Design and Evaluation of Feature Interaction Filtering with Use Case Maps

Pattara Leelaprute[1], *Masahide Nakamura*[2], *Ken-ichi Matsumoto*[2] *and Tohru Kikuno*[1]
[1]*Graduate School of Information Science and Technology, Osaka University, Japan*
{*pattara, kikuno*}*@ist.osaka-u.ac.jp*
[2]*Graduate School of Information Science, Nara Institute of Science and Technology, Japan*
{*masa-n, matumoto*}*@is.aist-nara.ac.jp*

ABSTRACT — Feature interaction (FI, in short) is a functional conflict among multiple telecommunication services, which is never expected from services in isolation. Detecting all possible FIs is an expensive and even infeasible task, due to the combinatorial explosion in the number of service combinations and scenarios. To reduce the cost of FI detection, *FI filtering* is known as a low-cost process conducted prior to FI detection, which identifies *FI-prone* service combinations. However, each FI-prone combination usually contains many service scenarios. Deriving only *FI-prone scenarios* in the combination makes the FI detection process more efficient.

This paper proposes a new FI filtering method consisting of two phases. The proposed method extensively uses the requirement notation Use Case Maps (UCMs). We characterize each service by a stub configuration of UCMs. The stub configuration is formalized as a matrix form, called SC-matrix. With the SC-matrix, Phase 1 derives FI-prone service combinations. Next, in Phase 2 we derive the FI-prone scenarios based on two heuristics: (H1) FI tends to occur when two (or more) services are activated, or (H2)FI tends to occur when a service bypasses a feature of the other service.

Through a practical experiment, we have evaluated the proposed method with respect to; scenario coverage, filtering quality and reduction ratio in the number of scenarios. The result showed that the FI-prone scenarios obtained successfully covered all scenarios that lead to actual FIs. Also, the proposed method derived only 10% of the total number of scenarios as FI-prone ones, which implies that 90% reduction of the cost for the scenario investigation was achieved.

KEY WORDS — Feature Interactions, Telecommunication Services, FI filtering, Use Case Maps

1. Introduction

In recent years, as seen in the Advanced Intelligent Network [32], Mobile Network, the Next Generation IP Network [22] and such, communication networks have gone through a phenomenal evolution. Owing to these developments, communication services are diversifying from the conventional "Call waiting" and "Call forwarding when busy" to multimedia services such as electronic mails [12], video-conferences, and WEB services [29, 34]. The flood of services results in functional conflicts which are never caused by a single service alone. These conflicts are widely recognized as *Feature Interaction* [5, 33] (FI, for short). The FIs seriously hamper the development of new services [6]. Detecting all possible FIs among complex services requires a significant cost due to the nature of communication services; concurrency, branches of complex scenarios, and the massive number of service combinations.

In practical service development, the analysis of feature interactions has been conducted in an ad hoc manner by subject matter experts. However, as the number and complexity of services grow, the ad hoc analysis does not work in a feasible way, which leads to time-consuming service design and testing without any interaction-free guarantee.

Much research has been conducted to tackle the FI problem [17]. Most of the communication services are usually modeled by finite state machines: FSMs. In this model, a global state consisting of user's local states successively moves to the next state by the occurrence of a user's event [27, 31], and FIs are defined on certain states in the FSM where some undesirable properties hold. The way to detect FIs in the FSM is to enumerate all possible states and to identify undesirable states that cause FIs [6, 7, 10, 19]. The advantage of this model is that it can detect all existing FIs. However, due to the concurrent characteristics of telecommunication services, the number of states in the model becomes exponential in the face of the number of services and users. This is called the *state explosion problem*.

Thus, research that aims to reduce the number of states exists. Cameron et al. [6] proposed the tool CADRES-

FI which utilizes state abstraction based on heuristics. Static FI detection methods that do not have to enumerate reachable states in the state transition model also exist. By using Petri Net to reduce the transition states, Nakamura et al. [23] proposed the P-invariant and Kawarazaki et al. [16] proposed the T-invariant. Yoneda et al. [30] reduced complexity by using the structure of a rule-base specification, STR (State Transition Rules) [15, 27], and Harada et al. [13] proposed the detection algorithm EXH for rule-base specification.

As an alternative approach to reducing the cost of FI detection, Kimbler [20] proposed a low-cost method called *FI filtering*, which is conducted before the FI detection process. FI filtering indicates whether or not the combination of services is likely to cause FI, or has a possibility in causing FI. Therefore before FI detection, we concentrate on only combinations that are likely to cause FI (FI-prone). Moreover, in each FI-prone combination, there exists several scenarios, some of which are not relevant to the actual FIs. Hence, deriving only relevant scenarios from the FI-prone combinations helps us to concentrate on the problematic scenarios in the latter process.

This paper proposes a new FI filtering method at the requirement stage of telecommunication services and features. Implementing FI filtering at the requirement stage is more efficient for eliminating FI at the former process, the requirement stage (the service design stage) than at the latter process, the coding and execution processes [16]. We employ a requirement notation method, called the Use Case Maps (UCMs) [2, 4]. So far, several notations have been proposed. Aho et al. [1] proposed an event-base language known as "Chisel" at the requirement level. Chisel's well-defined semantics based on trace equivalence provided the framework for FI detection, but it is unable to describe the concurrent behaviors of its characteristics. Also, because the feature addition is performed by "gluing nodes" in the chisel diagram, the scenario of the entire system cannot be seen. For the other service description methods, SDL (Specification and Description Language)[28], LOTOS (Language Of Temporal Ordering Specification) [3, 9] and STR [15, 27], are also proposed. However, these are formal description methods which should be adopted at the specification level. Therefore, it is not quite easy to apply them directly to the requirement level, in which system details are not yet determined.

The reasons we chose UCMs for FI filtering are summarized as follows: (1) in the entire picture of global call scenarios at the requirement level, UCMs do not require understanding the details of system behaviors or complicated semantic models. (2) in concurrencies, alternatives and hierarchical designs which are indispensable to requirements level notation can be easily achieved with UCMs, (3) the tool called UCM navigator [8] necessary for syntactically correct UCM notations is provided.

The proposed FI filtering method consists of two phases. In the first phase, we identify FI-prone service combinations. Next, in the second phase, we derive FI-prone scenarios from each FI-prone combination.

In UCMs, we use the concept of *stub plug-in* to add features into a basic call model (or POTS — Plain Ordinary Telephone Service). Specifically, we describe the basic call scenarios in the top-level UCMs, which are called a *root map*. Then we represent supplementary features as a set of sub UCMs, which are called *submaps*. Next, we put them into places (called stubs) in the *root map* to extend or modify the basic call. The key idea for the first phase is to characterize each service by a stub configuration in the root map, and then see how the stub configuration changes according to the service combination. To facilitate the representation of the stub configuration and feature combination, we propose the stub configuration matrix and a combination operator of the matrices. Consequently, the first phase derives one of the following verdicts for each feature combination: (a) FI occurs, (b) FI never occurs and, (c) FI-prone.

At the second phase, we derive FI-prone scenarios from the FI-prone combinations of the two services. The derivation method is based on two heuristics: (H1) FI tends to occur in scenarios where both services are activated, and (H2) FI tends to occur in scenarios when the activation of one service bypasses the activation of the other service. Deriving only the scenarios relevant to the potential FIs makes the FI detection process efficient.

As an experimental evaluation, we conducted FI filtering and subsequently scenario derivation on the 8 types of services taken from the FI Detection Contest held at the International Conference FIW2000 [21]. We evaluated the proposed method from the viewpoints of coverage, filtering quality and scenario reduction ratio. As a result, we confirmed the legitimacy of the proposed method, since all actual FI-occur scenarios are covered by the derived FI-prone scenarios. Also, if a feature combination does not have any scenarios derived by Heuristic H1 or H2, then the combination does not cause actual FIs. Therefore, we can improve the filtering quality by concluding such combinations to be (b) FI never occurs. It was also shown that the derived scenarios were only 10% in the total number of scenarios. Thus, the proposed method was able to filter 90% of the irrelevant scenarios, which implies a significant cost reduction of the scenario investigation.

2. Feature Interaction

In the literature, the terms "features" and "supplementary services" are often used interchangeably. We use *feature* to refer to a set of service functionalities that extend or modify the basic call.

2.1. Feature Examples

Throughout this paper, we use the following features as examples.

(a) Calling Number Delivery Blocking (CNDB): This service blocks the provision of the caller's number at the terminating side. Suppose that user x subscribes to the CNDB, when x dials y, x's telephone number will not appear on y's telephone.

(b) Terminating Call Screening (TCS): This service allows a subscriber to screen incoming calls based on a screening list. Suppose that user x registers y in the screening list, then any call from y to x would be screened.

(c) Call Forwarding on Busy (CFB): A subscriber of this service can forward incoming calls to another pre-determined number when the subscriber is busy. Suppose that user y sets the forwarding number to z, and that y is busy. If x dials y, then the call would be forwarded to z and x would be connected to z.

(d) Reverse Charging (RC): This service is known as freephone billing, and allows the subscriber to be charged for the calls in which the subscriber is the terminating party. Suppose that user y subscribes to the RC, when x dials y, y will pay the telephone charge instead of x.

(e) Splitting Bill (SB): This service allows the sharing of costs between the partners in a call. A company might provide local call charge lines to customers as a service, in which case the customer (the originator of the call) pays the local charges and the company pays the rest. Suppose that user y subscribes to the SB and allows the originator who calls y to only pay the local call charge. When x dials y, x will pay the local call charge and y will pay the rest of the long-distance call charge.

