Feature Interaction Filtering with Use Case Maps at Requirements Stage

Masahide Nakamura, Tohru Kikuno
Department of Informatics and Mathematical Sciences, Osaka University, JAPAN
{*masa-n,kikuno*}*@ics.es.osaka-u.ac.jp*

Jameleddine Hassine, Luigi Logrippo
School of Information Technology and Engineering, University of Ottawa, CANADA
{*jhassine, luigi*}*@site.uottawa.ca*

Abstract. This paper proposes a new Feature Interaction (FI) filtering method at the requirements level. FI filtering is to screen out some irrelevant feature combinations before the FI detection process, by using simple indications that certain combinations are likely to cause FI.

To achieve this, we extensively utilize a requirement notation method, called Use Case Maps (UCMs), which helps designers to visualize a global picture of call scenarios. Individual features are added to the basic call by using the stub plug-in concept of UCMs. That is, a set of UCMs describing the feature's functionality are plugged into certain parts (called stubs) of the basic call scenario in a "root" UCM. Thus, each feature is characterized by the stub configuration. Then, we propose a composition method of different stub configurations in terms of a matrix, called a stub configuration matrix. Finally we present an FI filtering method for a given combination, which gives a verdict: (a) FI occurs, (b) FI never occurs or (c) FI-prone.

Experimental evaluation using examples taken from the recent FI detection contest shows that almost half of the feature combinations can be filtered without costly feature analysis. This suggests that the number of feature combinations to be analyzed with more expensive FI detection methods can be reduced to half, by using the proposed method as a front end of the detection process.

1 Introduction

In order to exactly detect all Feature Interactions (FIs, in short) in given features, the FI detection process requires sophisticated methods. A number of detection methods have been proposed so far (see survey [7]). It is known, however, that most of the detection algorithms need a significant amount of work due to the need of considering all possible combinations of behaviors. To make matters worse, the inclusion of new features increases further the number of feature combinations to be analyzed. Thus, FI detection can be an expensive and even infeasible task.

Therefore, it would be helpful to have a method that can be used before FI detection to estimate briefly "which feature combinations have a possibility of FI". Based on this motivation, Kimbler[9] proposed a notion of *FI filtering*. The aim of FI filtering is to screen out some irrelevant feature combinations before the detection process. In general, the filtering is performed in an inexpensive way by using simple indications that certain feature combinations *are likely to cause* FI or not. Since FI filtering enables us to narrow down the scope of FI detection to only FI-prone combinations, the cost in the

detection process can be significantly reduced by using filtering as a front end of the detection process.

In this paper, we propose a new FI filtering method at the requirements stage. To achieve this, we first employ a requirements notation method, called Use Case Maps (UCMs, in short) [2, 3], for the service description. The UCMs provide system-wide path structures that enable requirements engineers to get a global understanding of large scale dynamic situations.

Many service description methods have been proposed so far. Aho et al. [1] proposed an event-based language at the requirements level, called Chisel. Its well-defined semantics based on trace equivalence provides an FI detection framework. However, due to the nature of the method, it is difficult to represent concurrent behaviors. Moreover, feature addition is performed by "glueing nodes" in the Chisel diagram, which results in difficulty to achieve global visualization in one picture. The other well-known methods, such as SDL, LOTOS [2], State Transition Rules (STR) [13] are categorized at the software specification level rather than at the requirements level [1]. Hence, they need some sophisticated requirements elicitation procedures. The reasons why we chose UCMs for the FI filtering are summarized as follows:

(1) UCMs help to visualize global call scenarios at the requirements level without knowing the detailed system behaviors or complex semantic models.

(2) UCMs possess adequate characteristics for service description at the requirements level such as concurrency, alternatives and hierarchical design.

(3) There is a tool called UCM Navigator [10], which helps designers to draw syntactically correct UCMs.

The key idea for the FI filtering is to use one of the concepts of UCMs, called a *stub plug-in*. We have two kinds of UCMs in this framework. One is a root UCM (or simply root map), which specifies the scenario path structure commonly used by all features in the system. The other is the sub UCM (or simply submap), which represents functionality specific to each individual feature. Once a feature is specified by a collection of submaps, these submaps are plugged in certain parts of the root map, called *stubs*, to complete the call scenarios. Thus, a feature is characterized by its *stub configuration*, i.e. which submap is plugged in which stub in the root map.

Next, we propose a composition method for different stub configurations in order to examine FI filtering between different features. For this, we define a new matrix representation, called a *stub configuration matrix*. Feature composition is carried out with the matrix composition. Then, we present a filtering method. Using simple indications with respect to user's call scenarios, the method not only screens out some irrelevant combinations free of FIs, but also identifies feature combinations causing nondeterministic FIs [12]. This enables us to reach one of the following verdicts: (a) FI occurs, (b) FI never occurs or (c) FI-prone.

We conducted an experimental evaluation by using the features of the recent FI contest [4]. The experiments are carried out by using a tool, the UCM Navigator. It is shown that almost half of all feature combinations can be filtered in an inexpensive way. This fact implies that the number of feature combinations to be analyzed in the FI detection process can be significantly reduced by half, by using the proposed method as a front end of the detection process.

2 Use Case Maps for Service Description

2.1 Basic Principle

UCMs[1] describe the structure of behavior for a system based on scenario paths, in diagrams that are above the detailed level of messages and protocols. Figure 1 represents UCMs for the Basic Call Model, (or POTS — Plain Ordinary Telephone Service), according to the first FI detection contest specifications [4]. There are seven UCMs in Figure 1 and each is identified by a name (*identifier*), e.g. *root*, def_3.

