Feature Interaction Detection in Active Networks

Go Ogose, Jyunya Yoshida, Tae Yoneda, and Tadashi Ohta

Soka University, Faculty of Engineering
1-236, Tangi-cho, Hachioji-shi, Tokyo 192-8577, Japan
{gogose, jyoshida}@edu.t.soka.ac.jp
{anne, ohta}@t.soka.ac.jp

Abstract. This paper presents the results of experiments with an active gatekeeper for VoIP. Problems arising with supplementary service programs uploaded by users: such as event conversion and detection of feature interactions between the programs, are presented and their solutions are proposed. The VoIP gatekeeper (GK) and the validation server, based on architecture, proposed by the authors at IWAN2000 and called STAR, were implemented and evaluated. With the proposed GK, the up-loaded program is described using a declarative language ESTR instead of using a procedural language, such as Java. Unlike most conventional architectures, here, one common execution environment is used for execution with all up-loaded programs. 12 service programs were tested to evaluate the proposed systems. Results show that the proposed GK and validation server were reasonable. ...

1 Introduction

VoIP is becoming a standard technique to provide telecommunication services in the IP network. A VoIP gatekeeper has functions such as address translation from a telephone number to an IP address, network access control, and so on. Many architectures for Active Network have been proposed in order to provide new services instantly [1][2]. The authors are investigating using VoIP gatekeeper as an active node. The active VoIP gatekeeper (GK) and the validation server based on the architecture proposed by the authors at IWAN2000 [3] and called STAR (Software archiTecture for Active network using Rule based language), were implemented and evaluated. This paper presents results of experiments with STAR. Problems arising with supplementary service programs uploaded by unspecified users (third party service providers and end users): such as event conversion and detection of feature interactions between programs, are presented and their solutions are proposed. The problem of feature interaction, in particular, is a serious problem in developing an active network.

With the proposed GK, the up-loaded program, from now on referred to as the 'service program', is described using a declarative language ESTR (Enhanced State Transition Rule) instead of using a procedural language, such as Java. ESTR makes it easier to detect feature interactions. Unlike most conventional

J. Sterbenz et al. (Eds.): IWAN 2002, LNCS 2546, pp. 241–252, 2002.

architectures, here, one common execution environment is used for execution with all service programs.

12 service programs, including POTS (Plain Old Telephone Service), were described using ESTR and validated on the validation server. Because of the limitations of the system purchased, 11 service programs were tested on the proposed GK. TWC (Three Way Call service program) was tested using a simulator. It was confirmed that all service programs were executed correctly.

The results of experiments were evaluated by comparison with the international benchmark [4][5]. It was confirmed that all interactions described in the benchmark were detected. In fact, many more interactions, which were not described in the benchmark, were detected.

Results show that the proposed GK and validation server were reasonable.

Section 2 contains a brief description of the proposed architecture, STAR. In section 3, problems with implementation of the GK and the validation server are described. In section 4, solutions for the problems are proposed. Section 5 evaluates the proposed systems.

2 STAR

Much research into Active Networks has been done all over the world. In the IWAN2000, we proposed an architecture for the Active Network for VoIP Gate Way where the up-loaded program, ('service program'), is described using a declarative language. We have also implemented an experimental VoIP gatekeeper based on the proposed architecture (Figure 1). First, the proposed architecture STAR is explained.

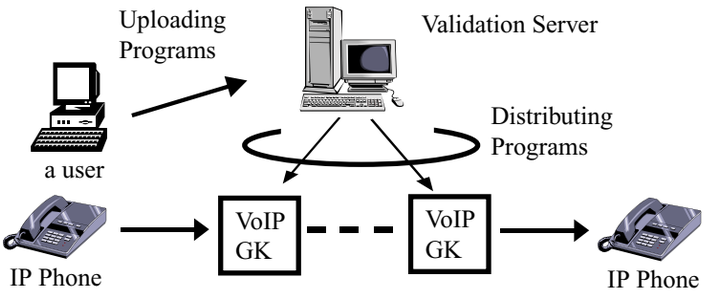

Fig. 1. Proposed Active Network

Characteristics of STAR are as follows:

1) A declarative language, ESTR, is adopted to describe service programs instead of a procedural language, such as Java.

2) One common Execution Environment (EE) is used for all service programs, instead of using an individual EE for each service program.
3) A Validation Server is used to detect feature interactions between service programs described by unspecified users, before the service programs are installed to network nodes.