2.2. Practical Examples of Feature Interactions

Let us look at practical examples of FIs. In the following, A, B, C denote actual users. Note that these FIs do not occur if the services are used in isolation.

FI-(a) - Interaction RC & SB: Suppose that B subscribes to RC and SB. If A dials B, it cannot be determined that the call should be charged 100% to B by the function of RC, or charged by the rule of the payment of SB which B had set.

FI-(b) - Interaction TCS & CNDB: Suppose that B subscribes to TCS and puts A's telephone number on the screening list, and that A subscribes to CNDB. Now, if A dials B, because of the call number delivery blocking function of A, B can not screen the incoming call from A because B cannot know whether or not the phone number of incoming call is A's.

FI-(c) - Interaction CFB & RC: Suppose that B subscribes to RC, and that A subscribes to CFB and sets the forwarding address to B. If C dials A when A is busy, then CFB forwards the call to B. When A forwards the call to B, the RC function is bypassed, and thus inactivated. As a result, A has to pay the call charge, although B is the freephone subscriber.

3. Use Case Maps for Service Description

3.1. Basic Principles

Use Case Maps[1](UCMs in short) is a requirements notation method designed to bridge the gab between the requirement (use cases) written in the natural language and the detailed design written in some strict specification language.

UCMs describe a system by a set of *scenario paths* with causally-ordered *responsibilities* (events). In this paper, we briefly review some concepts used in this paper. Additional details can be found in [2, 4].

(a) The core notation consists of only *scenario paths* and *responsibilities* along the paths. A path starts at a *starting point* (depicted by a filled circle) and ends at an *end point* (shown as a bar). The *starting point* can be associated with a *precondition* and the *end point* can be associated with a *postcondition*, both of which are represented in square brackets []. Between the start and end points, the scenario path may perform some responsibilities along the path, which are depicted by crosses $\times$ with labels. Responsibilities are abstract activities that can be refined in terms of functions, tasks, procedures, events, and are identified *only* by their labels. Tracing a path from the start to the end is used to explain a scenario as a causal sequence of events.

(b) Several paths can be composed by superimposing common parts and by introducing *fork* and *join*. There are two kinds of forks/joins. One is the *OR-fork/join*, describes *alternative* scenario paths, which mean that one of the paths is selected to proceed at each branch. The other type is the *AND-*

[1]Strictly speaking, the UCMs discussed here are *unbound UCMs* without system components shown explicitly since this paper focuses on the requirements entities for FI filtering.

fork/join, depicted by branches with bars, which describes *concurrent* scenario paths.

(c) A *stub plug-in* concept allows UCMs to have a hierarchical path structure, to defer details, and to reuse the existing scenarios. A *stub*, depicted by a dotted diamond, identifies a place where path details in the UCM are described by other UCMs, called *submaps* (or sub-UCMs). On the other hand, the UCM with the stub is called a *root map* (or root-UCM). In this paper, we assume that submaps are not allowed to contain stubs. This assumption is for simplicity, and will be relaxed in future research.

A submap can be *plugged into* a stub in a root map. This is done by binding the start and end points of the submap to the corresponding entrances and exits of the stub in the root map, respectively, in accordance with labels on the start and end points.

3.2. Describing Services by UCMs

In the domain of telecommunication services, such as the ones in IN (Intelligent Networks)[32], sophisticated service functions are usually implemented by a *basic service* and its *supplementary services (features)*. In the telephony services, the basic service is known as the *basic call model* (or POTS — Plain Ordinary Telephone Service). The basic service is the base for every supplementary service; the addition of supplementary services is achieved by the reuse of almost all the functions, as well as partial alteration or extension of the basic service.

Focusing the basic/supplementary service provision, we make use of the UCMs to describe the service. First, we describe the basic call model as the *root map* of UCMs. Second, for the scenario of the basic service, we plug the *default submap* [2] into the *root map*. Finally, we describe the supplementary services as the *submaps* of UCMs, and plug them into the root map to make changes to the default scenario.

3.2.1. Basic Call Model Figure 1 represents UCMs for the basic call model according to the second FI detection contest specifications [21]. This basic call model is the so-called *global call model* of the *end-to-end view* [11], which contains both the caller's and the callee's scenarios in one model.

There are eight UCMs in Figure 1 and each is identified by a name (*identifier*), e.g. *root*, *def*$_3$. The responsibilities in the diagram are those explained in [21]. Symbols A, B, C, D refer to *constants* representing actual users (subscribers). Symbols V, W, X, Y are *variables* to which constant values are assigned dynamically. In this example, A is the caller, whereas Y is the callee. Since Y is a variable, the callee may change depending on the destination of the call. When A calls B, for instance, then Y is B.

For instance, take UCM *def*$_5$ in Figure 1(b). This UCM explains a scenario where *busytoneA* occurs. Each scenario path can be associated with a *pre-condition*. The pre-condition is a condition for the path to start with. For example, a pre-condition "[A = idle]" in $root$ represents the scenario which starts only when A is idle.

An example of an *OR-fork* would be, UCM *def*$_1$ in Figure 1(b), which contains an OR-fork describing two possible scenarios: "$dialtoneA$ occurs, and the scenario ends at $out11$" or "$dialtoneA$ occurs, and the scenario ends at $out12$". It is also possible to explicitly specify conditions for path selection. This is done by using *guards*, represented with square brackets [] at the OR-fork. For example, a guard [Y = idle] at an OR-fork in a UCM *root* implies that the scenario proceeds to the upper path when Y is idle.

As an example of *AND-fork/join*, in the upper part of UCM *Root Map* in Figure 1, one AND-fork and one AND-join appear between *out41* and *in61*. After *out41*, two scenario paths start concurrently. As a result, *ringbackY* and *alertA* are performed in any order (by interleaving semantics). The concurrent paths are synchronized at the AND-join, and then the scenario ends before *in61*.

For the example of the *stub plug-in*, the root map in Figure 1(a) contains seven stubs. All other UCMs are submaps. Let us plug a submap *def*$_1$ into stub 1 in *root*. The starting point $in11$ is connected to the stub entry $in11$, and end points $out11$ and $out12$ are connected to the exits $out11$ and $out12$, respectively. Similarly, other submaps *def*$_i (2 \leq i \leq 7)$ are plugged into the corresponding stubs $i (2 \leq i \leq 7)$, which completes the whole scenario path structure of the basic call model.

3.2.2. Supplementary Features As mentioned in subsection 3.2, adding the supplementary features extends the scenarios of the basic call model. In our framework, this is achieved in a simple way by using the stub plug-in concept of UCMs. Intuitively, we only *replace* some default submaps with specific ones, called *feature submaps* which describe the features' scenarios. In this subsection, we explain only how to add each single feature at a time to the basic call. Adding multiple features is achieved by combining the individual features using a combination operator $\oplus$. This operator will be presented in Section 5.

Figure 2 shows feature submaps for the features presented in Section 2.1. Each submap has a name (identifier) with an index which represents a stub location of the root map to be plugged into. For example, let us add SB to the basic call. Adding SB to the basic call is done by plugging feature submaps sbV_2 in Figure 2 into stub 2 of the root map in Figure 1(a). As a result, *def*$_2$ is replaced by sbV_2 and all other submaps *def*$_i (i = 1, 3, 4, 5, 6, 7)$ remaining to be reused in the corresponding stubs.

[2] For convenience, we call the submaps for basic call scenarios *default submaps*.

