates the communication between the user terminal and the user server by (A) the API adapter test support function, (C) and (D) verify the communication of U-Plane by using commands such as ping and curl from (C) to (D) or vice versa. As a result, it is possible to verify whether the contents of the API request is correct and whether the service provisioning/changing/abolishing operations are properly reflected in the network equipment group and customer terminals (as described in Section I).

B. *Test script execution management technology*

The test script generated by (A) the API adapter test support function is passed to the test execution management part” and executed (step 4 in Fig. 2). The test script is run by the C-Plane signal test execution part and the U-Plane signal test execution part. The test execution management part controls the execution of the test script. Specifically, it instructs the C-Plane signal test execution part to test the C-Plane signals (e.g., sending an provisioning operation for a service). When the C-Plane signal test is completed (steps 5 to 7 in Fig. 2), the U-Plane signal test (e.g., curl command from (C) the user terminal simulator to (D) the user server simulator via wholesale service) is instructed to the U-Plane signal test execution part (steps 8 to 10 in Fig. 2), and the result is output (steps 11 to 12 in Fig. 2). If the C-Plane signal test fails, the test result is output without performing the U-Plane signal test. The test script not only sends API requests and receives API responses but also verifies whether the contents of the received API responses are the same as expected in the test scenario. For example, the test script checks if the response code is 200 OK, if the response body is in JSON format, and if the JSON format array contains the expected value in the Nth entry. The test script removes the manual review from the testing process and enables test scripts to be created that are highly compatible with continuous integration (CI) such as "Automatically tests once a day and sends the results via chat."

C. *Benefit of our proposal*

The proposed method can comprehensively test the C-Plane and U-Plane signals. As a result, it can verify that not only signals such as service provisioning/changing/abolishing operations are correctly exchanged but also that the parameters in the API request are correct and are precisely reflected in network equipment groups, customer terminals, etc., thus expanding the automation range of the API adapter test (Fig. 3).

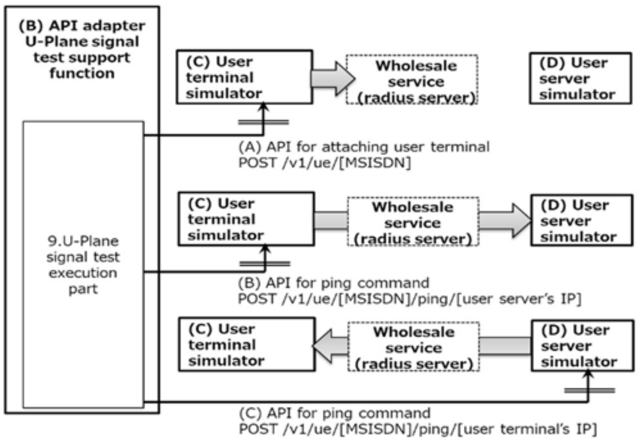

(B) API adapter U-Plane signal test support function
9.U-Plane signal test execution part
(C) User terminal simulator
Wholesale service (radius server)
(D) User server simulator
(A) API for attaching user terminal
POST /v1/ue/[MSISDN]
(B) API for ping command
POST /v1/ue/[MSISDN]/ping/[user server's IP]
(C) API for ping command
POST /v1/ue/[MSISDN]/ping/[user terminal's IP]

Fig. 4. APIs provided by (C) user terminal simulator and (D) user server simulator

IV. Implementation

A. *Implementation scope*

This section describes the implementation of step 4 and subsequent steps in Fig. 2. Other steps (i.e., steps 1 to 3 in Fig. 2) will be implemented in the future.

B. *Test scenario example*

In the example test scenario, a network service using a SIM card is activated, the SIM card configuration is changed, and the SIM card is terminated.

1. *Service activation order input (C-Plane signal)*
2. *Verify SIM information is registered with authentication server (Verifying the Results of Scenario 1)*
3. *User Terminal Attach Process (U-Plane signal)*
4. *Verify ping command from User Terminal to User Server **succeeds** (U-Plane signal)*
5. *Service change order to suspend SIM card usage (C-Plane signal)*
6. *Verify the SIM Information on the Auth.Server has changed (Verifying the Results of Scenario 5)*
7. *Verify ping command from User Terminal to User Server **fails** (U-Plane signal)*
8. *Service termination order (C-Plane signal)*
9. *Verify the SIM Information for the Auth. Server has been deleted (Verifying the Results of Scenario 8.)*
10. *Verify ping command from User Terminal to User Server **fails** (U-Plane signal)*

C. *Example of equipment configuration*

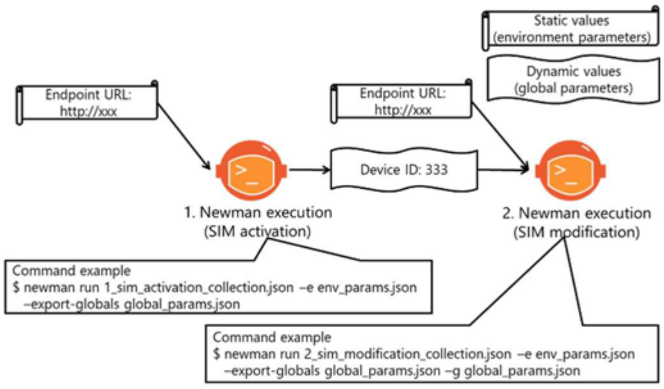
We used a Newman to implement the test execution management part, the C-Plane signal test execution part, and the U-Plane signal test execution part. To implement a user terminal simulator, a server having APIs shown in Fig. 4 (A)(B) is used. To implement an user server, a server having the API shown in Fig. 4 (C) is used. We use a Mobile Subscriber ISDN Number (MSIDSN) as an user terminal identifier. To implement Wholesale Service Equipment, a radius server is used as the authentication function of the SIM card.