These characteristics are explained briefly.

2.1 Related Work

Several special languages to describe service programs for the Active Network has been reported [6]. But, no declarative language other than ours has been reported as description language for the service program in the Active Network, especially for an active gatekeeper. Feature interaction detection is one of the most exciting research themes in the field of telecommunication systems research. Much research for definition, detection, and resolution of feature interactions has been discussed, mainly in the International workshop on feature interaction (FIW) [7-11]. But, as far as the authors know, there has been no proposal for feature interaction detection in the Active Network.

2.2 ESTR

The minimum explanation of ESTR, which is necessary for this paper, is given. See paper [2] for a detailed description.

ESTR has the form of Pre-condition, event, Post-condition and Action-description. It is a rule which defines the condition for state transition, state change while the rule is applied, and the system control required for state transition. Pre-condition consists of states description elements called primitives. Primitives are states of terminals or the relationship between terminals that are targets of the state transition. An event is a trigger that causes the state transition, e.g. a signal input to the node and some trigger occurs in the node. Post-condition is the state description part that also consists of primitives. Action-description is the system control description part that shows the system controls required for the state transition. Action-description is described in which follows after Post-condition separated by ', '(see Figure 2). When no system controls are required, the content of is empty. A description example of ESTR is shown in Figure 2.

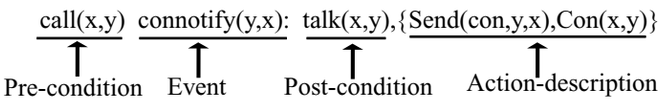

Fig. 2. An Example of ESTR

The example in Figure 2 is explained. Terminal x and y are in calling state, denoted by call(x,y). If terminal y makes off hook, denoted by connotify(y,x), a signal Connect is sent to terminal x, denoted by Send(con,y,x), and terminal x and y transit to talk state, denoted by talk(x,y). call(x,y) and talk(x,y) are called status primitives. All arguments in status primitives are described as variables so that a rule can be applied to any terminals.

When an event occurs, a rule which has the same event, and whose Pre-condition is included in the system state, is applied. When the rule is applied, stored programs designated by Action-description are executed. When the programs end normally, the system state changes as follows. A state corresponding to the Pre-condition of the applied rule is deleted from the current system state and a state corresponding to Post-condition of the applied rule is added. Here, a state corresponding Pre/Post-condition is obtained by replacing arguments in Pre/Post-condition with actual terminals when the rule is applied.

2.3 Execution Environment (EE)

In most conventional architectures for the Active Network, each EE for the service program is uploaded to network nodes with the respective service programs [1]. Consequently, uploaded programs become very large. This adds significantly to the workload of users wanting to upload service programs. However, with STAR, one common EE is used for all service programs. Thus, users simply have to upload service programs only. Comparison of the software architectures of STAR and conventional architectures is shown in figure 3.

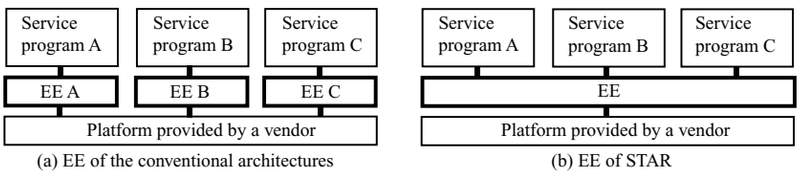

Fig. 3. EE Comparison

For STAR, the EE consists of: 1) an ESTR Interpreter, which selects a rule and executes the selected rule, 2) an Input processing part, which receives input signals from a platform provided by a vendor and converts them into events, and 3) an Action executing part, which analyzes and executes the Action-description of the rule. The software architecture of the EE of STAR is shown in figure 4.

Input Processing Part When a signal is received, the Input processing part converts the signal to an event corresponding to the signal, so that the ESTR Interpreter can handle the event. The event is sent to the interpreter in the EE program.

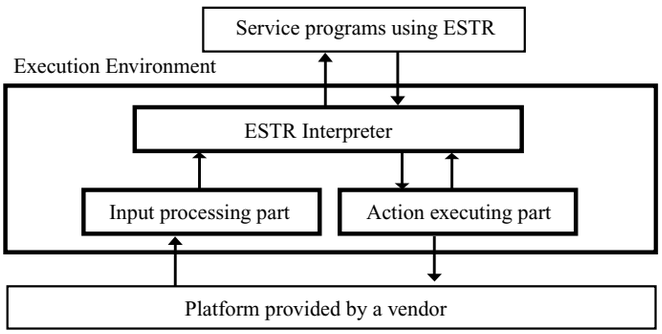

Fig. 4. Software Structure of EE

ESTR Interpreter On receiving an event from the Input processing part, the interpreter selects a rule which has the same event as the one sent.