The core notation consists of only *scenario paths* and *responsibilities* along the paths. In the diagram, the scenario paths are represented by wiggly lines. A path starts at a *starting point* (depicted by a filled circle) and ends at an *end point* (shown as a bar). 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 start to end is to explain a scenario as a causal sequence of events. For instance, take UCM def_4 in Figure 1(b). This explains a scenario where *busytoneA* occurs first and then *onhookA* is performed. 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 is idle]" in *root* means that the scenario starts only when A is idle.

Several paths can be composed by superimposing common parts and introducing *fork* and *join*. There are two kinds of forks/joins. One is the *OR-fork/join*, depicted by branches on paths. It describes *alternative* scenario paths, which mean that one of the paths is selected to proceed at each branch. For example, UCM def_1 in Figure 1(b) contains an OR-fork describing two possible scenarios: "*dialtoneA* occurs and the scenario ends at $OUT11$" or "*dialtoneA* occurs, followed by *onhookA*". It is also possible to explicitly specify conditions for path selection. This is done by using *guards*, represented by italic text with brackets [] at the OR-fork. For example, a guard *[Y is idle]* at an OR-fork in UCM *root* implies that the scenario proceeds to the upper path when Y is idle.

The other types of fork/join is the *AND-fork/join*, depicted by branches with bars, which describes *concurrent* scenario paths. UCM def_5 contains one AND-fork and one AND-join. After *offhookY* occurs, three scenario paths start concurrently. As a result, *LogBeginAYA*, *StopARingAY* and *StopRingYA* are performed in any order (by the interleaving semantics). The concurrent paths are synchronized at the AND-join, and then the scenario ends at $OUT51$.

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 that 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 just 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 entries and exits of the stub in the root map, respectively, in accordance with labels on the start and end points.

For example, the root map in Figure 1(a) contains six stubs. All other UCMs are

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

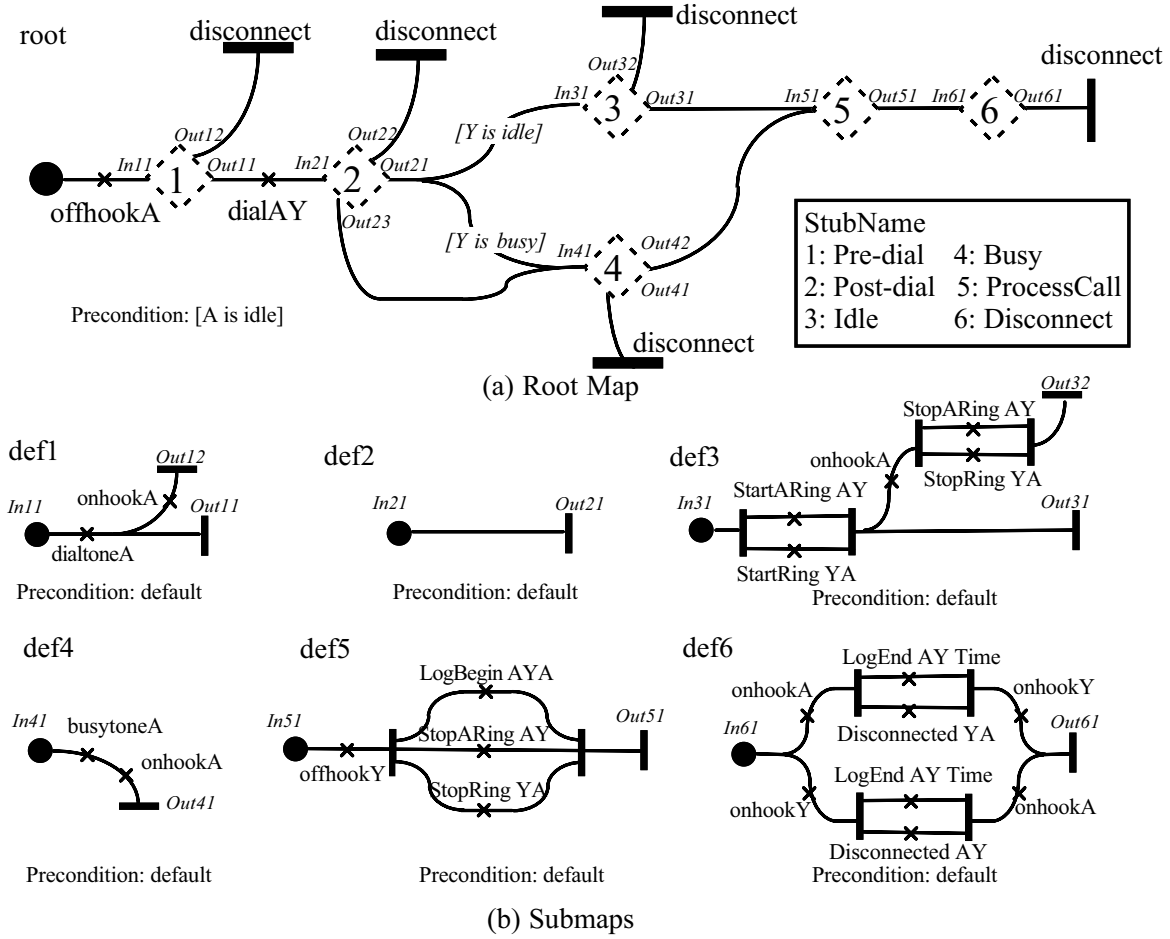


Figure 1: Use Case Maps for Basic Call Model

submaps. Let us plug a submap def_1 into stub 1 (Pre-dial stub) in *root*. The start 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 $def_i (2 \leq i \leq 6)$ are plugged into the corresponding stubs $i (2 \leq i \leq 6)$, which completes the whole scenario path structure of the basic call model. For convenience, we call submaps describing the basic call scenarios *default submaps*.

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.