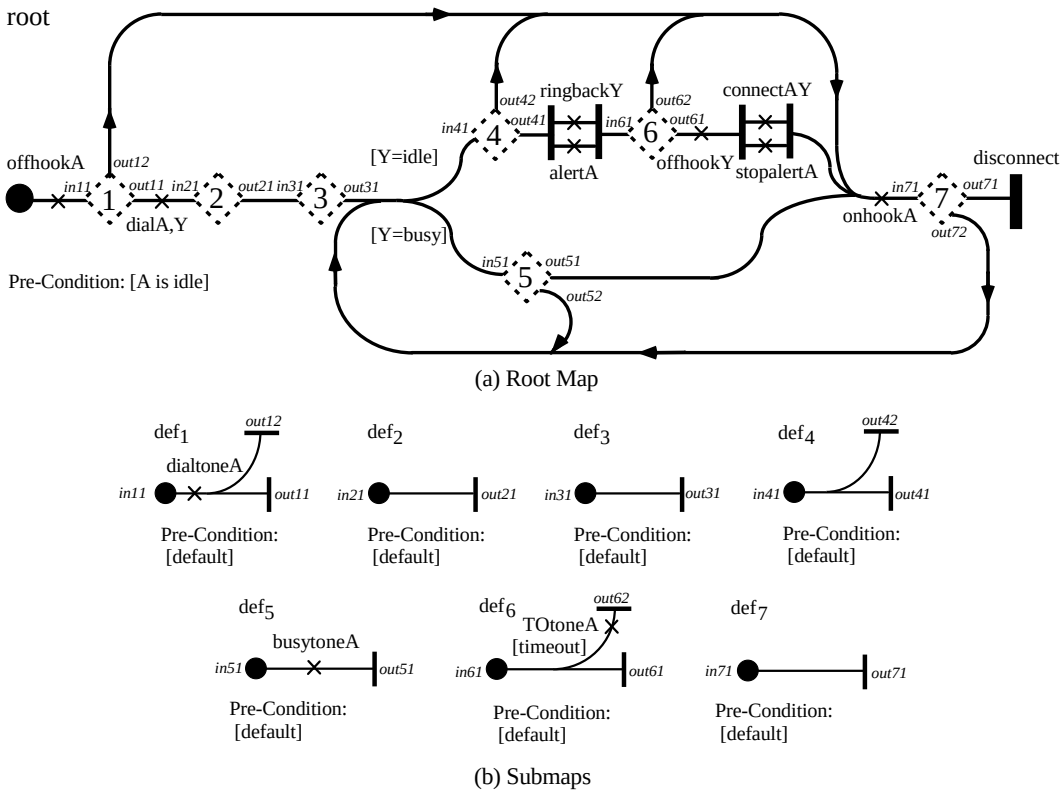


Figure 1. Use Case Maps for the basic call model

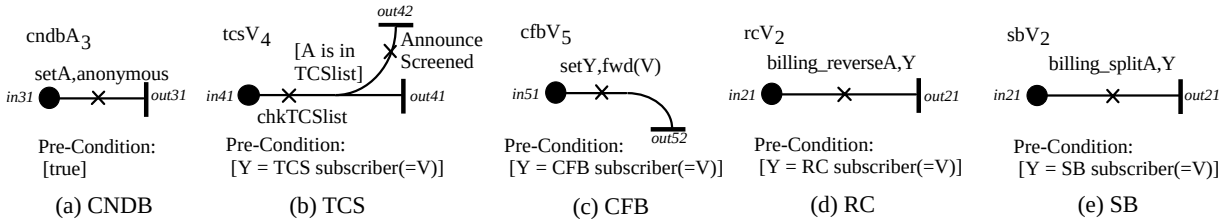


Figure 2. Submaps for supplementary features

In UCMs, a stub is allowed to contain multiple submaps, whose selection can be determined at run-time according to a *submap selection policy* [3], which helps to describe dynamic situations in scenarios. The selection policy is usually described with pre-conditions of submaps. When there are several submaps for one stub, the submap whose pre-condition takes true value is chosen to be plugged into the stub. If two or more pre-conditions hold simultaneously, then we cannot decide which submap should be plugged in. This is a problematic situation called non-determinism [24] and it must be avoided. For convenience, let $pre(f_i)$ denote a pre-condition of a submap f_i.

[3]In this sense, the stubs discussed here are called *dynamic stubs*, strictly speaking.

Let us illustrate how to plug multiple submaps into a stub. For example, suppose that B is an SB subscriber. In case of B subscribing to this service, B can set the percentage of the telephone charge he/she wants to pay. If B sets to pay the long-distance call charge for the originator, when the caller A makes a call to B, A pays only the local call charge and B will pay the remaining long-distance charge. When A calls C, however A pays all of the call charges normally. Thus, the call scenario dynamically changes depending to whom A makes a call.

To explain this, let us take a submap sbV_2 shown in Figure 2. The variable V in sbV_2 represents an SB subscriber. We assign a value B to V. Then, the submap sbV_2 and its pre-condition $pre(sbV_2) = [Y = V]$ are instantiated to sbB_2 and $[Y = B]$. sbB_2 is plugged into

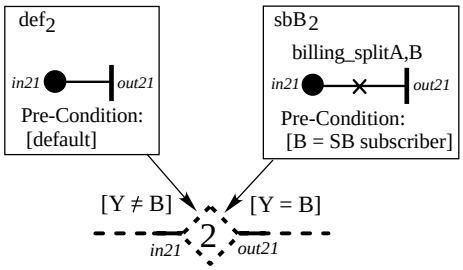


Figure 3. Plugging two submaps into one stub

stub 2 only when $[Y = B]$ holds, i.e. the callee Y is B. On the other hand, a submap *def*$_2$ in Figure 1(b) is also for stub 2, which describes *default* scenarios in the basic call. The $pre(def_2)$ is a default, which means *def*$_2$ is plugged in when none of the other submap's pre-conditions hold. One of sbB_2 or *def*$_2$ is chosen to be entered in stub 2 depending on whom A calls (see Figure 3). This shows that SB applies only when A calls B. If the callee Y is C, the call proceeds to the default scenario.

For different submaps f_i and g_i, if $pre(f_i)$ and $pre(g_i)$ hold simultaneously, then the submap selection policy does not work correctly. When two pre-conditions are satisfied at the same time, a *non-deterministic behavior* occurs regarding which of f_i and g_i should be chosen to be plugged into stub i. If $pre(f_i)$ and $pre(g_i)$ are not simultaneously satisfiable, we say that f_i and g_i are *mutually exclusive*, denoted by $mex(f_i, g_i)$. In general, since UCMs do not force any formalism on pre-conditions, evaluating these UCMs without human input is difficult.

Therefore, we assume that for any pair f_i and g_i given, the scenario designer can always tell whether $mex(f_i, g_i)$ holds or not. All the submaps for the same stub must be mutually exclusive to achieve a consistent selection policy.

For instance, since *def*$_2$ and sbB_2 are mutually exclusive as shown in Figure 3, *mex(def*$_2$, sbB_2) holds. However, rcB_2 and sbB_2 are NOT mutually exclusive, since both preconditions of rcB_2 and sbB_2 are $[Y = B]$ (See Figure 2). As a result, scenarios in rcB_2 and sbB_2 cause non-determinism when $[Y = B]$ holds. The non-deterministic behavior is well known as a typical situation of FIs [24].

4. Characterizing Feature Interaction

Researchers agree on an informal definition of FI: *FI occurs iff combining multiple features changes the requirements properties of each feature in isolation.* The definition is not formal enough to perform FI detection. Hence, researchers have been trying to give formal definitions of FIs. As a result, different definitions are proposed for different FI detection frameworks.

However, the aim of this work is not to present an FI detection method, but to propose an FI filtering method, which is supposed to be a quick and rough evaluation deployed before the FI detection process. In order to make the proposed method *generic*, i.e. applicable to different FI detection frameworks, we briefly characterize FIs by a necessary condition and a sufficient condition with respect to call scenarios of users.

Let us consider again the informal definition above. First, we can say that the requirement properties do not change unless each user's call scenario changes. So, if FI occurred, some call scenarios must have been changed by the feature combination. Therefore, we have:

Condition C1 If FI occurs, then the combination of multiple features changes some user's call scenarios in an individual feature.

All FIs in Section 2.2 can be explained by Condition C1. In every example, the user's call scenarios have been changed somehow by feature combination.

Next, we focus on a typical type of FI, called non-determinism [24]. This type of FI occurs when the feature combination changes a call scenario in a way that multiple scenarios can be triggered in the same condition. Note that, however, not all FIs are caused by this non-determinism.

Condition C2 If a combination of multiple features enables different call scenarios to be performed under the same conditions, then FI occurs.

A typical FI characterized by C2 is FI-(a), as shown in Section 2.2. Note that the reverse of each condition does not necessarily hold. Thus, our characterization of FI by the above conditions is relatively weak. However, the characterization is essential in performing FI filtering with low cost.

5. Phase 1: Identifying FI-prone Feature Combinations

We have 2 phases in our proposed Feature Interaction Filtering. The key idea of the first phase is to categorize each service according to the stub configuration, devise the service scenario for each user, and see how each stub configuration changes according to a concurrent execution of multiple features. For this purpose, we propose a matrix representation of the stub configuration, called the SC-matrix. With the SC-matrices, the first phase derives one of the following verdicts for each feature combination: (a) FI occurs, (b) FI never occurs and (c), FI-prone. Next, at the second phase, we derive FI-prone scenarios from the FI-prone combinations.

5.1. Stub Configuration Matrix (SC-matrix)

The stub plug-in concept in UCMs enables us to isolate specific scenarios of features from common scenarios. That is, the specific scenarios for a feature are given as a set of feature submaps, while the common scenarios are given as a root map with default submaps, into which the feature submaps are plugged. We can then characterize features in terms of *stub configurations*, i.e. information regarding which feature submap is plugged into which stub in the root map. In this section, we propose a matrix representation, called a *stub configuration matrix* (SC-matrix), to characterize features.

Definition 5.1 Let SM denote the set of all given submaps. A *matrix element* is recursively defined as follows: (a) $f \in SM$ is a matrix element, (b) if p and q are matrix elements consisting of submaps plugged into the same stub, then $p|q$ are matrix elements, where $|$ is a *deterministic choice operator*.

The matrix elements are regarded as expressions in the language composed by submap identifiers and operator $|$. These elements are used to represent which submaps are plugged into each stub. A matrix element $f_1|f_2|...|f_k$ means that exactly one submap f_i of $f_1, ..., f_k$ is deterministically chosen and plugged into the stub, according to a certain selection policy. Consider again the UCMs in Figure 1 and 2. Then, for instance, def_2, sbB_2, rcB_2 and $def_2|sbB_2$ are all matrix elements.

Next, we express the configuration of all stubs in a root map, in terms of a vector representation, which intuitively describes a *subscriber profile*.

Definition 5.2 Suppose that a given root map has n stubs. A *stub configuration vector* (or simply SC-vector) is an n-dimensional vector $\boldsymbol{h} = [h_1, ..., h_n]$, where h_i is a matrix element for i-th stub.

Consider again all UCMs in Section 3. Let us briefly characterize A's scenarios for individual features, in terms of an SC-vector.

In this UCM, when the user does not subscribe to any feature, his/her scenario is characterized by an SC-vector:

$$[def_1, def_2, def_3, def_4, def_5, def_6, def_7]$$

This SC-vector means that, all of the default submaps are plugged into the stub of the root map.

