Interface Blending/Diagnosis Technology for Automatically Generating an Interface Adapter

Yu Miyoshi[†] *and Tatsuyuki Kimura*

Abstract

Operating a network while new services are being continuously introduced is a complex and difficult task. This article introduces interface blending/diagnosis technology developed in NTT Laboratories. It consists of 1) a blending method that automatically generates network management system (NMS) interface adapters for different vendors from commands and responses transmitted and received between an NMS and a network element (NE) and 2) a diagnosis method that examines blended interfaces with an actual NE and verifies the results. This technology alleviates the effort of network operations by automating the development of functions that operators use to set up NEs and the NMS.

1. Importance of network operations

An IP (Internet protocol) network consists of various types of network elements (NEs) such as routers, switches, and servers. With the recent growth of networks and services, network operators and service system developers have had to handle an increasing number and variety of NEs, and their everyday work has become very complicated. For IP-based network services, the technology is changing especially quickly, with new services appearing all the time while the life cycles of individual services tend to become shorter and shorter. As a result, operators and developers are being required to make ever-increasing efforts.

Without any doubt, the importance of the roles played by the operations support system (OpS) and network management system (NMS) in providing network services will continue to increase in the future. In particular, it is essential to support the establishment of large-scale networks, since any delay in introducing a new service is likely to have an adverse effect on business profits. Moreover, OpSs and NMSs enable us to provide services without customers feeling any stress from a slow response by the operator. They enable a quick response to a customer's service order.

† NTT Network Service Systems Laboratories
Musashino-shi, 180-8585 Japan
E-mail: miyoshi.yu@lab.ntt.co.jp

2. Operations issues

2.1 Service operations issues

In a large-scale network, even modifying the network configurations can involve a lot of work. IP networks are configured from various types of NEs developed by multiple vendors, even for basic components such as routers. The network operator and an NMS must be able to handle all these different equipment types. The operator's job includes various tasks such as setting diverse service parameters, booting up equipment, updating operating systems, and collecting error logs. Although these might seem to be simple tasks at first glance, they require different knowledge and techniques about the interface specifications and complex operating procedures of each NE, and there is no excuse for making mistakes while services are being provided via the network. A number of tools that support NE setting operations have appeared recently. Device Authority Suite [1] has functions that record the commands sent by operators on a command line interface (CLI) and the responses obtained from the NE. Since these exchanges are recorded in the format of a scripting language, the operator can easily adapt them to individual circumstances if the differences are only minor, which allows them to be re-used more often and supports the automated processing of similar settings. However, operators must still perform separate setup operations for NEs provided by different vendors because of their different interface specifications. Moreover, they must still address the problems of how to adapt quickly to the introduction of new NEs or modified interface specifications and how to continue operations smoothly.

2.2 System interface development issues

An NMS that supports the running of a network monitors the entire network including multiple types of equipment. This includes monitoring not only each individual item of equipment, but also the traffic path, and the impact of equipment failures. The commercial products described in references [2] and [3] are well known in this category. At NTT Laboratories, we have also been developing technology for monitoring networks over multiple layers [4] and ascertaining the paths used by specific services and the detour paths [5].

SNMP (simple network management protocol) is a well-known protocol for gathering information from NEs. Since it is a standard protocol in many types of equipment, it is widely used. However, it is not possible to monitor services that use new technology that has yet to be fully implemented in the management information base (MIB) whose contents are acquired via SNMP or to use parameter setting in the equipment because of NE implementation and security problems. Therefore, the use of a CLI to enter settings and acquire detailed information is currently the most popular approach. In this case, the NMS is equipped with an interface that transmits commands to the equipment, but even for settings with identical meanings, it is necessary to provide different processing statements for NEs from different vendors. These processing functions are called "interface adapters" and they are usually developed individually (**Fig. 1**).

The interface adapter is made using a script language that can easily be augmented as opposed to the NMS itself, which is implemented in a programming language such as Java. This is because it is necessary to use interactive processing whereby the response to a transmitted command is used to select the subsequent processing and because processing statements are expected to be frequently updated to keep up with ongoing changes to NE specifications. Well-known script languages that can be used for this purpose include Perl, JavaScript, Python, and Expect.

However, the rapid introduction of developed interface adapters has a problem. For example, in situations where a new NE is released, it is preferable to match up the specifications by developing an interface adapter at the same time as the NMS (**Fig. 2**). But matching up is a difficult task when the NE and the NMS interface adapter have been developed in parallel. Even if it is assumed that the specifications can be matched up, it is still necessary to perform verification trials on the actual equipment. If a connection failure occurs during these trials, even more time must be spent on the specification design and interface adapter coding.