The basic call model in Figure 1 is the so-called *global call model* of *end-to-end view* [5], which contains both caller's and callee's scenarios in one model. The responsibilities in the diagram are those explained in [4]. 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.

2.2 Adding Features

Adding features extends scenarios in 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* describing the feature's scenarios. Figure 2 shows feature submaps for the following features:

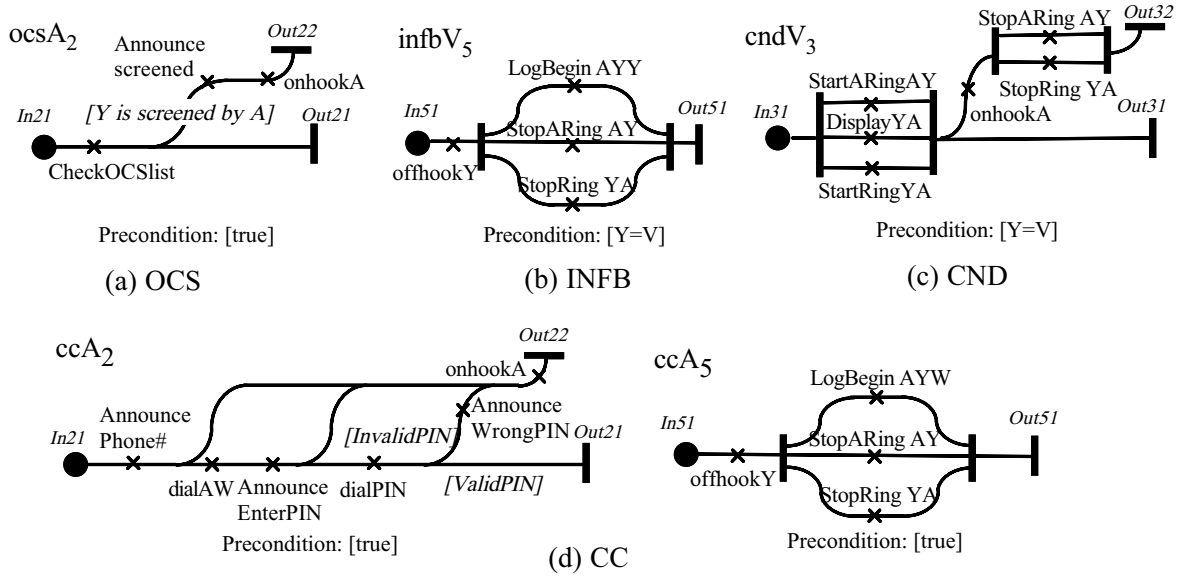


Figure 2: Submaps for supplementary features

(a) Originating Call Screening (OCS): OCS allows a subscriber A to screen originating calls based on a screening list. If a callee Y is in A's OCS list, the outgoing call from A to Y is screened.

(b) IN Freephone Billing (INFB): INFB allows the subscriber V to be charged for incoming calls instead of caller A. The scenario is activated only when the callee Y matches the subscriber V. The responsibility *LogBegin AYY* means that the call from A to Y is charged to Y.

(c) Calling Number Delivery (CND): CND enables the subscriber V to receive and display the number of the originating party A on an incoming call. The scenario applies only when the callee Y is the subscriber V. The responsibility *Display YA* means that the number of A is displayed on Y's phone.

(d) Charge Call (CC): CC permits the subscriber A to charge a call to a different address W than the originating address A, if the correct PIN is entered.

Each feature is described as a set of submaps describing scenarios specific to the feature. Each feature submap has an identifier with index that represents into which stub the submap can be plugged. For example, let us add CC to the Basic Call. This is done by plugging feature submaps ccA_2 and ccA_5 into stub 2 and 5 of the root map in Figure 1(a). As a result, def_2 and def_5 are replaced by ccA_2 and ccA_5, respectively, and all other submaps $def_i (i = 1, 3, 4, 6)$ remain to be reused in the corresponding stubs.

In UCMs, a stub is allowed to contain multiple submaps, whose selection can be determined at run-time according to a *submap selection policy* [2], which helps to describe some dynamic situations in scenarios. The selection policy is usually described with pre-conditions of submaps. When there are several submaps for one stub, exactly one of those, whose pre-condition takes true value, is chosen to be plugged into the stub. For convenience, let $pre(f_i)$ denote a pre-condition of a submap f_i.

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

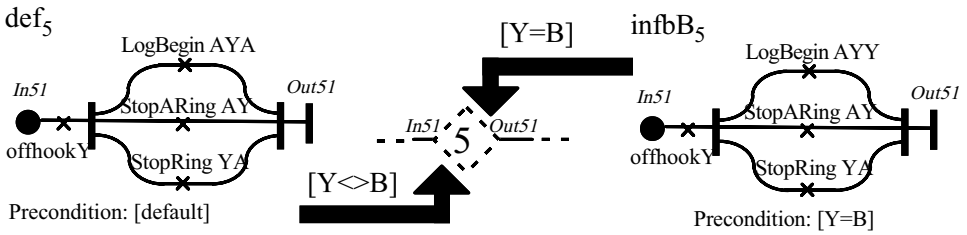


Figure 3: Plugging two submaps into one stub

For example, suppose that B is an INFB subscriber. If the caller A makes a call to B, the call is charged to B. When A calls C, A pays the fee normally. The call scenario dynamically changes depending on to whom A makes a call. To explain this, let us take a submap $infbV_5$ shown in Figure 2(b). The variable V in $infbV_5$ represents an INFB subscriber. We assign a value B to V. Then, the submap $infbV_5$ and its pre-condition $pre(infbV_5) = [Y = V]$ are instantiated to $infbB_5$ and $[Y = B]$. $infbB_5$ is plugged into stub 5 only when $[Y = B]$ holds, i.e. the callee Y is B. On the other hand, a submap def_5 in Figure 1(b) is also for the stub 5, which describes *default* scenarios in the Basic Call. The $pre(def_5)$ is default, which means def_5 is plugged in when none of other submap's pre-condition holds. One of $infbB_5$ or def_5 is chosen to be entered in stub 5 depending on whom A calls (see Figure 3). This shows that INFB 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 plug 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, it would be hard to evaluate them without human input. So, 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, def_5 and $infbB_5$ are mutually exclusive as shown above, thus $mex(def_5, infbB_5)$. However, ccA_5 and $infbB_5$ are not, since $pre(ccA_5) = true$ always holds. As a result, scenarios in ccA_5 and $infbB_5$ cause non-determinism when $[Y = B]$ holds. The non-deterministic behavior is well known as a typical situation of FI [12].