When the rule to be applied is selected, the Action-description of the rule is sent to the Action executing part in the EE program. If execution in the Action executing part ends normally, the system state is changed according to the Post-condition of the selected rule. If execution in the action executing part ends abnormally, the system state is not changed.

Action Executing Part On receiving an Action-description from the interpreter, the Action executing part of the program analyzes it and decides which programs are to be executed and in what order. The programs will have been stored in the system beforehand by the service provider.

2.4 Validation Server

In an Active Network, unspecified numbers of users send their service programs to the nodes. This may cause feature interactions between the service programs. Feature interactions may cause serious problems to the network. Therefore, before being installed to the network nodes, the service programs are tested at the validation server. Multiple validation servers are set in the network and each validation server sends the service programs to nodes to which the user wants to upload the programs.

3 Problems with Implementing the System

The gatekeeper (GK) was implemented based on the proposed architecture [3]. Some problems with implementing the GK are described.

3.1 Event Conversion

With STAR, as mentioned in section 2, a single EE is used for all service programs up-loaded by unspecified users. As stated above, all the service programs should be processed in the same EE. This means that the EE should understand all the service programs. Thus, how the EE understands the service programs, which the user described freely, is a problem. Since the ESTR interpreter handles a primitive as a mere character string, without knowing the meaning of the primitive, to select the rule and to change the system state, there is no problem for primitives. For the action description part, the problem can be solved by preparing commands beforehand, for users to describe action description parts. The remaining problem is event conversion. The EE is invoked by receiving a signal. The ESTR interpreter is invoked by the event. Therefore, in order to invoke an interpreter, the signal has to be converted into an event.

On receiving a signal, the Input processing part converts the signal into an event described in the service programs. The signal event conversion is done as follows: the Input processing part searches an event conversion table, and converts the signal into the corresponding event.

The problem, therefore, is how to automatically revise the event conversion table when a new service program, which uses new events, is installed into the node.

3.2 Validation Server

Services that independently operate normally will behave undesirably when simultaneously initiated with another service. This behavior is called a feature interaction. As shown in AIN, JAIN, Parlay and Active Networks, a telecommunication network architecture changes to a new one where the third-party service providers can provide network services. This architecture enables multiple providers to provide services in the same network, simultaneously. As a result, feature interactions between different provider services inevitably occur. This causes serious problems to service deployment. A great deal of research on detecting feature interactions has been done all over the world [7-17] to solve the problems.

4 Solutions

4.1 Event Conversion

An event conversion table is used to convert a signal to an event. The event conversion table provides the relationship between an input signal and a corresponding event. To revise the event conversion table for newly used events, the following method is proposed:

When a user plans to use a new event in a new service program, he/she should send information on the correspondence relation between a new event and a signal with the new service to the validation server. After checking feature

interactions, the validation server sends the information to the GK with the new service program. On receiving the information, the GK registers the information into the event conversion table, so that the Input processing part can identify the correspondence between a new event and a signal. On receiving a signal, the Input processing part converts the received signal into a corresponding event by referring to the event conversion table.

As an example of a registration process of a new event and an event conversion, a registration process and an event conversion of a specific number are explained. A specific number is a special telephone number for identifying a specific service.

Suppose, sp1setup(x) and 1901 are a newly defined event used in a service program and specific number input by an end user, respectively. First, the user, who uploads the service program which uses a specific number as an event, sends the information which defines the correspondence relation between the new event sp1setup(x), and the specific number, 1901, beforehand, to the validation server (VS). On receiving the information, the GK puts 1901 and sp1setup(x) into the event conversion table (Figure 5).