With regard to the methods used to develop system functions, such as interface adapters where results are needed quickly, attempts are being made to automate their creation by applying MDA (model driven architecture), which is being studied by the OMG (object management group) [6]. However, since this method requires that the UML (unified modeling language) specifications of the functions are declared as models, it is necessary to support the creation of models. Furthermore, instead of creating functions from

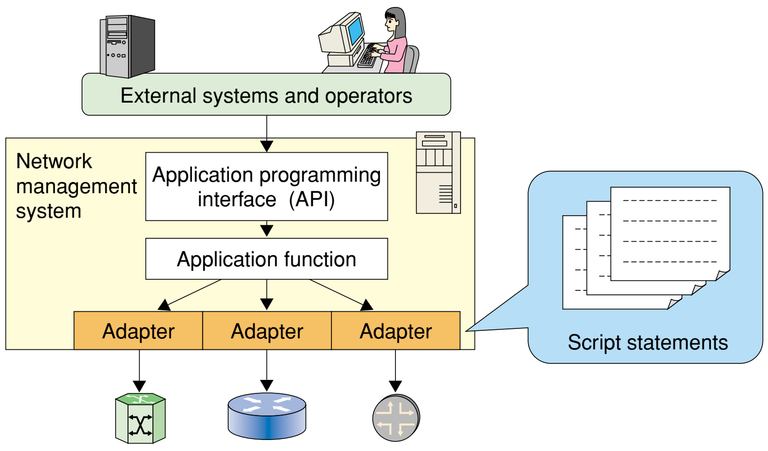

External systems and operators
Network management system
Application programming interface (API)
Application function
Adapter
Adapter
Adapter
Script statements

Fig. 1. Position of interface adapters.

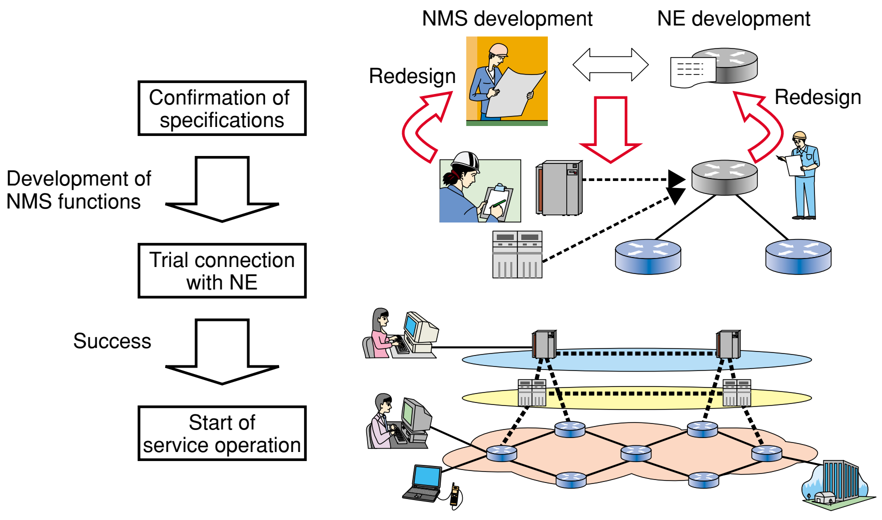

Confirmation of specifications
Development of NMS functions
Trial connection with NE
Success
Start of service operation
NMS development
NE development
Redesign
Redesign

Fig. 2. NMS and NE development process.

scratch with models, it may be necessary to consider enhancing efficiency by re-using existing adapters.

3. Interface blending/diagnosis technology

The interface blending/diagnosis technology automates part of the work involved in operator settings input and interface adapter development with the aim of reducing the workload (especially in the complexity of dealing with settings for different vendors and with frequent specification changes), thereby contributing to the rapid introduction and smooth operation of network services. If the specification design and development of setting functions and adapters can be automated, then work can be performed directly on NE and NMS connection trials and operations, and it should be possible to greatly reduce the effort on operators and developers. This technology will also be able to check problems such as misconfigurations caused by human error. Specifically, we are considering providing a system with “blending” and “diagnosis” methods that automatically bridge the differences between the interfaces of different vendors when performing similar processes (**Fig. 3**).

Our system extracts information such as command strings from the operator’s previous setting history or from an interface adapter that has already been installed and stores it in a database after adding attributes such as processing sequences. After that, this existing information is combined to create the commands to send to another target NE. It is possible to operate in this way because many commands include words with similar meanings even though the NEs come from different vendors and have different specifications. Our system utilizes these characteristics. Moreover, using knowledge based on the interface specifications defined by each vendor makes it more likely that commands and interface adapters can be created automatically. This method is called “blending”.