3 Feature Interaction

Researchers agree on an informal definition of FI: *FI occurs iff the combined use of 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 used before the FI detection process. In order to make the proposed method *generic*, i.e. applicable to different FI detection frameworks, we do not give in this paper an exact definition of FI. Instead of giving such a definition, we briefly characterize FIs by a necessary condition and a sufficient condition with respect to call scenarios of users.

Condition 3. 1 FI occurs $\Rightarrow$ Composition of features changes some user's call scenarios in an individual feature.

Condition 3. 2 FI occurs $\Leftarrow$ Composition of features enables different call scenarios to be performed under the same conditions.

Condition 3. 1 means that if no scenario changes occur in individual features, then FI does not occur. But the reverse does not necessarily hold, since scenario changes do not always contradict the requirements, and are sometimes acceptable. Condition 3. 2 characterizes non-determinism [12], which is one of the most typical situations of FIs. However, the reverse does not hold, since not all FIs are caused by non-determinism.

4 Proposed Feature Interaction Filtering

4.1 Stub Configuration Matrix

The stub plug-in concept in UCMs enables us to isolate specific scenarios of features from common scenarios between features. 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, to characterize features.

Definition 4. 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, 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 $|$. They are used to represent which submaps are plugged into each stub. A matrix element $f_1|f_2|...|f_k$ represents the fact 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 all UCMs in Section 2. Then, for instance, def_4, $ocsA_5$, $def_5|infbB_5$ are all matrix elements.

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

Definition 4. 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 2. Let us briefly characterize A's scenarios for individual features, in terms of an SC-vector. First consider the stub configuration where A is a CC subscriber. Submaps ccA_2 and ccA_5 are plugged into stub 2 and 5, respectively, and each other stub i contains the default submap def_i. Hence, A's scenarios are characterized by an SC-vector:

$$[def_1, ccA_2, def_3, def_4, ccA_5, def_6]$$

If B subscribes to INFB, then the call scenarios where A is a caller is characterized by:

$$[def_1, def_2, def_3, def_4, def_5|infbB_5, def_6]$$

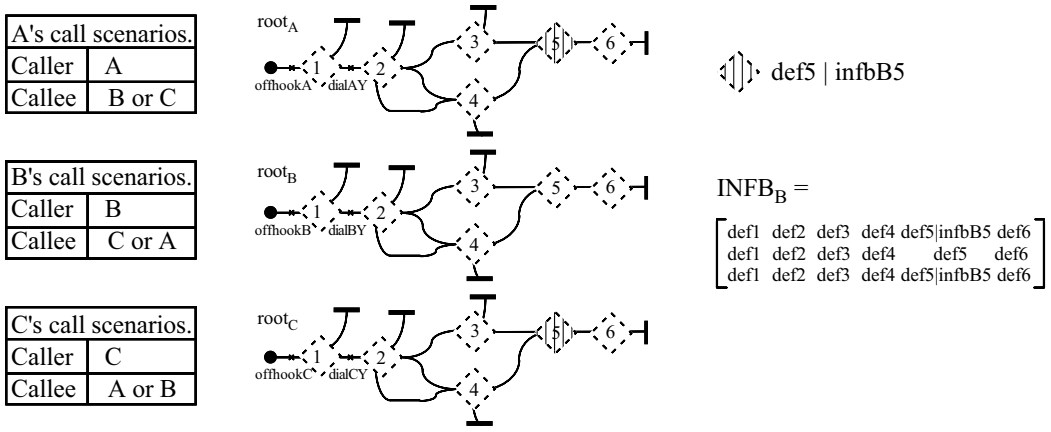


Figure 4: Extending a root map for three users, supposing that B subscribes to INFB

where the inclusion of $infbB_5$ or def_5 is determined by whether A calls B or a user who does not have INFB. Thus, the individual features on A's scenarios are concicely 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 [11][3] in telecommunication services. The stub configuration describes the allocation of feature submaps to stubs in the root maps of all users. Accordingly, the SC-vector is extended to a matrix form, called SC-matrix.

Definition 4. 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 name F and of its subscriber u. For convenience, we introduce a notation F_u to denote an SC-matrix where user u subscribes to feature F.

For example, consider all submaps in Figures 1 and 2, and root maps in Figure 4. Here we suppose that there are three users, A, B and C, as introduced in [4]. Since the root map has six stubs, an SC-matrix is a 3×6 matrix. Let us express the stub configuration where A is a CC subscriber. Submaps ccA_2 and ccA_5 are respectively plugged into stubs 2 and 5 in A's root map. Similarly we can describe the stub configuration that B is a CC subscriber just by swapping A and B.

$$CC_A = \begin{bmatrix} def_1 & ccA_2 & def_3 & def_4 & ccA_5 & def_6 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \end{bmatrix} \quad CC_B = \begin{bmatrix} def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & ccB_2 & def_3 & def_4 & ccB_5 & def_6 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \end{bmatrix}$$

[3]Suppose that A and B subscribe to the same feature F. Then, A and B are guaranteed to use F in the same way.

