Feature Interactions in Telecommunications and Software Systems V

Edited by

K. Kimbler

Department of Communication Systems, Lund University, Lund, Sweden

and

L.G. Bouma

KPN Research, Leidschendam, The Netherlands

IOS Press

Ohmsha

Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

*Feature Interactions in Telecommunications
and Software Systems V
K. Kimbler and L.G. Bouma (Eds.)
IOS Press, 1998*

Feature Interaction Detection Using Permutation Symmetry

Masahide Nakamura*, Yoshiaki Kakuda, Tohru Kikuno
*Department of Informatics and Mathematical Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka, 560-1531, Japan*
Phone: +81 6 850 6567 Fax:+81 6 850 6569
Email: {masa-n, kakuda, kikuno}@ics.es.osaka-u.ac.jp

Abstract: Most of conventional interaction detection methods on FSM model utilize an exhaustive search to identify undesirable states at which interactions occur. The exhaustive search is fundamentally very powerful in the sense that all interactions are exactly detected. However, it may suffer from the state explosion problem due to the exponential growth of the FSM when the number of users and the number of features increase.

In order to cope with this problem, we propose a new detection method using a state reduction technique of the FSM. By means of a symmetric relation, called permutation symmetry, we succeed in reducing the size of the FSM with preserving the necessary information for the interaction detection. As a result, we can exactly identify any interactions defined on the original FSM with smaller state space and time.

Experimental evaluation shows that, for practical interaction detection with three users, the proposed method achieves about 80% reduction in space and time, and is more scalable than the conventional ones especially for the increase of the number of users in the service. Thus, the proposed detection method enables us to verify complex services with many users.

1. Introduction

Feature interaction detection is one of the most important steps to realize consistent creation and deployment of new telecommunication services. So, it is strongly required to develop systematic method for effective and efficient interaction detection[9].

As discussed in [12][14], telecommunication services are often modeled by a finite state machine (FSM), in which a global state consisting of user's local states successively moves to a next state by the occurrence of user's event. Then, the interactions are usually defined on certain states in the FSM at which undesirable properties such as deadlock hold[8][12]. Therefore, a straightfoward approach to the interaction detection is to identify such undesirable states by exhaustively exploring all possible reachable states. Since this approach is quite simple but powerful, it is adopted by most of the conventional detection frameworks based on the FSM[3][4][8][12]. However, the number of states in the FSM grows exponentially in the number of the users and the number of features. Hence, the application of this approach may be limited to relatively simple services with small number of users.

In order to make it possible to deal with more complex services with many users, it is necessary to reduce the state space needed for the detection in a certain manner. For this, two policies can be considered, which we call *efficiency-oriented* and *effectiveness-oriented.*

* This work is partly supported by JSPS Research Fellowships for Young Scientists

The efficiency-oriented reduction mainly focuses on the large reduction of the state space rather than the optimal interaction detection. Cameron et.al[2] proposed the tool *CADRES-FI* which abstracts the details of the service model by means of a kind of heuristics. Kimbler introduced a concept of interaction filtering[9] that makes rough and quick pre-evaluation of the *interaction-prone* service combination before the detection process. Nakamura et.al[11] proposed a Petri Net based method using only necessary condition for the non-deterministic interactions. In general, the efficiency-oriented reduction will attain drastic state reduction and semi-optimal detection as shown in [11]. Instead, the detection algorithms may miss some interactions, or may detect the redundant interactions which do not actually occur. Also, the types of detectable interactions may be limited.

On the other hand, the effectiveness-oriented reduction attempts to reduce the state space with completely preserving necessary information for the interaction detection. To the best of our knowledge, the feature interaction detection based on this policy has not been proposed yet, but there are several state reduction techniques based on this policy such as partial order reduction[6] and symbolic model checking[10] in other research fields. The effectiveness-oriented reduction will realize the *exact* (optimal) interaction detection and it can be applied to any types of interactions defined on the FSM. Instead, the reduction ratio may not be so significant as that of the efficiency-oriented reduction. The relationship between two policies is clearly trade-off, thus, they should be chosen for different purposes.