First consider the stub configuration where A, who is the caller, subscribes to the *originating feature*; the feature that is activated only when A makes a call. Let us take CNDB as an example of the originating feature. In this case, submaps $cndbA_3$ is plugged into stub 3, and all other stub i contain the default submap def_i. Hence, A's scenarios are characterized by an SC-vector:

$$[def_1, def_2, cndbA_3, def_4, def_5, def_6, def_7]$$

Next, consider the stub configuration where B, who is the callee, subscribes to the *terminating feature*; the feature that is activated when B accepts a call. Let us take TCS as an example of the terminating feature. When B subscribes to TCS, then the caller's scenario is characterized by:

$$[def_1, def_2, def_3, def_4|tcsB_4, def_5, def_6, def_7]$$

where the inclusion of $tcsB_4$ or def_4 is determined by whether A calls B or another user who does not subscribe to TCS.

Thus, the individual features on A's scenarios are concisely characterized by SC-vectors.

In order to represent clearly all possible user scenarios, we replicate the root map for each user's scenarios, as shown in Figure 4. The replication of the root map makes sense, since common scenarios described in the root map are the same for all users, due to the "equivalently-served" constraint [25][4] in telecommunication services. The stub configuration describes the allocation of feature submaps to stub in the root maps of all users. Accordingly, the SC-vector is extended to a matrix form, called the SC-matrix.

Definition 5.3 Suppose that a given root map has n stubs, and that we have m users. A *stub configuration matrix* (or simply SC-matrix) is an $m \times n$-dimensional matrix:

$$H = \begin{bmatrix} \boldsymbol{h_1} \\ \boldsymbol{h_2} \\ \vdots \\ \boldsymbol{h_m} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ h_{m1} & h_{m2} & \cdots & h_{mn} \end{bmatrix}$$

where $\boldsymbol{h_i}$ is an SC-vector for the i-th user, and h_{ij} is a matrix element for the j-th stub in the i-th user's root map. Usually an SC-matrix for a feature is specified on the basis of the feature named F and of its subscriber u. For convenience, we introduce a notation F_u to denote an SC-matrix where the user u subscribes to feature F.

For example, consider all submaps in Figure 1(b) and Figure 2, and root maps in Figure 1(a). Here we suppose that there are three users, A, B and C, as introduced in [21]. Since the root map has seven stubs, the SC-matrix is a 3×7 matrix. Let us express the stub configuration where A is a CNDB subscriber. Submaps $cndbA_3$ is plugged into stubs 3 in A's root map. Similarly we can describe the stub configuration that B is a CNDB subscriber just by swapping A and B.

$$CNDB_A = \begin{bmatrix} def_1 & def_2 & cndbA_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \end{bmatrix}$$

[4] Suppose that A and B subscribe to the same feature F. Therefore, A and B are guaranteed to be able to use F in the same way.

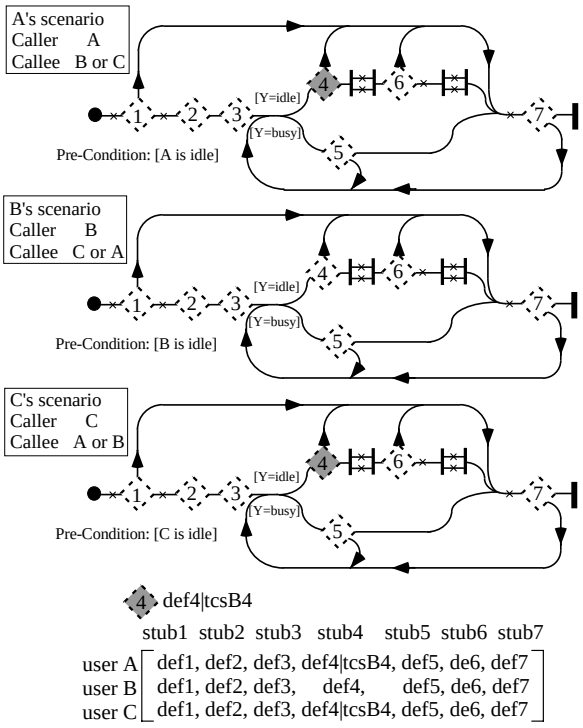


Figure 4. Extending a root map for three users (when B subscribes to TCS)

$$CNDB_B =$$

$$\begin{bmatrix} def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & cndbB_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \end{bmatrix}$$

By looking at the matrix row-wise, we can visualize how each user's scenario is configured, under a certain feature subscription. Next, let us give an SC-matrix, TCS_B:

$$TCS_B =$$

$$\begin{bmatrix} def_1 & def_2 & def_3 & def_4|tcsB_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4|tcsB_4 & def_5 & def_6 & def_7 \end{bmatrix}$$

Note that B's subscription to TCS affects call scenarios of both A and C, since TCS applies when A (or C) calls B (i.e. the condition $[Y = B]$ holds). Here, we do not consider the case where B calls him/herself. Therefore, B follows the default scenario. Figure 4 shows a correspondence between the root maps and the SC-matrix TCS_B. The shaded stubs represent the submaps for TCS, which are plugged into those stubs.

One useful guideline for systematically constructing an SC-matrix is to classify the features into two categories: *originating features* or *terminating features*. The originating features are the features whose subscriber is on the caller side, while terminating features are the features whose subscriber is on callee side. In our example, CNDB is the originating feature, whereas TCS, SB, RC and CFB are the terminating features [21]. Note that the root map in our example is described from the caller's point of view. Subscribing to originating features affects the scenarios of only the subscriber. On the other hand, subscribing to terminating features affects the scenarios of all users, except the subscriber. Based on this observation, let us give SC-matrices RC_B and SB_B:

$$RC_B =$$

$$\begin{bmatrix} def_1 & def_2|rcB2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2|rcB2 & def_3 & def_4 & def_5 & def_6 & def_7 \end{bmatrix}$$

$$SB_B =$$

$$\begin{bmatrix} def_1 & def_2|sbB2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2|sbB2 & def_3 & def_4 & def_5 & def_6 & def_7 \end{bmatrix}$$

Note that it is possible to represent feature configurations for arbitrary subscribers in terms of SC-matrices. This representation is accomplished by instantiating feature submaps with a value of the subscriber and allocating such submaps to appropriate rows of the SC-matrix. For instance, RC_C can be obtained from RC_B by swapping the second and third rows, and by letting $V = C$ instead of $V = B$ in submap rcV_2.

5.2. Feature Combination

Once each individual feature is characterized by an SC-matrix, we combine different configurations, in order to examine FI filtering between multiple features. The combination is carried out by a well-defined SC-matrix combination as shown in this section. First, we define the combination operator for two submaps:

Definition 5.4 Suppose that f and g are given submaps, which can be plugged into the same stub in a root map. Let *def* denote any default submap describing the basic call scenarios. Let ng [5] denote a special identifier not contained in the given submaps. Then, a combination of f and g, denoted by $f \cdot g$, is defined as follows:

$$f \cdot g = g \cdot f =$$

$$\begin{cases} f & (\text{if } f = g) & (A1) \\ f & (\text{if } g = def) & (A2) \\ f|g & (\text{if } [f \neq g], [f, g \neq def] \text{ and } [mex(f, g)]) & (A3) \\ ng & (\text{if } [f \neq g], [f, g \neq def] \text{ and } [\neg mex(f, g)]) & (A4) \end{cases}$$

The intuitive semantics of the combination is explained as follows: (A1) a combination of the same submaps yields the same submap, (A2) a feature submap f can override a default map for the basic call scenario, (A3) two different feature submaps can be composed with a

[5] The ng here stands for "no good".

deterministic choice when f and g are mutually exclusive and (A4) two different feature submaps cannot be plugged into the same stub when f and g are not mutually exclusive, since a non-deterministic behavior arises between f and g.

Then, the combination operator is extended for matrix elements containing "|", by the following definition:

Definition 5.5 Let $p = f_1|f_2|...|f_k$ and $q = g_1|g_2|...|g_l$ be matrix elements. Then, the combination of p and q is defined by applying $\cdot$ to all pairs of submaps f_i and g_j:

$$p \cdot q = f_1 \cdot g_1|f_1 \cdot g_2|...|f_k \cdot g_l \quad (A5)$$

The following proposition is useful for simplifying the combination results.

Proposition 5.6 Let p, q and r be matrix elements. The following properties are satisfied: (B1) $p|p = p$, (B2) $p|q = q|p$, (B3) $(p|q)|r = p|(q|r)$, (B4) $p|ng = ng$.

For example, consider SC-matrices RC_B and SB_B in the previous section. Let us compose two matrix elements $def_2|rcB_2$ and $def_2|sbB_2$, with respect to stub 2.

$$(def_2|rcB_2) \cdot (def_2|sbB_2)$$

$$\begin{array}{lll} = & def_2 \cdot def_2|def_2 \cdot sbB_2|rcB_2 \cdot def_2|rcB_2 \cdot sbB_2 & (A5) \\ = & def_2|sbB_2|rcB_2|rcB_2 \cdot sbB_2 & (A2) \\ = & def_2|sbB_2|rcB_2|ng & (A4)^6 \\ = & ng & (B4) \end{array}$$

Here, we can define a combination operator of SC-matrices as:

Definition 5.7 Let F and G be given SC-matrices. Then, the combination of F and G, denoted by $F \oplus G$, is defined as $F \oplus G = [f_{ij}] \oplus [g_{ij}] = [f_{ij} \cdot g_{ij}]$

combination of two SC-matrices is carried out by applying $\cdot$ to each pair of corresponding matrix elements. For instance, let us compose TCS_B and $CNDB_A$ shown in the previous subsection 5.1.

$$TCS_B \oplus CNDB_A =$$

$$\left[\begin{array}{ccccccc} def_1 & def_2 & def_3 & def_4|tcsB_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4|tcsB_4 & def_5 & def_6 & def_7 \end{array}\right]$$

$$\oplus \left[\begin{array}{ccccccc} def_1 & def_2 & cndbA_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \end{array}\right]$$

$$=$$

$$\left[\begin{array}{ccccccc} def_1 & def_2 & cndbA_3 & def_4|tcsB_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 & def_7 \\ def_1 & def_2 & def_3 & def_4|tcsB_4 & def_5 & def_6 & def_7 \end{array}\right]$$

[6]Note that the precondition of rcB2 and sbB2 are both [Y=B], i.e. $\neg mex(rcB2, sbB2)$, see Figure 2(d) and (e), and replace V with B.