However, the command strings and interface adapter script statements generated by blending are not used without alteration. If a large amount of accurate knowledge is available then the probability of successfully creating an accurate adapter also increases, but this cannot be guaranteed and it may be inadvisable to use these statements without any verification. Therefore, the interface diagnosis system is equipped with methods that use the adapter generated by the blending method to perform connection tests on the NE in question. We call this method “diagnosis” because it discovers the correct solution by connecting the system to the actual equipment. The adapter is automatically configured based on the responses obtained by this diagnosis. We are considering generating the adapter in script form (we are currently using Expect), so that it can be easily modified by operators and developers. Furthermore, it can be run as part of an NMS, or as a standalone application on an external system.

4. Detailed design

This section presents details of the design that we implemented in a system for an interface blending and diagnosis method (**Fig. 4**). There are four components, which are described in sections 4.1. to 4.4.

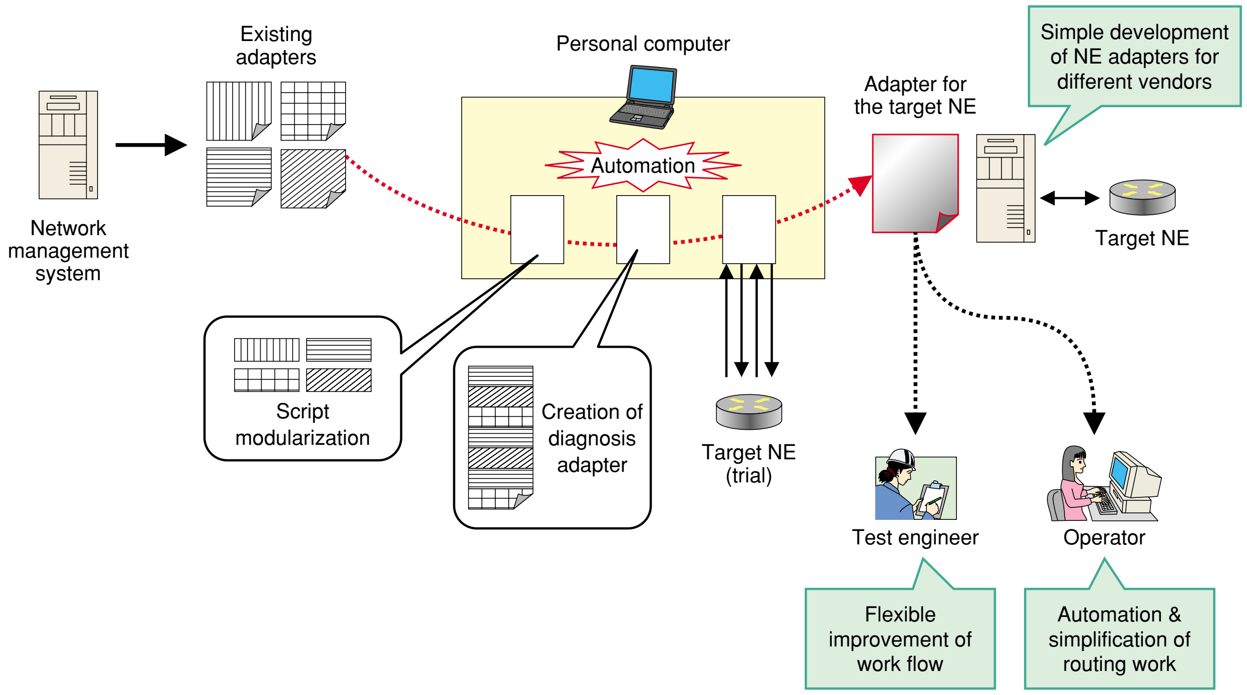

Existing adapters
Personal computer
Simple development of NE adapters for different vendors
Adapter for the target NE
Automation
Network management system
Target NE
Script modularization
Creation of diagnosis adapter
Target NE (trial)
Test engineer
Operator
Flexible improvement of work flow
Automation & simplification of routing work

Fig. 3. Generation of adapters by interface blending and diagnosis technology.