In this paper, we propose a new effectiveness-oriented reduction method for the interaction detection. The key idea to achieve the reduction is to utilize a *permutation symmetry*[7] with respect to users. In the telecommunication systems, there exists a specific constraint that all subscribers of a service X are guaranteed to be able to use the same functionality of X. Under this constraint, suppose that both users A and B are subscribers of X. If we know A's possible behavior on X, then we can infer B's behavior on X from A's, because B can use X in the same way as A. Therefore, we can discard the state transitions for B's behaviors since they can be reproduced from A's. Based on this idea, we define a relation *symmetrical* on the states and transitions for the state reduction, and then provide theorems for the detection of three types of interactions: deadlock, loop, and non-determinism.

Also, we evaluate the proposed method through two experiments. The first experiment shows that, for the practical interaction detection with three users, the proposed method can exactly identify all interactions and realizes about 80% reduction of the space and time in the original FSM. The second experiment shows that the proposed method is very scalable for the increase of the number of users. These imply that the proposed method is much more applicable to the interaction detection of complex services than the conventional exhaustive methods.

The rest of this paper is organized as follows: Section 2 provides the necessary definitions for the interaction detection. Section 3 discusses the conventional detection method based on the exhaustive search. In Section 4, we propose a new interaction detection method using permutation symmetry. In Section 5, we perform the experimental evaluation. Finally, Section 6 concludes this paper with future research.

2. Preliminaries

2.1. Service Specification

In this paper, we adopt a *rule-based service specification* for a service description method. In this method, the functionalities of the service (or feature) are described as a set of rules, each of which prescribes a state transition. The rule-based descriptions are studied for the practical

```
U = {A,B}
V = {x,y}
P = {idle, dialtone, calling, busytone, talk}
E = {offhook, onhook, dial}
R = {          /* (pre-condition)          (event)        (post-condition)  */
    pots1:                idle(x)       [offhook(x)]     dialtone(x).
    pots2:            dialtone(x)       [onhook(x)]      idle(x).
    pots3:   dialtone(x) , idle(y)      [ dial(x,y) ]    calling(x,y).
    pots4:  dialtone(x) , ~idle(y)      [ dial(x,y) ]    busytone(x).
    pots5:           calling(x,y)       [onhook(x)]      idle(x) , idle(y).
    pots6:           calling(x,y)       [offhook(y)]     talk(x,y) , talk(y,x).
    pots7:   talk(x,y) , talk(y,x)      [onhook(x)]      idle(x) , busytone(y).
    pots8:            busytone(x)       [onhook(x)]      idle(x).       }
s0 =   idle(A), idle(B).
```

Figure 1. Rule-based specification for POTS

use, as shown in STR[12] and declarative transition rules[4]. Note that the following notation is similar to STR, but is slightly different from STR.

A) Notation

Definition 1: A service specification S is defined as $S = \langle U, V, P, E, R, s_0 \rangle$, where

(a) U is a set of *constants* representing service users.

(b) V is a set of *variables*.

(c) P is a set of *predicate symbols*.

(d) E is a set of *event symbols*.

(e) R is a set of *rules*. Each rule $r \in R$ is defined as follows:

r: pre-condition [event] post-condition.

Pre(post)-condition is a list of *predicates* $p(x_1, ..., x_k)$'s, where $p \in P$, $x_i \in V$ and k is called *arity* which is a fixed number for each p. Especially, pre-condition can include *negations* $\sim p(x_1, ..., x_k)$'s which implies $p(x_1, ..., x_k)$ does not hold. *Event* is a predicate $e(x_1, ... x_k)$, where $e \in E$, $x_i \in V$. For convenience, we represent pre-condition, event and post-condition of rule r as *Pre[r], Ev[r]* and *Post[r]*, respectively.

(f) s_0 is the *(initial) state*. A state is defined as a list of *instances* of predicates $p(a_1, ..., a_k)$'s, where $p \in P$, $a_i \in U$. We think of each state as representing a truth valuation[10] where instances in the list are true, and instances not in the list are false.

Example 1: Figure 1 shows a service specification for POTS. For example, take a rule pots4. Then, *Pre[pots4] = dialtone(x), ~idle(y), Ev[pots4] = dial(x,y)* and *Post[pots4] = busytone(x)*. Intuitively, *pots4* describes a functionality of POTS meaning that "Suppose that user x receives dialtone and y is not idle. At this time, if x dials y, then x will receive a busytone". In state s_0, two instances *idle(A)* and *idle(B)* are true because they are included in s_0. On the other hand, any other instances (e.g., *dialtone(B)* or *calling(A,B)*) are false since they are not included in s_0. State s_0 means that two users A and B are idle.

B) State Transition Model

Here we define the state transitions prescribed by the service specification.