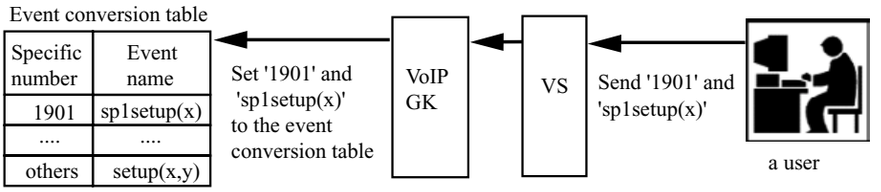

Fig. 5. Event Conversion Table Set

Next, an event conversion process for a specific number is explained (Figure 6). When the Input processing part receives a SETUP signal, the Input processing part calls a number analysis program. The number analysis program decides whether the terminating telephone number contained in a SETUP signal is a specific number or a usual telephone number, by referring to the event conversion table. If the terminating telephone number is a specific number, the number analysis program returns the event sp1setup(T1) to the Input processing part. If the terminating telephone number is a usual telephone number, the number analysis program returns the event setup(T1,T2) to the Input processing part. (Here, T1 and T2 are the originating terminal identifier and the terminating terminal identifier, respectively.)

4.2 Validation Server

Outline The Authors have proposed efficient methods for detecting feature interactions by analyzing Pre-conditions and Post-conditions of uploaded ESTR

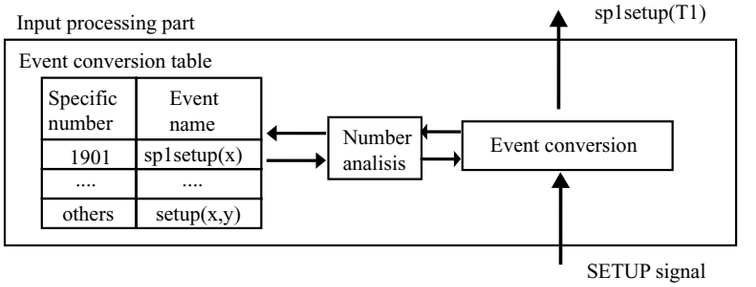

Fig. 6. Event Conversion Table Use

rules. To detect feature interactions between service programs uploaded by unspecified users, a validation server based on the proposed methods was implemented (Figure 7).

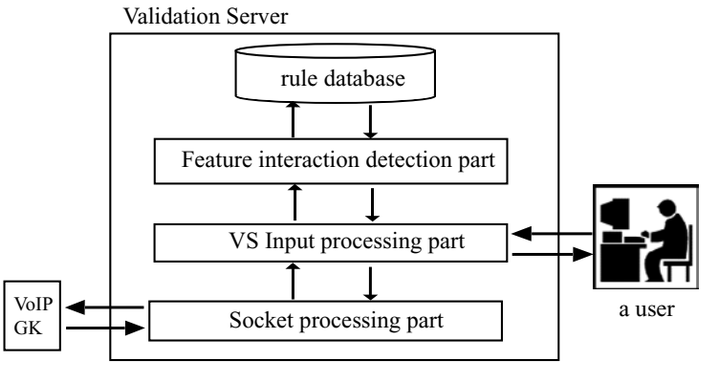

Fig. 7. Validation Server

The validation server checks for each uploaded ESTR rule whether the feature interactions are caused with existing rules. Feature interactions are classified into 7 categories [14]. The validation server implemented here detected the following 5 categories, which frequently occur in telecommunication services:

- Non-determinacy: there is more than one next state for one event.
- Appearance of abnormal states: Appearance of states which are not defined in the individual service.
- Disappearance of normal states: Disappearance of states which are defined in an individual service.

- Appearance of abnormal transitions: Appearance of transitions, which are not defined in an individual service.
- Disappearance of normal transitions: Disappearance of transitions, which are defined in an individual service.

First one causes EE service cancellation. Resting 4 interactions, called 'semantic interaction', cause various service malfunctions. When a feature interaction is detected between service A and B, the validation server regulates the service activation so that either of service A or B can be activated simultaneously.

Detection Algorithm A brief explanation of detection algorithm for semantic interactions will be described. For detailed descriptions, please refer to paper [13], [15], [16], and [17].

Semantic interactions can be considered as follows. Suppose two services are activated. When either specification of the services is applied, one state transition, according to the specification, clashes with the specification for the other service. Therefore, feature interactions are detected as follows: To make a rule pair, select a rule from each service, respectively, which is applicable to the same system state. Apply either rule to the system state. Check if the state transition by the rule causes an abnormal state transition from the viewpoint of the other service whose rule is not applied.

In some conventional methods, all possible states must be generated in one way or another and all state transitions checked to detect feature interactions. This causes an explosion of the number of states, resulting in a huge increase in computation time for detecting feature interactions.