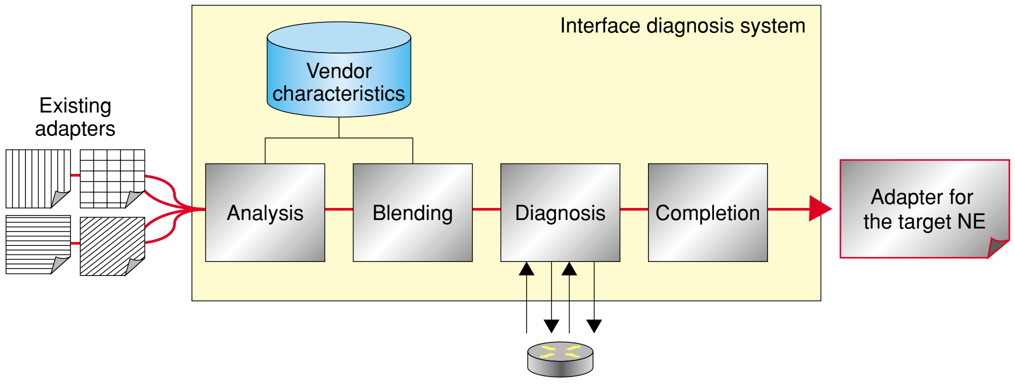


Fig. 4. Components of interface diagnosis system.

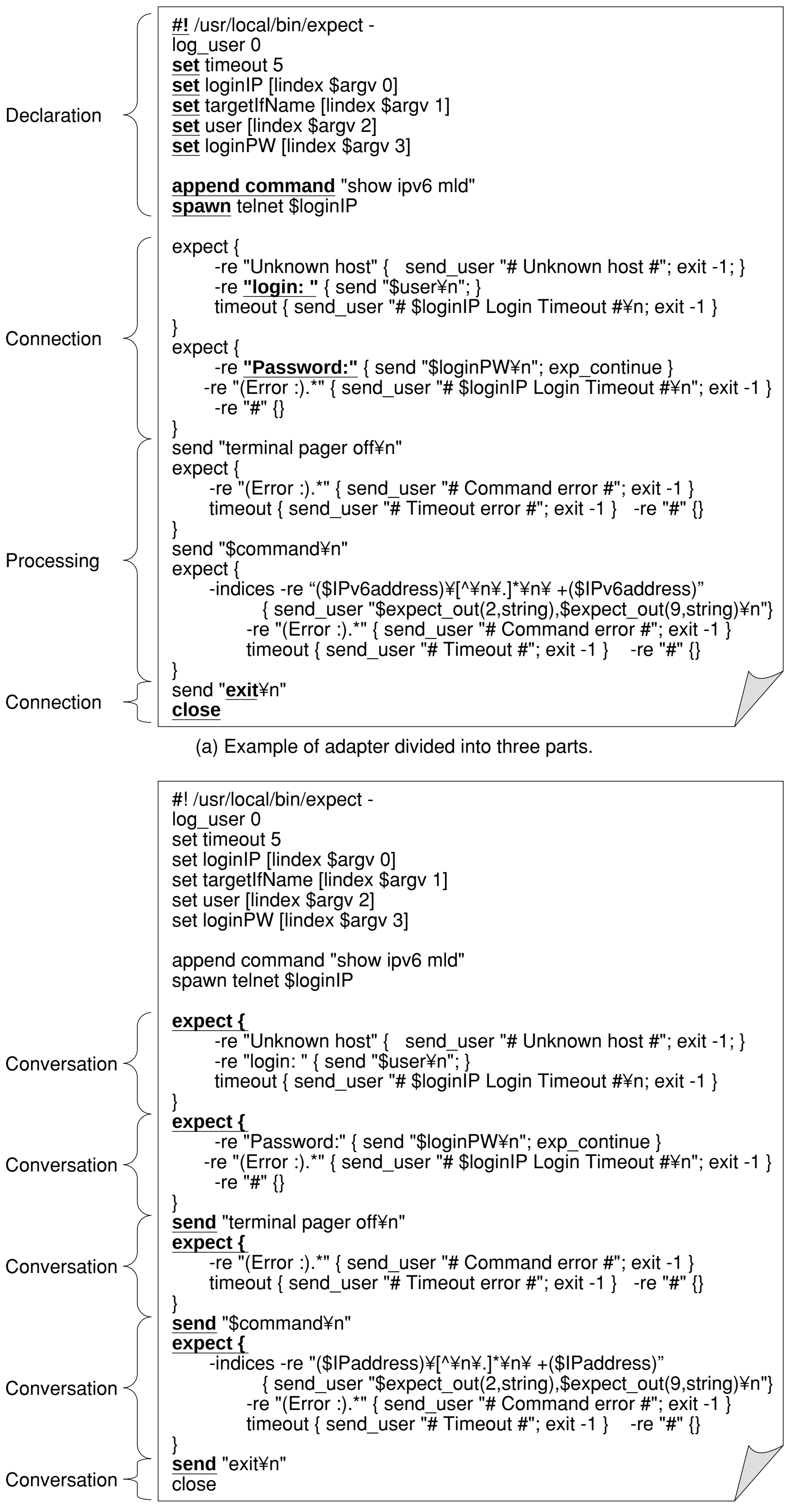
4.1 Analysis component

First, we describe the analysis component of existing adapters entered by a developer. We found that all adaptors have a structure for connecting to a target NE by telnet, transmitting commands, obtaining data from responses, and disconnecting from it. They also have a structure that can be divided into the following three parts (**Fig. 5(a)**).