5.3. FI Filtering

We assume that a root map, a set of default submaps, sets of submaps for features, and SC-matrices for individual features are given.

Figure 6 depicts the correspondence between matrix combination and related root maps. The stubs depicted by shaded diamonds represent that some feature submaps are plugged into the stubs.

First, we provide two theorems used for the proposed FI filtering method. These theorems are derived from the FI characterizations (Condition C1 and C2) presented in Section 4. Let F and G be given SC-matrices, and let $H = F \oplus G$. Let $\boldsymbol{f_i}$, $\boldsymbol{g_i}$ and $\boldsymbol{h_i}$ be i-th rows in F, G and H, respectively.

Theorem 5.8 *If there exists ng in H, then FI occurs (non-determinism).*

Proof: By Definition 5.4, an ng entry appears in H iff a submap f_{ij} in F and a submap g_{ij} in G are not mutually exclusive. Since f_{ij} and g_{ij} are plugged into a stub j simultaneously, different scenarios are possible under the same (pre-)condition with respect to user i. According to Condition C2, we can conclude that FI occurs.

The example in Figure 6(a) illustrates the verdict (a) "FI occurs" for the combination of RC_B and SB_B. After the combination, an ng entry appears in stub 2 of A's and C's root map. This is a non-deterministic interaction: "Suppose that B subscribes to RC and SB. If A calls B, should the call be charged 100% to B by the function of RC, or should the call be charged by the rule of payment of SB, which B had set".

Theorem 5.9 *If $[\boldsymbol{h_i} = \boldsymbol{f_i}$ or $\boldsymbol{h_i} = \boldsymbol{g_i}]$ holds for all i, then FI does not occur.*

Proof: Each row in an SC-matrix is an SC-vector that characterizes one user's scenario. The condition $[\boldsymbol{h_i} = \boldsymbol{f_i}$ or $\boldsymbol{h_i} = \boldsymbol{g_i}]$ holds iff for user i, the stub configuration $\boldsymbol{h_i}$ yielded by the combination had been already expected in the individual feature F (=$\boldsymbol{f_i}$) or G (=$\boldsymbol{g_i}$). This fact means that no stub configuration is changed by the combination. Hence, no scenario change occurs with respect to the user i. If the condition $[\boldsymbol{h_i} = \boldsymbol{f_i}$ or $\boldsymbol{h_i} = \boldsymbol{g_i}]$ holds for all i, then no user's scenario is changed by the combination. According to a contraposition of Condition C1, we can conclude that FI never occurs.

The example in Figure 6(b) illustrates the verdict (b) "FI never occurs" between TCS_B and $CNDB_B$. The condition $[\boldsymbol{h_i} = \boldsymbol{f_i}$ or $\boldsymbol{h_i} = \boldsymbol{g_i}]$ holds for $i = 1, 2, 3$ (for users A,B,C). In this case, the scenarios of users A and C had been expected in TCS_B before the combination, whereas the scenarios of user B had been found in $CNDB_B$. As a result, no scenario change occurs, and thus we can conclude that there is no FI.

With the above theorems, we finally present the filtering method (Phase1) in Figure 5. The method gives one

Feature Interaction Filtering Method (First Phase)

Input : Stub configuration matrices

$$F = \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} \text{ and } G = \begin{bmatrix} g_1 \\ \vdots \\ g_m \end{bmatrix}$$

Output : One of the following verdicts
(a) FI occurs (non-determinism).
(b) FI never occurs.
(c) FI-prone.

Procedure :

Step1 : Make a composed matrix

$$H = \begin{bmatrix} h_1 \\ \vdots \\ h_m \end{bmatrix} = F \oplus G.$$

Step2 : If some ng elements exist in H, conclude (a) FI occurs (by Theorem 5.8). Otherwise, go to Step 3.

Step3 : For each row h_i of H, check a condition [$h_i = f_i$ or $h_i = g_i$].

Step3-1 : If the condition holds for all i ($1 \leq i \leq m$), conclude (b) FI never occurs (by Theorem 5.9). Otherwise,

Step3-2 : Conclude (c) FI-prone.

Figure 5. Feature Interaction filtering method

of the verdicts: (a) FI occurs (non-determinism), (b) FI never occurs, (c) FI-prone, for two given SC-matrices F and G.

Since Theorems 5.8 and 5.9 at Steps 2 and 3 can be checked easily, the filtering procedure is quite simple and easy to use. Step 1 is used simply for making a composed matrix H from F and G. Step 2 is used to check the non-determinism caused by the combination by Theorem 5.8. Step 3 is used for checking if any scenario changes occur due to the combination using Theorem 5.9. If we reach Step 3-2, this means that non-determinism does not exit, but some scenarios change in the combination. We cannot definitely conclude the existence of FI at this point. The verdict is "FI-prone" and some detection method has to be employed.

The example in Figure 6(c) is for the verdict (c) "FI-prone" between TCS_B and $CNDB_A$. Due to the combination, user A's scenarios have been changed, which can be interpreted as follows: "Suppose that B subscribes to TCS and sets A to the screening number, and that A subscribes to CNDB. Since A subscribes to CNDB, A's number may not be displayed on B's telephone. Therefore, the screening function of B may not work properly." Whether this is an FI or not depends on the exact definition of FI adopted in the subsequent detection process. The only thing we can say here regarding filtering process is that the system is FI-prone. Note that even if the feature combination is the same (as in Figure 6(b)), we can still get a different verdict depending on who the subscribers are.

The proposed filtering method is applied to all possible combinations of SC-matrices, derived from given features.

The number of combinations increases combinatorially with the number of users and features. However, the number of combinations can be reduced by using *symmetry*. For example, if we have analyzed a combination $TCS_B \oplus CNDB_A$, then we no longer need to try $TCS_B \oplus CNDB_C$, since all subscribers of a feature can use the feature in the same way. Due to space limitations, the detailed definition of symmetry is omitted here. Interested readers can refer to relevant papers [18, 25].

Note that we have only shown the example where one user subscribes to only one feature at a time. However, the proposed method can be used when one user subscribes to more than two features as well. Suppose that user A subscribes to feature f, g, and h. In this case, the combinations to be analyzed will be, $f \oplus g$, $g \oplus h$ and $f \oplus h$. Generally for n features, the number of the combinations will be ${}_nC_2$.

6. Phase 2: Deriving FI-prone Scenarios

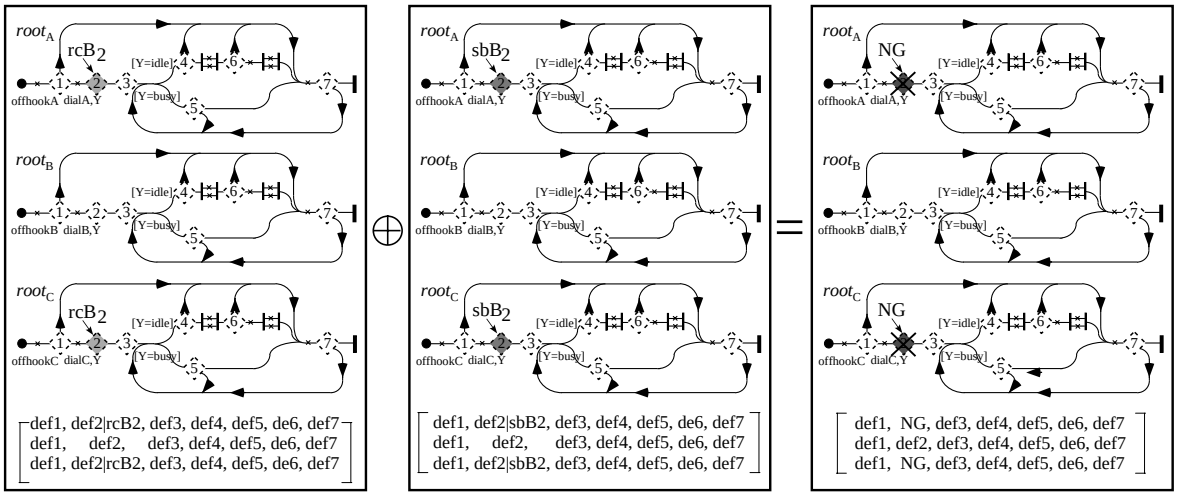
In the second phase, we derive FI-prone scenarios from the (c) FI-prone combinations. The FI-prone service combinations do not always cause actual FIs. To make the FI filtering more accurate, we derive the *scenarios* which may be relevant to actual FIs. For this, we propose two heuristics on the scenario paths.