In our method, on the other hand, interactions are detected solely by analyzing Pre-conditions, events, and Post-conditions of selected rules as follows:

step 1) If the rule pair can be applied to the same system state, go to step 2. Otherwise, a feature interaction is not detected.
step 2) If both rules have the same event, go to step 4, otherwise, go to step 3.
step 3) If the Pre-condition of the rule, which is not applied, is not preserved in the next system state, a feature interaction is detected. Otherwise, a feature interaction is not detected.
step 4) If the Post-condition of the rule, which is not applied, is not preserved in the next system state, a feature interaction is detected. Otherwise, a feature interaction is not detected.

Suppose, r_a and r_b denote selected rules from service a and service b, respectively. r_{ac}, r_{an}, r_{bc}, and r_{bn} denote Pre-condition of r_a, Post-condition of r_a, Pre-condition of r_b, and Post condition of r_b, respectively. If redundancy can be neglected, formal descriptions of conditions used to detect interactions in step3 and step 4, respectively, are given as follows:

$$\text{For step3: } (r_{bc} - r_{ac}) \cup r_{an} \not\supseteq r_{bc}$$
$$\text{For step4: } \{(r_{bc} - r_{ac}) \cup r_{an} \not\supseteq r_{bn}\} \vee \{(r_{ac} - r_{an}) \not\supseteq (r_{bc} - r_{bn})\}$$

This method does not require any state creation and does not cause a huge increase in computation time for feature interaction detection. Other problems in implementing the feature interaction detection system are described in the next section. Redundancy will be evaluated in section 5.

Detection Process The detection process of the detecting system is shown in Figure 8.

First, select a pair of rules from the target service, one rule from each service, respectively [13]. After assigning real terminals to terminal variables in each rule [16], check if the pair of rules causes any non-determinacy interactions [17] and/or semantic interactions [13]. If the pair of rules causes any interactions, execute a reachability test. If the system state is reachable, the pair of rules actually causes interactions.

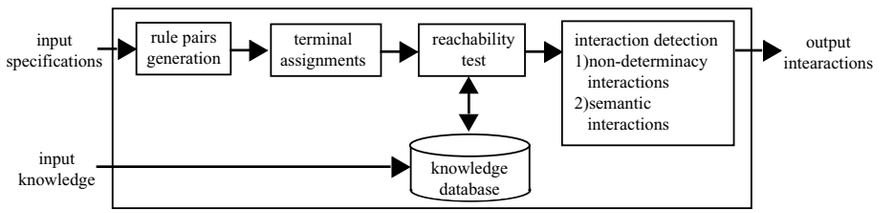

Fig. 8. Detection Process

5 Evaluations

A VoIP gatekeeper and a validation server were implemented based on the proposed methods mentioned in section 4. The gatekeeper was implemented by modifying an existing commercial gatekeeper.

5.1 Event Conversion

12 services programs: POTS, OCS(Originating Call Screening Service), TCS (Terminating Call Screening Service), CFV(Call Forwarding Service), FB(Free phone Billing), FR(Free phone Routing), CC(Charge Call service), CFBL(Call Forwarding Busy Line), TL(Teen Line service), TWC(Three Way Call Service), and two game service programs using specific numbers, a memory test game and a personality analysis game were described using ESTR. 8 services programs such as POTS, OCS, TCS, CFV, FB, FR, CC, and CFBL were uploaded to the GK. It was confirmed that all programs worked correctly. Although the two game service programs and TL were uploaded to the GK, it was only possible to

confirm event conversion functionality; this was because of the limitations of the commercially purchased gatekeeper and gateway. The same restrictions made it necessary to test the TWC using a simulator; results confirmed that it worked correctly.

Thus, it was confirmed that the proposed method of event conversion was reasonable.

5.2 Validation Server

Except for POTS, the 11 services mentioned in the previous section were validated on the implemented validation server. A benchmark for feature interaction detection was published in April 2000 [4][5]. To evaluate the proposed validation server, experimental results for nine service programs, which were shown in the benchmark, were compared with the benchmark. For example, the pairs of services of the feature interaction detected are TCS and CFBL, TCS and OCS, and CFV and FB. The feature interaction of TCS and CFBL is briefly explained. Terminal B activates CFBL, and registers terminal D as a forwarding terminal. Terminal D activates TCS, and registers terminal A as a screening terminal. Terminal A is off the hook, and terminal B is talking with terminal C. Then, if terminal A dials terminal B, and CFBL is applied, the state of terminal A will change the calling state with the terminal D. This is an illegal state transition because the system state doesn't change to a correctly state of TCS.