1) The declaration describes variable declarations and argument definitions.
2) The connection describes the process of connecting with an NE such as connecting, disconnecting, and entering a username or password.
3) The processing describes interactive applications such as command transmission and response handling.

Moreover, we found that the connection and processing parts can be divided more minutely and that the description of a command transmission can be paired up with the response. Therefore, we call the pair of descriptions (command and response) a "conversation" and define a "conversation" as the smallest unit that expresses an NMS adapter function (**Fig. 5(b)**).

The analysis component divides an input adapter into three parts, which are subdivided into conversa-



```
#! /usr/local/bin/expect -
log_user 0
set timeout 5
set loginIP [lindex $argv 0]
set targetIfName [lindex $argv 1]
set user [lindex $argv 2]
set loginPW [lindex $argv 3]

append command "show ipv6 mld"
spawn telnet $loginIP

expect {
        -re "Unknown host" {   send_user "# Unknown host #"; exit -1; }
        -re "login: " { send "$user¥n"; }
        timeout { send_user "# $loginIP Login Timeout #¥n; exit -1 }
}
expect {
        -re "Password:" { send "$loginPW¥n"; exp_continue }
      -re "(Error :).*" { send_user "# $loginIP Login Timeout #¥n"; exit -1 }
        -re "#" {}
}
send "terminal pager off¥n"
expect {
       -re "(Error :).*" { send_user "# Command error #"; exit -1 }
       timeout { send_user "# Timeout error #"; exit -1 }   -re "#" {}
}
send "$command¥n"
expect {
       -indices -re "($IPv6address)¥[^¥n¥.]*¥n¥ +($IPv6address)"
                 { send_user "$expect_out(2,string),$expect_out(9,string)¥n"}
             -re "(Error :).*" { send_user "# Command error #"; exit -1 }
             timeout { send_user "# Timeout #"; exit -1 }    -re "#" {}
}
send "exit¥n"
close
```

(a) Example of adapter divided into three parts.

```
#! /usr/local/bin/expect -
log_user 0
set timeout 5
set loginIP [lindex $argv 0]
set targetIfName [lindex $argv 1]
set user [lindex $argv 2]
set loginPW [lindex $argv 3]

append command "show ipv6 mld"
spawn telnet $loginIP

expect {
        -re "Unknown host" {   send_user "# Unknown host #"; exit -1; }
        -re "login: " { send "$user¥n"; }
        timeout { send_user "# $loginIP Login Timeout #¥n; exit -1 }
}
expect {
        -re "Password:" { send "$loginPW¥n"; exp_continue }
      -re "(Error :).*" { send_user "# $loginIP Login Timeout #¥n"; exit -1 }
        -re "#" {}
}
send "terminal pager off¥n"
expect {
       -re "(Error :).*" { send_user "# Command error #"; exit -1 }
       timeout { send_user "# Timeout error #"; exit -1 }   -re "#" {}
}
send "$command¥n"
expect {
       -indices -re "($IPaddress)¥[^¥n¥.]*¥n¥ +($IPaddress)"
                 { send_user "$expect_out(2,string),$expect_out(9,string)¥n"}
             -re "(Error :).*" { send_user "# Command error #"; exit -1 }
             timeout { send_user "# Timeout #"; exit -1 }    -re "#" {}
}
send "exit¥n"
close
```

(b) Example of adapter divided into “conversations”.

Fig. 5. Example of adapter description.

tions (**Fig. 6**). In addition, it stores dependency and sequence information between conversations as attributes.

4.2 Blending component

The blending component generates an adapter for testing a target NE (Fig. 6). In a survey of interface adapters, we found similar meanings in many command strings. The interface specifications are not compatible between different NEs, but NE interfaces are implemented considering operator usability and the words that the operator can use in specifications. Therefore, we assume a system can obtain responses that enable it to generate correct adapters for a target NE if it queries the NE with existing or similar commands of other vendors’ NEs. The blending component generates trial adapters by recalibrating the existing commands. We can use various methods to generate new trial commands, such as the following calibrations rules.