6.1. Observations on FI-prone Scenarios

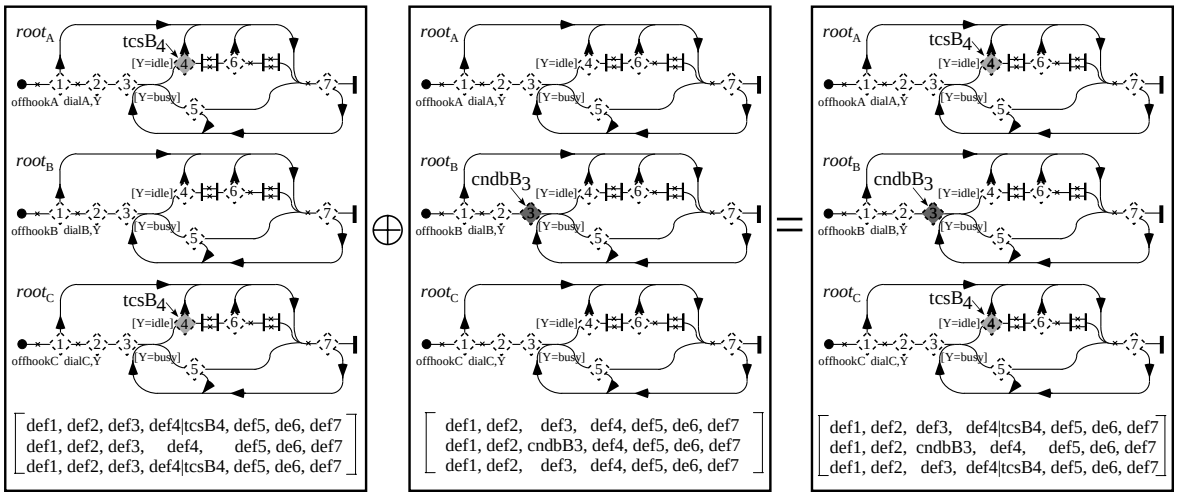
In the first phase of FI filtering, the verdict of FI-prone is derived by Condition C1 (in Section 4). That is, some user's scenarios in the root maps have been changed by the feature combination. What we have to consider next is how the scenarios change and which scenarios have the potential of FIs.

Let us consider again the examples FI-(b) and FI-(c) in Section 2. As shown in Figure 7(I), FI-(b) occurs in a scenario where B subscribes to TCS and B puts A's telephone number on the screening list, A subscribes to CNDB and calls B while B is idle. In this scenario, both TCS and CNDB services are activated.

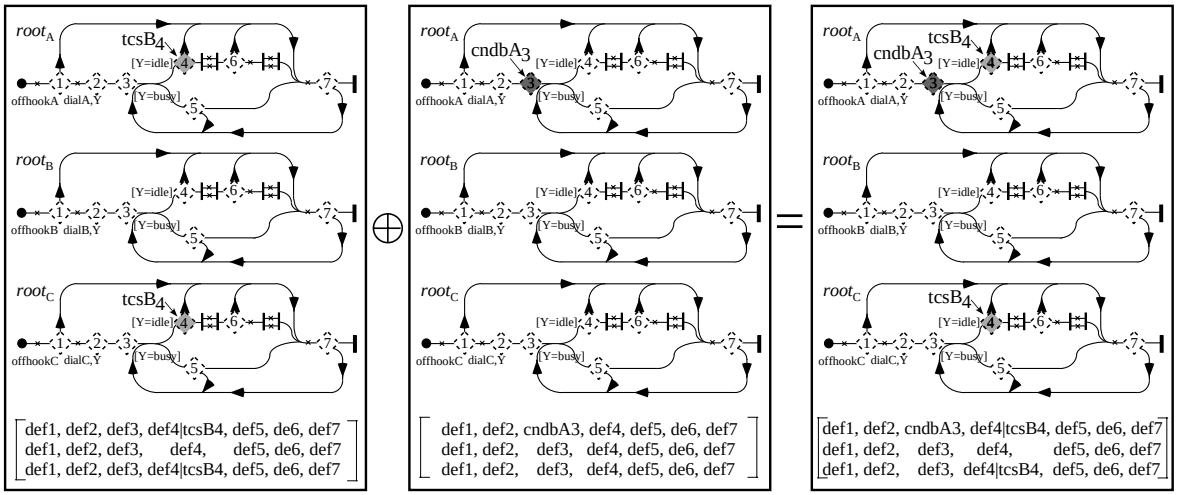
Next, FI-(c) occurs in a scenario where A subscribes to CFB and sets the forwarding address to B, and B subscribes to RC (see Figure 7(II)). If C calls A when A is



(a) $RC_B \oplus SB_B$ = FI occurs



(b) $TCS_B \oplus CNDB_B$ = FI never occurs



(c) $TCS_B \oplus CNDB_A$ = FI prone

Figure 6. Illustrative examples of FI Filtering

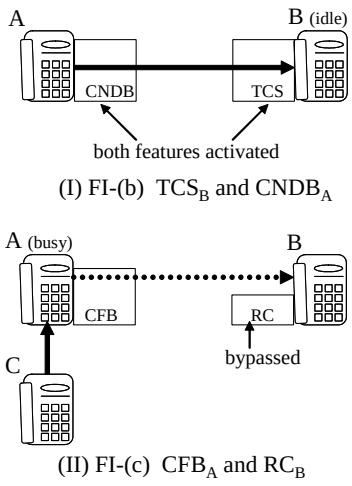


Figure 7. The example of FI-(b) and FI-(c)

busy, CFB of A is activated and A creates a new path to originate the new call to B directly without passing the RC feature of B. As a result, in this scenario, a feature of RC which allows the subscriber B to be charged for the calls, is not activated because of the activation of CFB of A. Thus, we can see that the activation of RC is bypassed by the activation of CFB. Examining many other practical examples, we have reached the following observations of FIs.

Observation 1: FI tends to occur in scenarios where two different features are *sequentially activated*.

Observation 2: FI tends to occur in scenarios where the execution of one feature *bypasses* the execution of the other feature.

The above two observations will be mapped into the two heuristics defined on the scenario paths of UCMs, in Section 6.4.

6.2. FI-prone Root Map

Each FI-prone combination obtained in Phase 1 has multiple root maps, each of which corresponds to a user. From them, we first pick up only root maps that contain problematic scenarios. Specifically, we only pick up the root maps whose stub configurations is changed before/after the service combination.

According to Condition C1 (see Section 4), the FI-prone scenarios must be contained in a root map in which the stub configuration is modified by a feature combination. For instance, let us take Figure 6(c). For this combination, FI-prone scenarios must be contained in $root_A$ only, but neither in $root_B$ nor $root_C$.

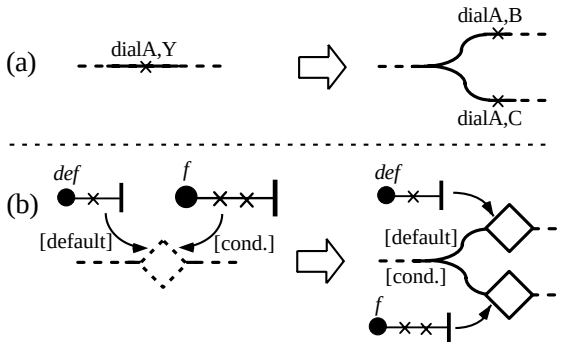


Figure 8. Expanding dynamic elements

6.3. Expansion of Dynamic Elements in Scenarios

If a root map contains *dynamic elements* such as variables and dynamic stubs, a scenario path can represent multiple scenarios dynamically depending on run-time conditions (see Section 3). In such a situation, we have to consider both the scenario path structures and the run-time conditions, simultaneously.

To avoid confusion, we eliminate the dynamic elements by expanding them into static ones, before deriving FI-prone scenarios. In the resulting root map, a scenario path exactly corresponds to a concrete scenario. Therefore, we can only concentrate on the path structure to derive FI-prone scenarios. The elimination of the dynamic elements is performed by replacing the dynamic elements with branches (fork/join) for all possible conditions, which is specifically described below.

For responsibilities with variables, we use a fork to describe a possible branch with respect to the range of the variable. For example, Figure 8(a) shows the case of a responsibility "dialAY", where callee Y is a variable. Assume that the range of Y includes B and C. Thus, two subsequent scenarios are possible, where A calls B or A calls C. Therefore, "dialAY" is expanded into two (static) responsibilities "dialAB" and "dialAC".

As mentioned before, the dynamic stub can have multiple submaps to be plugged into. The selection of the submaps is determined at run time by a selection policy that is usually specified in the pre-conditions of the submaps. The dynamic stub can be also expanded into a branch of the static stubs (denoted by solid diamonds). Let us examine Figure 8(b). In the figure, two submaps def and f can be plugged into the dynamic stub, thus allowing two possible scenarios. Therefore, we expand the scenario into two using a branch with guards (taken from the pre-conditions of the submaps). Thus, for each scenario, we place a static stub in which submap f or def is plugged. In addition, for the submaps that have post-conditions modifying conditions in a scenario, merging

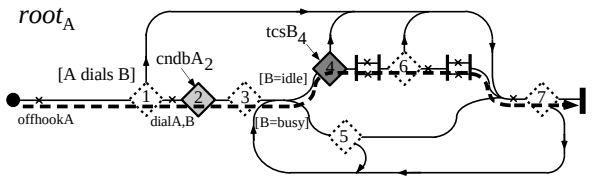


Figure 9. Heuristic H1

some expanded scenarios might be necessary. For this situation, we use a join.

The root map in which all dynamic elements are eliminated is called an *expanded root map*.

6.4. Deriving FI-prone Scenarios

After eliminating the dynamic elements on the root map, here we derive FI-prone scenarios in the suspected root map. Among all scenario paths in the *expanded root map*, a feature is activated in a path passing through a static stub with the feature submap. According to Observations 1 and 2 in Section 6.1, we propose the following heuristics H1 and H2 for deriving FI-prone scenario paths from the expanded root map.