D. *Example procedure*

First, input files prepared by the developer are read by the test scenario generation part. (a) The Northbound API specification and (b) Wholesale Services API specification are JSON files in Swagger Spec format. (c) Test parameters and (d) Wholesale Services data are CSV format files. Second, the test script generation part largely executes two kinds of processing. One creates the Postman collection (JSON files that Newman runs) as a test script from input files (a) and (b), and the other generates parameter patterns. Next, Newman runs the Postman collection created by the test script generation part. In addition to sending API requests and receiving API responses, Newman can run the scripts to verify if the received API response is as expected. In addition to verifying the Northbound API response, Newman runs the Wholesale Service Equipment (Radius Server) API to make sure that the Northbound API execution is reflected on the Wholesale Service Equipment side as well (e.g., scenarios 2, 6, and 9). After Newman executes scenarios 1 to 2, the U-Plane signal is tested. Newman executes the API of the user terminal simulator and performs terminal attachment processing (scenario 3) and ping communication confirmation processing (scenario 4). Success of processing in scenario 3 and 4 means that the contents of C-Plane signal (e.g., MSISDN) is correct and precisely reflected in the wholesale service equipment. If the processing succeeds, scenario 5 is executed, and if it fails, the test is interrupted and the test result is output. Then, operation of service change and U-Plane confirmation (scenarios 5 to 7) and operation of service abolition and U-Plane confirmation (scenarios 8 to 10) are executed, and the test result report is output after both operations are completed. As an example of a test result report, a Newman plug-in "Newman-reporter-htmlextra" [9] is used.

E. *Ideas in the test execution management part*

We faced a challenge when implementing the test execution management part with Newman. The test scenario described in Section IV-B consists of several test scripts (Postman collections) and state transitions. Therefore, the values used in the previous test script must be used in the next test script, and there must be a mechanism for passing values between scripts. There are two kinds of parameters in the test scripts: static and dynamic. Static parameters are values that do not change during test execution and are determined before the test execution (e.g., the API endpoint URL). Dynamic parameters are values generated during test execution and are unknown before the test execution. For example, the ID issued to the device by the system when the SIM card is activated. To manipulate these parameters, we have used two kinds of parameters provided by Postman: environment values and global values. Environment values are used for defining the parameters that depend on the environment (e.g., IP address and endpoint). We stored static values in the environment values. Environment values can be managed in the JSON file and the JSON file can be loaded with a "-e" option when we execute the Newman command (step 1 in Fig. 5). Global values are used for passing the parameters between multiple Postman collections. We stored dynamic values in the global values. Global values can also be managed in the JSON file, and the JSON file can be loaded with a "-g" option and can be written with a "--export-globals" option when we execute the Newman command (step 2 in Fig. 5). By combining multiple test scripts using environment and global values, we have successfully implemented a test scenario that includes state transitions.



```
$ newman run 1_sim_activation_collection.json –e env_params.json
  –export-globals global_params.json
```

```
$ newman run 2_sim_modification_collection.json –e env_params.json
  –export-globals global_params.json –g global_params.json
```

Fig. 5. Pass static parameters and dynamic parameters between multiple Postman collections

V. Evaluation

We introduced the implementation example described in Section IV into the agile development of a network service. We found that the contents of the C-Plane signal were correct by verifying not only the operation of the API adapter by C-Plane signal test but also the communication of the U-Plane signal after the C-Plane signal was input. Therefore, the proposed method solved the problems of the existing methods described in Section II-D and achieved the utility described in Section III-C. In addition, the number of tests that could be run in the same period was more than 25 times that before the proposed method was introduced (from 200 to 5223 items). In the test process without our proposed method, the curl command was used for the API execution, and the developer needed to manually verify the API response after the curl command execution. By introducing the proposed method, the manual review process, which is a bottleneck, was eliminated, and test efficiency was significantly improved.

VI. Conclusion

In this paper, we proposed a method to comprehensively test not only C-Plane signals but also U-Plane signals in the testing process of an API adapter, which is an important element of network service development in the B2B2X model. This method expands the testing range and enhances the efficiency of the API adapter. This paper described each equipment and automation system for automating tests and described the implementation examples using OSS on the basis of test scenario examples. Evaluation results showed the proposed system is feasible and effective. As a future challenge, we will implement the test script generation part as shown in Fig. 2.

References

[1] N. Take, "Method to Simplify API Adapter Development using GUI", Proc. of the 2018 IEICE Society Conference, B-14-13, Sep. 2018

[2] Sho Kanemaru, "A study on automation of testing API adapters", Proc. of the 2019 IEICE Society Conference, BS-4-12, Mar. 2019

[3] Swagger, https://swagger.io/

[4] Open API Specification, https://swagger.io/specification/

[5] Cucumber, https://cucumber.io/

[6] Postman, https://www.getpostman.com/

[7] Newman, https://learning.getpostman.com/docs/postman/collection-runs/command-line-integration-with-newman/

[8] Swagger-Codegen, https://github.com/swagger-api/swagger-codegen

[9] Newman-reporter-htmlextra, https://www.npmjs.com/package/newman-reporter-htmlextra