- Using just existing commands of other adapters
- Modifying existing commands. (e.g., changing *“show ip ospf interface”* ➔ *“show ip interface ospf”*)
- Replacing defined equivalent words. (e.g., changing *“show ipv6 ospf interface”* ➔ *“show inet6 ospf3 interface”*)

In addition, an adapter has description parts that do not depend on the meaning of NE-vendor-specific commands. For example, a connection part for log-in is the same as long as the vendor NE is the same. Therefore, we thought that the efficiency of generating an adapter automatically would increase if a system recycles patterns pertinent to the vendor NE. And we imple-

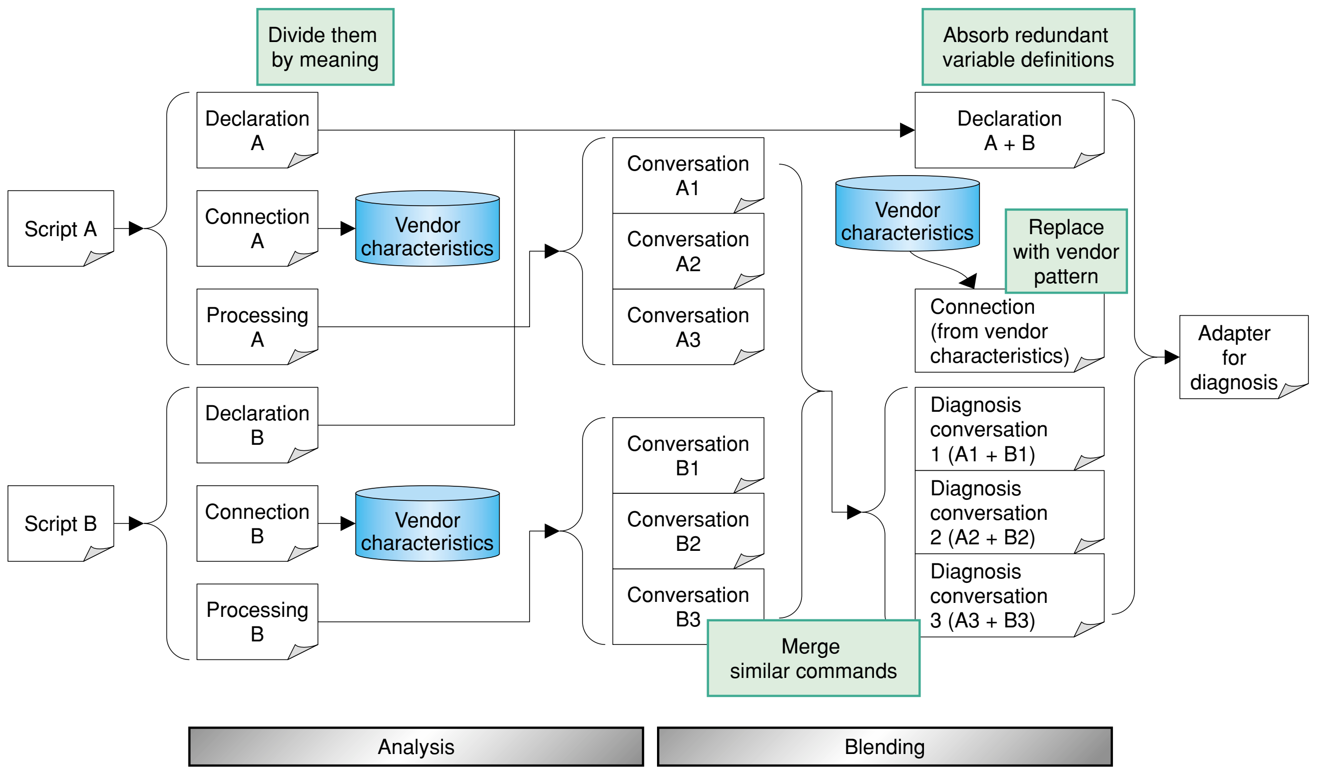


Fig. 6. Behavior of an analysis component and a blending component.

mented functions to store the vendor's reserved words in this component. If the target NE is supplied by a known vendor for our system, the vendor patterns are used prior to other blending methods in this component.

4.3 Diagnosis component

The behavior of the diagnosis component is shown in **Fig. 7**. Our system must search for effective information from replies of a target NE by transmitting many commands. To judge if this is effective, model answers are necessary. At the beginning of the process, the diagnosis component identifies the vendor of the target NE after the system has connected to it, and it identifies the vendor pattern automatically. The component acquires model answers from an existing NE automatically. It adds a conversation that records responses to an existing adapter. When the system connects with an existing NE, responses are recorded and defined as model answers. Next, the component searches for correct answers for a target NE by transmitting many commands generated by the blending component. The diagnosis component decides that an NMS can use the commands if the responses from the target NE match the model answers, and it incorporates the character string in an adapter for a target NE. Next, the diagnosis component transmits commands of conversations which are generated by the blending component as a test and gets real responses. It compares the responses with model answers, discovers a correct answer, and incorporates the test command into the adapter. In addition, this diagnosis work should be processed in a test network because backup and rollback processing are necessary.