Definition 6.1 Let f and g be feature submaps of features F and G, respectively. Then, derive any scenarios on the expanded root map based on the followings:

Heuristic H1: Derive a scenario path that passes through both f and g.

Heuristic H2: Derive a scenario path in which f is bypassed by g, and vice versa.

Figure 9 illustrates Heuristic H1, which derives an FI-prone scenario between $CNDB_A$ and TCS_B. The scenario activates CNDB and TCS sequentially as explained in Figure 7 (I).

On the other hand, Figure 10 describes Heuristic H2, where a function of RC is bypassed by CFB. In the scenario, an FI between CFB_A and RC_B occurs as explained in Figure 7 (II). Note in the figure that the root map $root_C$ is expanded into two cases (a) C dials A or (b) C dials B. The dotted line shows a scenario; after C dials A, the call is forwarded to B, but rcB_2 is bypassed.

Here we should note that the derived scenarios *do not* guarantee the existence of actual FIs. That is, Heuristics H1 and H2 derive only *FI-prone* scenarios. Hence, not all derived scenarios contain FIs, and some of these scenarios might even be FI-free. The exact FIs will be detected in the FI detection process, which is the next step of FI filtering. The goal of the proposed method is to provide the FI-prone scenarios as essential information for efficient FI detection.

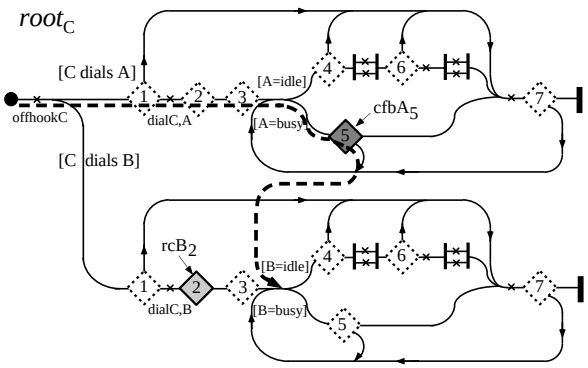


Figure 10. Heuristic H2

7. Evaluation

7.1. Preliminary

We have applied the proposed method to the specifications of the eight features taken from the second FI detection contest [21]. The features include: (1) Call Forwarding on Busy (CFB), (2) Teen Line (TL), (3) Terminating Call Screening (TCS), (4) Reverse Charge (RC), (5) Call Number Delivery Blocking (CNDB), (6) Ring Back when Free (RBF), (7) Voice Mail (VM), and (8) Split Billing (SB).

Since the contest specifications are given by Communicating Finite State Machines (CFSMs), we first construct the UCMs (root map and default/feature submaps), so that the causal relationships among events are preserved. The UCMs consist of a root map and default submaps as shown in Figure 1, and feature submaps as shown in Figure 2.

We combined each pair among the eight features in the following two ways: (A) both features are allocated to the same user, and (B) two users subscribe to different features. Next, for each combination, we applied the first phase of the proposed FI filtering method. Finally, for combinations with (c) FI-prone, we applied the second phase to derive FI-prone scenarios.

The evaluation is conducted from the following viewpoints for both filtering phases:

Filtering quality: To check the quality of the proposed filtering method, we evaluate the *filtering quality* that can be defined by the number of feature combinations filtered at the FI filtering process.

Scenario coverage: We evaluate the *scenario coverage* to check whether the scenarios derived by proposed filtering method cover actual FIs scenarios or not.

Reduction ratio: From the viewpoint of cost reduction, we evaluate the *reduction ratio*, which represents the percentage of the irrelevant scenarios filtered by the filtering process.

Table 1. Filtering result in the first phase

CFB		TL		TCS		RC		CNDB		RBF		VM		SB		
same	diff	same	diff	same	diff	same	diff	same	diff	same	diff	same	diff	same	diff	
	(c)	(b)	(b)	(c)	(c)	(c)	(c)	(b)	(c)	(a)	(c)	(c)	(c)	(c)	(c)	CFB
			(b)	(b)	(c)	(b)	(c)	(c)	(b)	(c)	(c)	(c)	(c)	(b)	(c)	TL
					(c)	(c)	(c)	(b)	(c)	(c)	(c)	(c)	(c)	(c)	(c)	TCS
							(c)	(b)	(c)	(c)	(c)	(c)	(c)	(a)	(c)	RC
									(b)	(c)	(c)	(c)	(c)	(b)	(c)	CNDB
											(c)	(c)	(c)	(c)	(c)	RBF
													(c)	(c)	(c)	VM
															(c)	SB

Table 2. Filtering result in the second phase (scenario derivation)

CFB		TL		TCS		RC		CNDB		RBF		VM		SB		
same	diff	same	diff	same	diff	same	diff	same	diff	same	diff	same	diff	same	diff	
	H1	(b)	(b)	Other	H1	H1	H2	(b)	H1	(a)	H1	Other	H1	H1	H2	CFB
			(b)	(b)	H1	(b)	H1	H1	(b)	H1	H1	H2	H1	(b)	H1	TL
					Other	H1	H1	(b)	H1	Other	H1	H2	H1	H1	Other	TCS
							Other	(b)	H1	H1	H1,H2	H1	H1	(a)	Other	RC
									(b)	H1	H1	H1	H1	(b)	H1	CNDB
											H1	H1	H1	H1	H1,H2	RBF
													H1	H1	H1	VM
															Other	SB

As a reference of the actual FIs among the eight services, we used FI detection results submitted by the team of Ottawa University [26].

To justify the effectiveness of the proposed filtering, we should compare the proposed method with other filtering methods. However, although several FI filtering method have been proposed [14, 18, 20], none of them conducts quality evaluation for practical settings. Therefore, carrying out comparative evaluation on filtering quality is impossible.

7.2. Filtering Quality

We want to see how many service combinations can be filtered by the proposed method. In other words, we examine how many combinations have a definite answer; (a) FI occurs or (c) FI never occurs. We define a metric *filtering quality* as; *(# of combinations with verdicts (a) or (b)) / (total # of combinations)*.

Table 1 shows the filtering result obtained by the first phase only. Table 2 shows the filtering result with both first and second phases.

Each entry of the tables represent one of the verdicts: (a) FI occurs, (b) FI never occurs or (c) FI-prone. The *same* (or *diff*) represents two services allocated to the same users (or different users, respectively), as mentioned in Section 7.1. The shaded entries represent the combinations that cause actual FIs detected in [26].

Table 1 shows that all combinations causing FIs are covered by the verdicts (a) or (c). For example, the combination of TCS_B and $CNDB_A$ (in subsections 2.2 and 5.3) has the verdict of (c) FI-prone in the entry *diff* of CNDB&TCS. Note that most combinations have the verdict (c), and no concrete scenario is available at this time. The number of combinations that have a definite answer, i.e., (a) or (b), is 14, and 50 FI-prone combinations still have to be examined at the subsequent FI detection process. Therefore the filtering quality of the first phase of the proposed method is 22.9% (=14/64).

On the other hand, in Table 2, for the combinations with (c), concrete FI-prone scenarios are derived by Heuristics H1 and/or H2. $H1$ (and $H2$) in the table represents the combination with a scenario derived by Heuristic H1 (and H2, respectively) [7]. $Other$ means that no scenario has been derived by neither H1 nor H2.

The table shows that scenarios in $Other$ do not cause actual FIs. Therefore, if we conclude that the combinations with $Other$ to be FI-free, then more combinations can be filtered at the filtering process. Since the number of combinations with (a), (b) or $Other$ is 22, the filtering quality is improved to 34.4% ($= 22/64$) with the information of concrete FI-prone scenarios.

[7] Sometimes, however multiple scenarios are derived from a combination.

7.3. Scenario Coverage

Next, we conduct a scenario-wise investigation to check whether the derived scenarios surely *cover* actual FIs or not. Here we define a metric *scenario coverage* as follows. Let n_d be the number of actual FI scenarios contained in the derived scenarios, and let n_t be the total number of actual FI scenarios. Then, the scenario coverage is defined by n_d/n_t.

Table 3 shows the number of scenarios derived from FI-prone combinations that were represented by (c) in Table 1. Among the total 74 scenarios investigated, 48 scenarios were derived by Heuristic H1, while 6 scenarios were derived by Heuristic H2. 20 scenarios matched neither H1 nor H2. Out of the 25 scenarios containing actual FI identified in [26] (thus,n_t = 25), 20 were from the scenarios derived by H1, and the remaining 5 scenarios were from the scenarios derived by H2.

For example, as mentioned in subsection 6, FI-(b) TCS_B and $CNDB_A$ was derived by H1, while FI-(c) RC_B and CFB_A was derived by H2.

From this result, it can be seen that all FI scenarios are contained in the FI-prone scenarios derived by Heuristics H1 and H2. None of the FI scenarios belong to the set $Other$. In this experiment, the proposed method achieves 100% coverage (thus, $n_d = 20 + 5 = 25$ and $n_t = 25$). Hence, it can be said that Heuristics 1 and 2 sufficiently cover actual FI scenarios in the experiment.

7.4. Reduction Ratio

The proposed method (phase 2) derives only FI-prone scenarios. In other words, it excludes (or filters) many scenarios that are *irrelevant* to the FI analysis, which significantly reduces the cost of scenario analysis. Our interest here is to evaluate how many such irrelevant scenarios can be filtered by the proposed method. We define a metric *reduction ratio* as follows: Let m_d be the number of the derived scenarios, and let m_t be the total number of existing scenarios in all combinations. Then, the number of the irrelevant scenarios is $m_t - m_d$. Thus, the reduction ratio is defined by $(m_t - m_d)/m_t$.