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, $INFB_B$:

$$INFB_B = \begin{bmatrix} def_1 & def_2 & def_3 & def_4 & def_5|infbB_5 & def_6 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3 & def_4 & def_5|infbB_5 & def_6 \end{bmatrix}$$

Note that B's subscription to INFB affects call scenarios of both A and C, since INFB applies when A (or C) calls B (i.e. the condition $[Y = B]$ holds). Figure 4 shows a correspondence between the root maps for this subscription and their SC-matrix. The shaded stubs imply that the submaps for INFB are plugged into those stubs.

One of the useful guidelines to construct systematically 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 caller side, while terminating features are the features whose subscriber is on callee side. In our example, OCS and CC are originating features, whereas INFB and CND are terminating features [4]. Note that the root map in our example is described from the caller's viewpoint. Subscribing to originating features affects the scenarios of the subscriber only. On the other hand, subscribing to terminating features effects the scenarios of all users, except the subscriber. Based on this observation, let us give SC-matrices OCS_B and CND_A:

$$OCS_B = \begin{bmatrix} def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & ocsB_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \end{bmatrix} \quad CND_A = \begin{bmatrix} def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3|cndA_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3|cndA_3 & def_4 & def_5 & def_6 \end{bmatrix}$$

Note that it is possible to represent feature configurations for arbitrary subscribers in terms of SC-matrices. This is done by instantiating feature submaps with a value of the subscriber and allocating them to appropriate rows of the SC-matrix. For instance, $INFB_C$ can be obtained from $INFB_B$ by swapping the second and third rows, and by letting $V = C$ instead of $V = B$ in submap $infbV_5$.

4.2 Feature Composition

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

Definition 4. 4 Suppose that f and g are given submaps plugged into the same stub in a root map. Let *def* denote any default submap describing basic call scenarios. Let ng [4] denote a special identifier not contained in the given submaps. Then, composition 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 composition is explained as follows: (A1) composition of the same submaps yields the same submap, (A2) a feature submap f can override

[4]The *ng* here stands for "no good".

a default map for the basic call scenario, (A3) two different feature submaps can be composed with a 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 composition operator is extended for matrix elements containing "|", by the following definition:

Definition 4. 5 Let $p = f_1|f_2|...|f_k$ and $q = g_1|g_2|...|g_l$ be matrix elements. Then, composition of p and q is definied 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 composition results.

Proposition 4. 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 CC_A and $INFB_B$ in the previous section. Let us compose two matrix elements ccA_5 and $def_5|infbB_5$, with respect to stub 5.

$$\begin{array}{rcll} ccA_5 \cdot (def_5|infbB_5) & = & ccA_5 \cdot def_5|ccA_5 \cdot infbB_5 & (A5) \\ & = & ccA_5|ccA_5 \cdot infbB_5 & (A2) \\ & = & ccA_5|ng & (A4) \\ & = & ng & (B4) \end{array}$$

Now, we define a composition operator of SC-matrices:

Definition 4. 7 Let F and G be given SC-matrices. Then, composition of F and G, denoted by $F \oplus G$, is defined as $F \oplus G = [f_{ij}] \oplus [g_{ij}] = [f_{ij} \cdot g_{ij}]$

Composition of two SC-matrices is carried out by applying $\cdot$ to each pair of corresponding matrix elements. For instance, let us compose OCS_B and CND_A shown in the previous section.

$$\begin{array}{rcl} OCS_B \oplus CND_A & = & \begin{bmatrix} def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & ocsB_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \end{bmatrix} \oplus \begin{bmatrix} def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3|cndA_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3|cndA_3 & def_4 & def_5 & def_6 \end{bmatrix} \\ & = & \begin{bmatrix} def_1 & def_2 & def_3 & def_4 & def_5 & def_6 \\ def_1 & ocsB_2 & def_3|cndA_3 & def_4 & def_5 & def_6 \\ def_1 & def_2 & def_3|cndA_3 & def_4 & def_5 & def_6 \end{bmatrix} \end{array}$$

4.3 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.

First of all, we provide two theorems used for the proposed FI filtering method. These theorems are derived from the FI characterizations (Conditions 3. 1 and 3. 2) presented in Section 3. 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 4. 8 *There exists ng in $H \Rightarrow$ FI occurs (non-determinism).*

Proof: By Definition 4. 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 3. 2 , we can conclude that FI occurs.

Feature Interaction Filtering Method

Input : Stub configuration matrices $F = \begin{bmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_m \end{bmatrix}$ and $G = \begin{bmatrix} \boldsymbol{g}_1 \\ \vdots \\ \boldsymbol{g}_m \end{bmatrix}$

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

Procedure :

Step1 : Make a composed matrix $H = \begin{bmatrix} \boldsymbol{h}_1 \\ \vdots \\ \boldsymbol{h}_m \end{bmatrix} = F \oplus G$.

Step2 : If some ng elements exist in H, conclude (a) FI occurs (by Theorem 4. 8). Otherwise, go to Step 3.

Step3 : For each row $\boldsymbol{h}_i$ of H, check a condition [$\boldsymbol{h}_i = \boldsymbol{f}_i$ or $\boldsymbol{h}_i = \boldsymbol{g}_i$].