4.4 Completion component

An adapter is completely generated once a completion component has added descriptions for error processing. However, we think it is necessary not only to establish technologies for increasing the success rate of automation, but also to strive to implement practical functions such as a system that repairs adapter errors easily through the intervention of an operator. Therefore, the system should have a graphical user interface (GUI) for an operator to add commands to the database if the diagnosis component could not be found. Our system will be made into a full solution by adding functions for correcting and complementing the adaptors easily.

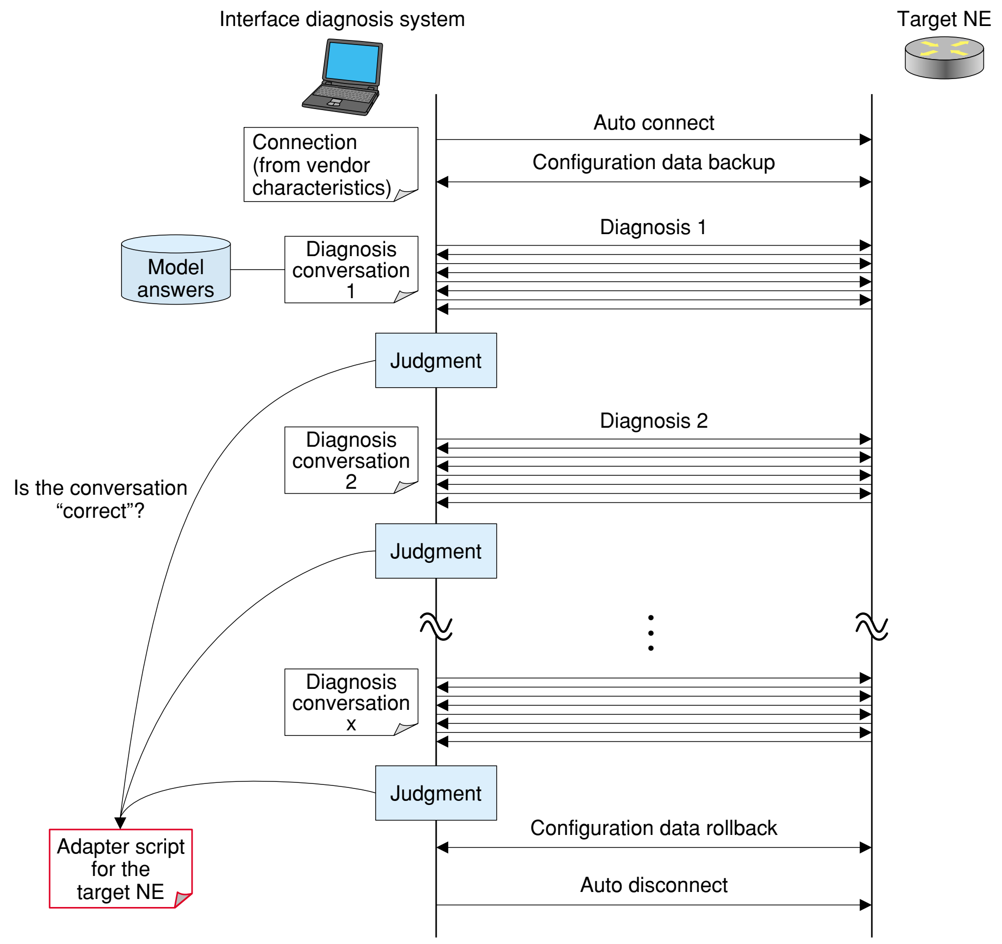

Interface diagnosis system
Target NE
Connection (from vendor characteristics)
Auto connect
Configuration data backup
Model answers
Diagnosis conversation 1
Diagnosis 1
Judgment
Diagnosis conversation 2
Diagnosis 2
Is the conversation "correct"?
Judgment
Diagnosis conversation x
Judgment
Configuration data rollback
Adapter script for the target NE
Auto disconnect

Fig. 7. Behavior of a diagnosis component.

5. Verification trials

5.1 Conditions

We performed verification trials on a prototype system (**Fig. 8**). In this trial, we assumed that an NMS developer was implementing an adaptor for a new target NE and that the NMS already had adapters for other NEs. We investigated how many correct answers our system could discover. For this investigation, we built a prototype of the above components. As existing adapters, we used adapters of a network management system that we had developed previously [5]. All adapters were described in Expect. They had commands for connecting to a target router, getting data, and disconnecting from the NE. We prepared three target NEs from different vendors and tested whether our system could generate three new adapters for each target NE. Therefore, we tested the automatic generation of nine adapters. It follows that the system did not know the correct commands of adapters of the target NEs. We prepared active NEs in a test network and the system queried them.