As shown in Table 3, 48 scenarios were derived by Heuristic H1 and 6 scenarios were derived by Heuristic H2. Then, the total number of the scenarios that were derived by Heuristic H1 and H2 were 54 (m_d = 48 + 6 = 54). On the other hand, the number of all scenarios existing in all combinations of the feature are 715 (m_t = 715). Thus, the percentage of the scenario reduction is 92.44% ($(m_t - m_d)/m_t = 715 - 54/715$).

This means that, instead of applying the FI-detection to all existing 715 scenarios, we can remove more than 90% of them and apply the FI-detection to only 54 scenarios that were derived by Heuristic H1 and H2. In other words, we can also say that in this case study; the proposed method could efficiently reduce more than 90% of the cost for FI-detection which is the next process after FI-filtering.

Table 3. Table of the result of scenario coverage

Filtering Method	The number of scenarios derived from FI-prone combinations	The number of scenario containing actual FI
Derived by H1	48	20
Derived by H2	6	5
Derived by neither H1nor H2	20	0
Total	74	25

Since the proposed filtering is a low-cost method, which is performed just by visually investigating whether or not the scenario satisfies two heuristics. Hence, it can be said that overall, the proposed method is expected to reduce considerable cost for FI detection process with a small amount of cost.

8. Conclusion

In this paper, we have proposed a two-phase FI filtering method based on UCMs. In the first phase, using the stub plug-in concept of UCMs, we characterize each feature in terms of its stub configuration. The stub configuration represented by the SC-matrix introduced a notational convention that is useful for representing features with UCMs. The different stub configurations are composed by means of a matrix combination. Basically, the matrix combination is performed by checking only the pre-conditions of the feature submaps, which are independent of the detail of the submaps. Thus, the proposed method is easy to use and scalable.

In the second phase, we have proposed a method to derive the FI-prone scenarios. Based on the two heuristics, the second phase derives FI-prone scenarios from root maps of FI-prone combinations.

The experimental evaluation through the FI detection contest showed that the derived scenarios successfully covered all scenarios of the actual FIs. Also, the combination did not cause actual FIs when there is no FI-prone scenario derived by the heuristics from a FI-prone combination. This fact implies that the heuristics are quite reasonable and they can improve filtering quality. Furthermore, the experimental evaluation showed that the proposed method could effectively filter 90% of the irrelevant scenarios, which implies a significant cost reduction of the scenario investigation.

Our future work is summarized as follows. We are currently investigating an efficient framework to use the de-

rived FI-prone scenarios in the FI detection process (e.g., test case generation, etc). Also, we plan to apply the proposed method to more services and features, which may reveal more effective heuristics for FI filtering.

Acknowledgments: This research was supported partly by a Grant-in-Aid for JSPS Fellow (No.100987) from the Ministry of Education Japan.

References

[1] A. Aho, S. Gallangher, N. Griffeth, C. Schell and D. Swayne, "$SCF3^{TM}$/Sculptor with Chisel: Requirements Engineering for Communications Services", *Proc. of Fifth Int'l. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98)*, pp.45-63, Oct. 1998.

[2] D. Amyot, L. Logrippo, R.J.A. Buhr and T. Gray, "Use Case Maps for the capture and validation of distributed systems requirements", *Proc. of Fourth Int'l Symposium on Requirements Engineering (RE'99)*, pp.44-53, June 1999.

[3] R. Boumezbeur, L. Logrippo, "Specifying Telephone Systems in LOTOS", *IEEE Communications Magazine*, Vol. 31, No. 8, pp.38-45, August 1993.

[4] R.J.A. Buhr, "Use Case Maps as architectural entities for complex systems", *IEEE Transactions on Software Engineering*, Vol.24, No.12, pp.1131-1155, 1998.

[5] E.J. Cameron, and H. Veithuijsen, "Feature interactions in telecommunications systems", *IEEE Communication Magazine*, Vol.31, No.8, pp.18-23, 1993.

[6] E.J. Cameron, K. Cheng, F-J. Lin, H. Liu, and B. Pinheiro, "A formal AIN service creation, feature interactions analysis and management environment: An industrial application", *Proc. of Fourth Int'l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW'97)*, pp.342-346, June 1997.

[7] C. Capellmann, P. Combes, J. Pettersson, B. Renard, and J.L. Ruiz, "Consistent interaction detection - A comprehensive approach integrated with service creation", *Proc. of Fourth Int'l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW'97)*, pp.183-197, June 1997.

[8] A. Daniel, "Use Case Maps Navigator", `http://www.usecasemaps.org/tools/ucmnav/index.shtml`

[9] M. Faci, L. Logrippo, "Specifying Features and Analyzing Their Interactions in a LOTOS Environment", *Feature Interactions in Telecommunications Systems*, L. G. Bouma, H. Velthuijsen (Eds.), IOS Press, pp.136-151, 1994.

[10] A. Gammelgaard and E.J. Kristensen, "Interaction detection, a logical approach", *Proc. of Second Int'l. Workshop on Feature Interactions in Telecommunications Systems (FIW'94)*, pp.178-196, 1994.

[11] A. Grinberg, "Seamless Networks: Interoperating Wireless and Wireline Networks", *Addison-Wesley*, 1996.

[12] R. J. Hall, "Feature Interactions in Electronic mail", *Proc. of Sixth Int'l. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'00)*, pp.67-82, May 2000.

[13] Y. Harada, Y. Hirakawa, T. Takenaka and N. Terashima, "A conflict detection support method for telecommunication service descriptions", *IEICE Trans. Comm.*, Vol.E75-B, No.10, pp.986-997, October 1992.

[14] M. Heisel and J. Souquieres, "A heuristic approach to detect feature interactions in requirements", *Proc. of Fifth Int'l. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98)*, pp.165-171, Oct. 1998.

[15] Y. Hirakawa and T. Takenaka, "Telecommunication service description using state transition rules", *Proc. of IEEE lnt'l Workshop on Software Specification and Design*, pp.140-147, October 1991.

[16] Y. Kawarazaki and T. Ohta, "New Proposal for Feature lnteraction Detection and Elimination", *Proc. of Third Int'l. Workshop on Feature Interactions in Telecommunications Systems (FIW'95)*, pp.127-139, Oct. 1995.

[17] D.O. Keck and P.J. Kuehn, "The feature interaction problem in telecommunications systems: A survey", *IEEE Trans. on Software Engineering*, Vol.24, No.10, pp.779-796, 1998.

[18] D.O. Keck, "A tool for the identification of interaction-prone call scenarios", *Proc. of Fifth Int'l. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98)*, pp.276-290, Oct. 1998.

[19] A. Khoumsi, "Detection and resolution of interactions between services of telephone networks", *Proc. of Fourth Int'l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW'97)*, pp.78-92, June 1997.

[20] K. Kimbler, "Addressing the interaction problem at the enterprise level", *Proc. of Fourth Int'l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW'97)*, pp.13-22, June 1997.

[21] M. Kolberg, E.H. Magill, D. Maples and S. Reiff, "Second Feature Interaction Contest", *Proc. of Sixth Int'l. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'00)*, pp.293-310, May 2000.

[22] Y. Nagatake, H. Sakai, T. Nohara and K. Takami, "An advanced IN control architecture for providing VoIP Supplementary Services", *Technical report of IEICE. ISSE2000-42*, pp.1-6, June 2000.

[23] M. Nakamura, Y. Kakuda, and T. Kikuno, "Analyzing non-determinism in telecommunication services using P-invariant of Petri-Net model", *Proc. of IEEE INFOCOM'97*, April 1997.

[24] M. Nakamura, Y. Kakuda and T. Kikuno, "Petri net based detection method for non-deterministic feature interactions and its experimental evaluation", *Proc. of Fourth Int'l. Workshop on Feature Interactions in Telecommunication Networks and Distributed Systems (FIW'97)*, pp.138-152, June 1997.

[25] M. Nakamura, and T. Kikuno, “Exploiting symmetric relation for efficient feature interaction detection”, *IEICE Trans. on Information and Systems*, Vol.E82-D, No.10, pp.1352-1363, 1999.

[26] M. Nakamura, T. Ding, J. Sincennes, X. Lu and L. Logrippo, “Second Feature Interaction Contest - Contest Report”, *Proc. of Sixth Int’l. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW’00)*, pp.314-317, May 2000.

[27] T. Ohta and Y. Harada, “Classification, detection and resolution of service interactions in telecommunication services”, *Proc. of Second Int’l. Workshop on Feature Interactions in Telecommunications Systems (FIW’94)*, pp.60-72, 1994.

[28] B. Renard, P. Combes, F. Olsen, “An SDL/MSC Environment for Service Interaction Analysis”, ICIN, Bordeaux, November 1996.

[29] M., Weiss, “Feature Interactions in Web services”, *Proc. of Seventh Int’l. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW’03)*, pp.149-156, June 2003.

[30] T. Yoneda and T. Ohta, “A formal approach for definition and detection of feature interactions”, *Proc. of Fifth Int’l. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW’98)*, pp.165-171, Oct. 1998.

[31] P. Zave, “Feature interactions and formal specifications in telecommunications”, *IEEE Computer*, Vol.26, No.8, pp.20-30, 1993.

[32] Bellcore, “Advanced Intelligent Network (AlN) Release 1, Switching Systems Generic Requirements”, *Bellcore Technical Advisory TA-NWT-OO1123* (1991)

[33] Feature Interaction in Telecommunications, Vol. I-VII, IOS Press (1992-2003)

[34] W3C, “Web Services Activity”, 2004, `http://www.w3.org/2002/ws/`