Step3-1 : If the condition holds for all i ($1 \leq i \leq m$), conclude (b) FI never occurs (by Theorem 4. 9). Otherwise,

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

Figure 5: Feature Interaction filtering method

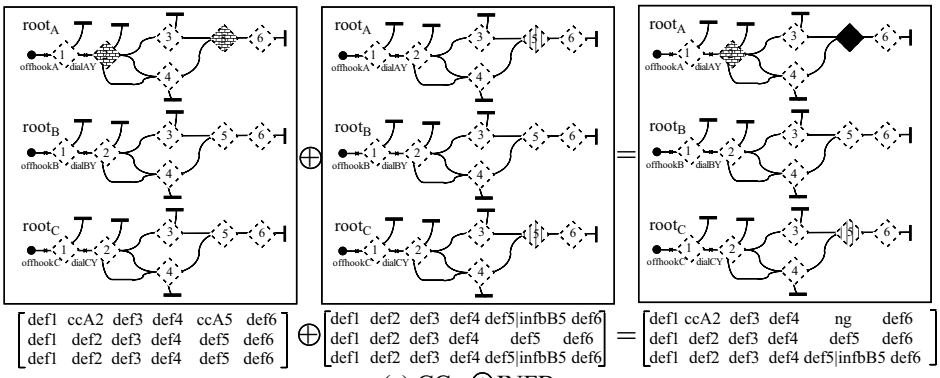
Theorem 4. 9 [$\boldsymbol{h}_i = \boldsymbol{f}_i$ *or* $\boldsymbol{h}_i = \boldsymbol{g}_i$] *holds for all* $i \Rightarrow$ *FI does not occur.*

Proof: Each row in an SC-matrix is an SC-vector that characterizes one user's scenarios. 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 composition 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 composition. Hence, no scenario change occurs with respect to 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 composition. According to a contraposition of Condition 3. 1 , we can conclude that FI never occurs.

Figure 5 shows the proposed filtering method. The method gives one of the verdicts: (a) FI occurs (non-determinism), (b) FI never occurs, (c) FI-prone, for given two SC-matrices F and G. Since Theorems 4. 8 and 4. 9 at Steps 2 and 3 can be checked easily, the filtering procedure is quite simple and easy to use. Step 1 is just for making a composed matrix H from F and G. Step 2 is to check the non-determinism caused by the composition by Theorem 4. 8 . Step 3 is for checking if some scenario changes occur due to the composition by using Theorem 4. 9 . If we reach Step 3-2, it means that there is no non-determinism, but some scenarios change in the composition. We cannot definitely conclude the existence of FI at this moment. The verdict is "FI-prone" and some detection method has to be employed.

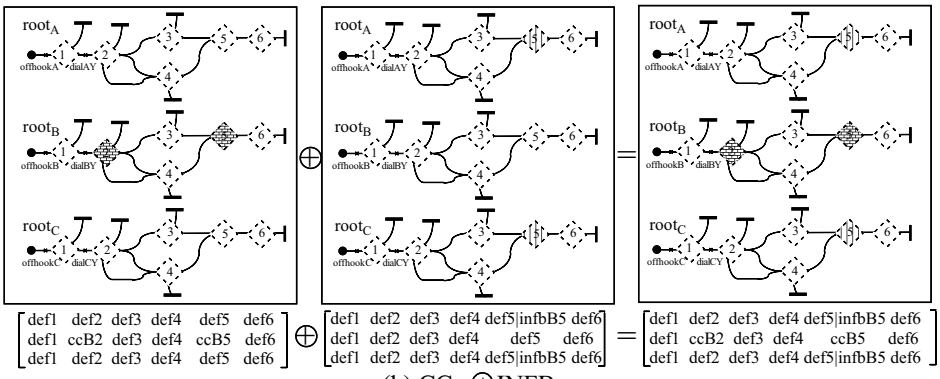
We will show some examples to illustrate each of verdicts (a), (b) and (c). Figure 6 explains the correspondence between matrix composition and related root maps. The stubs depicted by shaded diamonds represent that some feature submaps are plugged into the stubs.

First, Figure 6(a) illustrates the verdict (a) "FI occurs" for the combination of CC_A



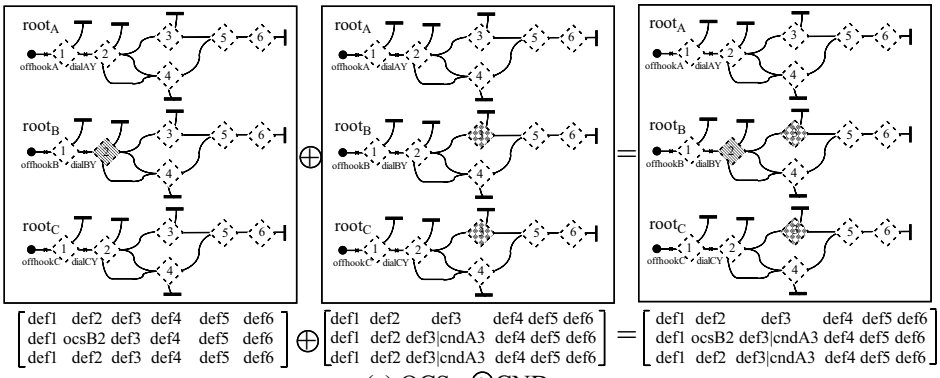
def1	ccA2	def3	def4	ccA5	def6	⊕	def1	def2	def3	def4	def5|infbB5	def6	=	def1	ccA2	def3	def4	ng	def6
def1	def2	def3	def4	def5	def6		def1	def2	def3	def4	def5	def6		def1	def2	def3	def4	def5	def6
def1	def2	def3	def4	def5	def6		def1	def2	def3	def4	def5|infbB5	def6		def1	def2	def3	def4	def5|infbB5	def6

(a) $CC_A \oplus INFB_B$



def1	def2	def3	def4	def5	def6	⊕	def1	def2	def3	def4	def5|infbB5	def6	=	def1	def2	def3	def4	def5|infbB5	def6
def1	ccB2	def3	def4	ccB5	def6		def1	def2	def3	def4	def5	def6		def1	ccB2	def3	def4	ccB5	def6
def1	def2	def3	def4	def5	def6		def1	def2	def3	def4	def5|infbB5	def6		def1	def2	def3	def4	def5|infbB5	def6