All the interactions described in the benchmark were detected, in fact, there are many more interactions detected, which were not described in the benchmark. It was confirmed manually that there was no redundancy and miss-detection.

Evaluation results for detecting interactions are given as follows:

1) A large number of interactions were detected, compared to what was shown in the benchmark. There was no redundancy or miss-detection. Thus it was confirmed that the proposed algorithm, including the detection system, is reasonable.

2) The reason so many interactions were detected is that, first of all, since ESTR allows a multiplicity of services to share states it has the mechanism to create feature interactions. With the conventional detection methods, limited terminal assignments were tested to suppress detection time because the correct way of making terminal assignments had not been made clear, but with the proposed methods, the way of making terminal assignments was clarified [16] and other filtering methods were developed. As the filtering methods reduced detection time considerably, all terminal assignments were considered and far more interactions were detected.

6 Conclusion

An active gatekeeper for VoIP and a validation server were proposed. Problems were identified and their solutions were proposed. The proposed systems were evaluated and it was confirmed that they were reasonable.

In future work, many more supplementary service programs will be applied in order to evaluate the proposed architecture and systems. Further work will be on how to guarantee that one user's action doesn't consume too many resources or disrupt other user's services. We are planning to make separations between users so that the problem does not occur.

References

1. H. Yasuda Ed., "Active Networks," Lecture Notes in Computer Science 1942, Oct. 2000.
2. T. Morinaga, G. Ogose, and T. Ohta, "Active Networks for VoIP GW using Declarative Language," Proc. of APCC2001, pp.89-92, Sep. 2001.
3. S. Komatsu and T. Ohta, "Active Networks using Declarative Language," Proc. of IWAN2000, pp.33-44, Oct. 2000.
4. N. Griffeth et al., "Feature Interaction Detection Contest of the Fifth International Workshop on Feature Interactions," The Interanational Journal of Computer and Telecommunications Networking, Computer Networks 32 (2000) pp.487-510, April 2000
5. N. Griffeth et al., "A Feature Interaction Benchmark for the First Feature Interaction Detection Contest," The Interanational Journal of Computer and Telecommunications Networking, Computer Networks 32 (2000) pp.389-418, April 2000
6. K.L. Calvert et al., "Directions in Active Network," IEEE Com. Magazine, Vol.36, No.10, pp.72-78, Oct. 1998.
7. L.G. Bouma et al., Feature Interactions in Telecommunications Systems, IOS Press, 1994
8. K.E. Cheng and T. Ohta, Feature Interactions in Telecommunications III, IOS Press, 1995
9. P. Dini et al., Feature Interactions in Telecommunication Networks IV, IOS Press, 1997
10. K. Kimbler and L.G. Bouma, Feature Interactions in Telecommunications and Software Systems V, IOS Press, 1998
11. M. Calder and E.Magill, Feature Interactions in Telecommunications and Software Systems VI, IOS Press, 2000
12. T. Ohta et al., "Classification, Detection and Resolution of Service Interactions in Telecommunication Services," Proc. of FIW'94, pp60-72, May 1994
13. T. Yoneda and T. Ohta, "A Formal Approach for Definition and Detection of Feature Interactions," Proc. of FIW'98, pp.202-216, Sep. 1998.
14. T. Ohta and F. Cristian, "Formal Definitions of Feature Interactions in Telecommunications Software," IEICE transactions on Fundamentals of Electronics, Communications and Computer Science, vol.E81-A No.4, pp.635-638, April 1998
15. T. Yoneda and T. Ohta, "Automatic Elicitation of Knowledge for Detecting Feature Interactions in Telecommunication Services," IEICE transactions on information and systems, vol.E83-D No.4, pp.640-647, April 2000
16. T. Yoneda and T. Ohta, " Reduction of the Number of Terminal Assignments for Detecting Feature Interactions in Telecommunication Services," Proc. of ICECCS, pp.202-209, Sep. 2001.
17. J. Kobayashi, T. Yoneda and T. Ohta, "An Effective Method for Testing Reachability Using Knowledge in Detecting Non-Determinacy Feature Interactions," IEICE Trans. on Information and Systems, vol.E85-D No.4, pp.607-614, April 2002