5.2 Results

First, we tested the diagnosis method using only the above-mentioned three calibration rules (4.3) implemented in the blending component. We were able to find 53% of the correct commands. This shows that the majority of commands resemble each other in the CLI of different vendor NEs. A developer may halve the effort of implementing a new adapter if our system can get 53% of the answers correct. Afterwards, we tried implementing new rules to automatically generate the commands that we were not able to confirm. As a result of adding new rules and having tested the diagnosis method again, our blending method was able to generate commands including those used for all 9 adapters and 44 conversations. New rules have general versatility, and they are useful in other scenarios, too. These results show that our system can help developers of this NMS sufficiently during the introduction of a new NE. On the other hand, it is very

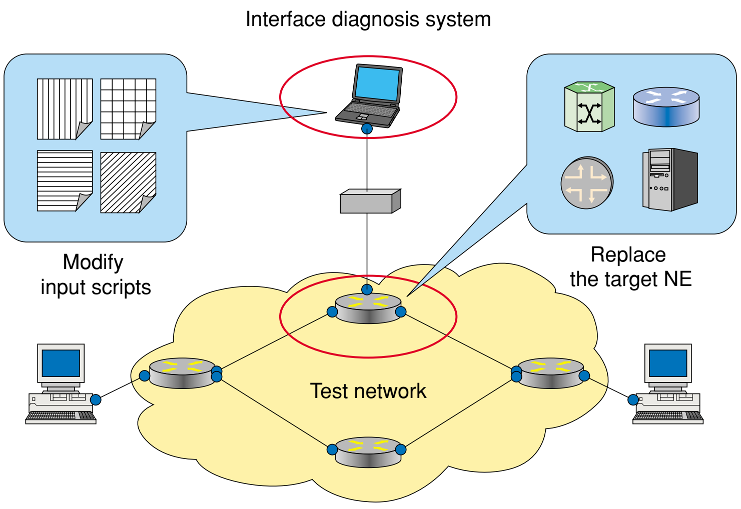


Fig. 8. Verification trials.

important for us that our system had a GUI function that enabled operators to easily add a new rule and start a trial immediately.

6. Conclusion

We described interface blending/diagnosis technology for automatically generating an interface adapter and explained the component design of an interface diagnosis system. We tested a prototype system to confirm the feasibility of our technology. In this trial, we were able to obtain effective responses for all conversations that we wanted to generate. Therefore, we think that we have proven the possibility of developing interface adapters. Moreover, we found command setting patterns that the system can use for NEs of other vendors.

References

[1] "Device Authority Suite Overview," http://www.alterpoint.com/products/
[2] "HP OpenView - Computer and Network Management," http://www.managementsoftware.hp.com/
[3] "JP1 Version 7i," http://www.hitachi.co.jp/Prod/comp/soft1/jp1/
[4] H. Yaginuma, N. Hatakeyama, T. Kimura, and Y. Otsuka, "Study of a risk analysis method based on multilayer network resource information," FIT2003 M-038, 2003.
[5] Y. Miyoshi, T. Kimura, Y. Otsuka, Y. Fujita, S. Majima, and K. Suda, "An Implementation of Service Resource Management Architecture," Technical Proceedings WTC/ISS2002, 2002.
[6] J. Strassner, "A Model Driven Architecture For Telecommunications Systems Using DEN-ng," 1st International Conference on E-business and Teleccomunication Networks Proceedings, Vol. 1, 2004.

Yu Miyoshi
Software Service Group, Network Software Service Project, NTT Network Service Systems Laboratories.
He received the B.E. and M.E. degrees in electronics, information, and communication engineering from Waseda University, Tokyo in 1998 and 2000, respectively. In 2000, he joined NTT Network Service Systems Laboratories, Tokyo, where he was engaged in R&D of network and service operation systems. He is now studying automation settings for network elements. He received the 2004 IEICE Young Researchers' Award. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE) and the Information Processing Society of Japan.

Tatsuyuki Kimura
Senior Research Engineer, Software Service Group, Network Software Service Project, NTT Network Service Systems Laboratories.
He received the B.E. degree in mechanical engineering from Keio University, Tokyo in 1990. He joined NTT in 1990. Since then, he has mainly been engaged in research on transmission network management. He is currently working on the modeling of network management functions and multi-layered network management systems. He is a member of IEICE.