(b) $CC_B \oplus INFB_B$



def1	def2	def3	def4	def5	def6	⊕	def1	def2	def3	def4	def5	def6	=	def1	def2	def3	def4	def5	def6
def1	ocsB2	def3	def4	def5	def6		def1	def2	def3|cndA3	def4	def5	def6		def1	ocsB2	def3|cndA3	def4	def5	def6
def1	def2	def3	def4	def5	def6		def1	def2	def3|cndA3	def4	def5	def6		def1	def2	def3|cndA3	def4	def5	def6

(c) $OCS_B \oplus CND_A$

Figure 6: Illustrative examples

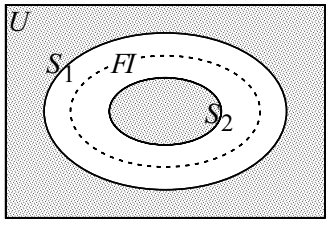


Figure 7: Classification of combinations

and $INFB_B$. After the composition, an ng entry appears in stub 5 of A's root map. This is a non-deterministic interaction: "Suppose that A subscribes to CC and that B subscribes to INFB. If A calls B, should the call be charged to B or W (third party to whom A wanted to charge)?".

The next example in Figure 6(b) is for the verdict (b) "FI never occurs" between CC_B and $INFB_B$. The condition $[\boldsymbol{h}_i = \boldsymbol{f}_i$ or $\boldsymbol{h}_i = \boldsymbol{g}_i]$ holds for $i = A, B, C$. In this case, the scenarios of users A and C had been expected in $INFB_B$ before the composition, whereas the scenarios of user B had been found in CC_B. As a result, no scenario change occurs, and thus we can conclude that there is no FI. Note that even if the feature combination is the same (as in (a)), we could get a different verdict depending on who are the subscribers.

The third example in Figure 6(c) is for the verdict (c) "FI-prone" between OCS_B and CND_A. Due to the composition, user B's scenarios have been changed, which can be interpreted as follows: "Suppose that B subscribes to OCS, and that A subscribes to CND. Even if B dials A, B's number may not be displayed on A's telephone, since A could screen the outgoing call to B." Whether this is an FI or not depends on the exact definition of FI adopted, and will have to be decided by an appropriate detection process. The only thing we can say here is that the system is FI-prone.

The proposed filtering method is applied to all possible combinations of SC-matrices, derived from given features. Figure 7 shows a classification of the combinations. In the diagram, U represents the set of all possible combinations, and FI represents the set of combinations that actually cause FIs. S_1 and S_2 respectively denote the sets of combinations satisfying Conditions 3. 1 and 3. 2 mentioned in Section 3. The combinations with verdicts (a) or (b) must be *filtered*. Then, Step 2 filters the inside of S_2, and Step 3 filters the outside of S_1. Consequently, the set of combinations filtered by the proposed method can be depicted by the shaded portions. The combinations with the verdict (c) FI-prone are contained in the non-shaded portion, for which we would not know if the combinations cause FI or not without further analysis. Such combinations are left for the FI detection process after the filtering.

The number of combinations increases combinatorially with the numbers of users and features. However, the number of combinations can be reduced by using *symmetry*. For example, if we have analyzed a combination $OCS_B \oplus CND_A$, then we no longer need to try $OCS_C \oplus CND_B$, since all subscribers of a feature can use the feature in the same way. Due to space limitation, the detailed definition of symmetry is omitted here. Interested readers can refer to relevant papers [8, 11].

5 Experimental Evaluation

We have conducted an experimental evaluation to measure the proposed method from the following two viewpoints: (a) *efficiency*: how many feature combinations can be filtered, (b) *quality*: whether the filtering method gives correct verdicts or not. We have prepared UCMs for the following eight features, and performed FI filtering for all pairs of the features: Terminating Call Screening (TCS), Originating Call Screening (OCS), IN Freephone Billing (INFB), IN Freephone Routing (INFR), Charge Call (CC)[5], Call Number Delivery (CND), IN Teen Line (INTL), and Call Forwarding Busy Line (CFBL). The feature definitions were taken from reference [4].

Table 1 shows the filtering results. For each pair of features, we have two combi-

[5]We assume that the system interprets the dialed number $0{+}Y$ in CC as Y.

Table 1: Filtering results

	TCS		OCS		INFB		INFR		CC		CND		INTL		CFBL	
	same	diff	same	diff	same	diff	same	diff	same	diff	same	diff	same	diff	same	diff
TCS		c	b	a*	c	c	a*	c*	b	a*	c	c	b	c	c*	c*
OCS				b	b	c*	b	a*	a*	b	b	c*	c	b	b	c*
INFB						c	c	c	b	a*	c	c	b	c	a*	c*
INFR								c*	b	a*	c	c	b	c	a*	c*
CC										b	b	c*	c	b	b	a*
CND												c	b	c	c*	c*
INTL														b	b	c*
CFBL																c

nations. The first is the case where both features are allocated to the same user (e.g. $CC_B \oplus INFB_B$), which is shown in the columns "same" in the table. The second is the case where two features are allocated to different users (e.g., $CC_A \oplus INFB_B$), shown as columns "diff". Each entry in the table shows one of the verdicts (a) FI occurs, (b) FI never occurs, (c) FI-prone, as defined by the proposed method. The entries with asterisk $*$ represent feature combinations that actually cause FIs, based on the results provided in [4].

First, we evaluate the efficiency of the proposed method. As shown in the table, there are 64 combinations in total. The number of combinations with verdicts (a) or (b) is 31, which is almost half of all the combinations. That is, half of the combinations can be filtered. This implies that the number of combinations to be analyzed with more expensive detection methods can be reduced to half by using the proposed method as a front end of the detection process.

Next, we evaluate the quality of the proposed method. All the combinations that cause FIs (entries with asterisk $*$ in Table 1) are correctly covered by verdicts (a) or (c). Although these percentages may not mean much on this small sample, it is interesting to note that 42% (14 out of 33) of the interaction-prone verdicts involved actual FIs. Also, and again accurately, there are no FIs in combinations with verdict (b). Therefore, the proposed method achieves a good predictive quality in this case study.

6 Discussion

6.1 Scalability

One of the key issues is *scalability*, i.e. whether the method can be extended to realistically large and complex sets of features. Although the filtering method proposed in Figure 5 is for composition of two SC-matrices F and G, it can be extended to more than two matrices. Due to the well-defined composition operator $\oplus$, the following properties are satisfied: (C1) $F \oplus G = G \oplus F$ (commutative), (C2) $F \oplus (G \oplus H) = (F \oplus G) \oplus H$ (associative), (C3) $F \oplus F = F$ (idempotent) (C4) $F \oplus POTS = F$ (zero element). Those properties allow us to analyze more than two features easily.

Also, the complexity cost in $\oplus$ is linear in the number of rows and the number of columns of SC-matrices, assuming that composition of each pair of submaps takes a constant time (See Definition 4. 4). This implies that even if the numbers of users and stubs increase along with the feature's complexity, the proposed method is well applicable to such complex cases. Thus, the proposed filtering is scalable with respect

to the numbers of features and users as well as to the complexity of the features.

6.2 Usefulness at Requirements Stage

The proposed method extensively uses the UCMs for filtering. Basically, paths and responsibilities are all formal elements of the UCM notation here. Additional information, such as guards, pre-conditions and user constants/variables, or labels for start/ends points may be associated with these elements for human use, but this information is not formal. This lack of formality in the details makes the notation and the filtering method itself *lightweight*, allowing requirements engineers to examine, visualize and understand the FI analysis without having detailed implementation information or complex theory/semantics. In this sense, UCM-based methods are semi-formal and they are appropriate for FI "filtering", but not for "detection".

6.3 Subsequent FI Detection Process

By the nature of FI filtering, we need to perform FI detection for the FI-prone feature combinations after the filtering process. In order to do this, it is usually necessary to employ more formal definition and techniques. However, the characterization of FIs described in this paper is so generic that it could be mapped to several formalisms. Research is being conducted in the context of adapting our technique to LOTOS [2], SDL, Petri nets and so on. These techniques are especially promising in this respect.

The proposed method gives us not only verdicts of likelihood of FIs, but also useful information for FI detection to localize them. For example, consider again the example shown in Figure 6(c). This combination is concluded to be FI-prone. If there is any FI in this combination, it must be in B's scenarios since only B's scenarios have been changed by the composition. Hence, in the detection process, we can focus on B's scenarios only. This information can be utilized in detection processes such as guided state space search and test case generation.

6.4 Related Work

Keck [8] proposed an FI filtering method that focuses on topological relationships between feature subscribers. Since the method examines relatively detailed information, such as detection points and resource description, it is well applicable at a detailed specification stage, but not at the requirements stage. Heisel et al. [6] proposed a heuristic approach at requirements level. However, this method requires sophisticated knowledge elicitation to map requirements to their formal specification based on schematic expressions, which may be a difficult task for requirements engineers. Static FI detection methods (e.g. Nakamura et al.[12], Yoneda et al.[13]) using only a necessary condition and/or heuristics might be called filtering. These methods are at the software specification level rather than at the requirements level [1].

7 Conclusion

We have proposed an FI filtering method based on UCMs. Using the stub plug-in concept of UCMs, we can characterize each feature in terms of its stub configuration. This introduced a notational convention that can be useful to represent features with UCMs. The different stub configurations are composed by means of matrix composition,

and two simple and generic conditions are exploited for the filtering method. The experimental evaluation shows that approximately half of the combinations can be filtered. Basically, the matrix composition is performed by checking only pre-conditions of the feature submaps, independently of the detail of the submaps. Thus, the proposed method is easy to use and scalable.

Some topics for future research present themselves. Further information in the feature submaps and more strict (and reasonable) conditions would improve filtering quality. In addition, the assumption that submaps cannot contain stubs should be relaxed. This would facilitate hierarchical analysis of complex feature scenarios. Also, the application to more complex features like conference calls has to be studied.

Acknowledgements: This research was partly supported by Grant-in-Aid for JSPS Fellow (No.100987) from Ministry of Education Japan, Nortel, NSERC and CITO.

References

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

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

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

[4] Griffeth, N., Blumenthal, R., Gregoire, J.C. and Ohta, T., "Feature interaction detection contest," *Proc. of Fifth Workshop on Feature Interactions and Software Systems (FIW'98)*, pp.327-359, IOS Press 1998.

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

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

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

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

[9] Kimbler, K., "Addressing the interaction problem at the enterprise level", *Proc. of Fourth Workshop on Feature Interactions in Telecommunications Systems (FIW'97)*, pp.13-22, July, 1997.

[10] Miga, A., "Application of Use Case Maps to system design with tool support", *M.Eng. thesis*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada.

[11] Nakamura, M. and Kikuno, T., "Exploiting symmetric relation for efficient feature interaction detection", *IEICE Trans. on Information and Systems*, vol.E82-D, no.10, pp.1352-1363.

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

[13] Yoneda, T. and Ohta, T., "A formal approach for definition and detection of feature interactions", *Proc. of Fifth Workshop on Feature Interactions and Software Systems (FIW'98)*, pp.165-171, IOS Press 1998.