Definition 2: Let $S = \langle U, V, P, E, R, s_0\rangle$ be a service specification. For $r \in R$, let $x_1, \ldots, x_n (x_i \in V)$ be variables appearing in r, and let $\theta = \langle x_1|a_1, \ldots, x_n|a_n\rangle$ $(a_i \in U)$ be a substitution replacing each x_i in r with a_i. Then, an *instance of the rule r based on substitution* θ (denoted by $r\theta$) is defined as a rule obtained from r by applying $\theta = \langle x_1|a_1, \ldots, x_n|a_n\rangle$ to r.

Definition 3: Let s be a state and let $r\theta$ be an instance of the rule r based on substitution θ. Then, we say rule r is *enabled* for θ at s iff all instances in $Pre[r\theta]$ are included at s (i.e., all instances take true value at s).

When r is enabled for θ at s, *next state* s' of s can be generated by deleting all instances in $Pre[r\theta]$ from s and adding all instances in $Post[r\theta]$ to s. For convenience, we describe it by

$$s' = s - Pre[r\theta] + Post[r\theta]$$

where + and - respectively represent addition and deletion operators on the list. These operators work in the same way as union and subtraction operators on the set, respectively. At this time, we say a *state transition* from s to s' caused by an event $Ev[r\theta]$ is *defined on S*. We represent this state transition by a triple $(s, Ev[r\theta], s')$ for simplicity.

We say that state s is *reachable* from s_0 iff $s=s_0$ or a sequence of state transitions $(s_0,e_0,s_1), (s_1,e_1,s_2),\ldots, (s_n,e_n,s)$ is defined on S. For a given service specification S, we use $RS(S)$ to denote a set of all reachable states from s_0.

Example 2: Let us consider again POTS specification in Figure 1 and take rule *pots1*. If we apply a substitution $\theta = \langle x|A\rangle$ to *pots1*, then we obtain an instance of *pots1* based on θ:

pots1θ: *idle(A)* *[offhook(A)]* *dialtone(A).*

Since *idle(A)* is true at the initial state s_0, *pots1* is enabled for θ at s_0. If we apply *pots1θ* to s_0, we can get the next state s_1 by deleting *idle(A)* from s_0 and adding *dialtone(A)* to s_0:

$$s_1 = s_0 - Pre[pots1\theta] + post[post1\theta] = \text{“}idle(A), idle(B)\text{”} - \text{“}idle(A)\text{”} + \text{“}dialtone(A)\text{”}$$
$$= dialtone(A), idle(B)$$

At this time, a state transition $(s_0, offhook(A), s_1)$ is defined meaning that “Suppose that A and B are idle. If A offhooks, then A will receive a dialtone”.

Similarly, *pots3* is enabled for $\theta' = \langle x|A, y|B\rangle$ at s_1. If we apply $pots3\theta'$ to s_1, a state transition $(s_1, dial(A,B), s_2)$ is defined where

$$s_2 = calling(A,B)$$

This means that “Suppose that *A* receives dialtone and *B* is idle. If *A* dials *B*, then *A* will be calling *B*.” Both s_1 and s_2 are reachable from s_0, thus, $s_1, s_2 \in RS(S)$.

2.2. Problem Formulation

In this paper, we especially focus on the following three types of interactions. These are very typical cases of interactions and are discussed in many papers(e.g., [5][8][12]):

(a) deadlock: Functional conflicts of two or more services cause a mutual prevention of their service execution, which result in a deadlock.

(b) loop: The service execution is trapped into a loop from which the service execution never returns to the initial state.

(c) non-determinism: An event can simultaneously activate two or more functionalities of different services. As a result, it cannot be determined exactly which functionality should be activated.

Before formally defining the above interactions, we define a combined operator of two service specifications. Because of the good modularity of the rule-based specification, we can easily combine two specifications as follows.

Definition 4: For two specifications $S_1 = \langle U_1, V_1, P_1, E_1, R_1, s_{10}\rangle$ and $S_2 = \langle U_2, V_2, P_2, E_2, R_2, s_{20}\rangle$, we define a *combined specification* $S_1 \oplus S_2 = \langle U, V, P, E, R, s_0\rangle$ such that $U = U_1 \cup U_2$, $V = V_1 \cup V_2$, $P = P_1 \cup P_2$, $E = E_1 \cup E_2$, $R = R_1 \cup R_2$ and $s_0 = s_{10} + s_{20}$ where + denotes the addition operator in Definition 3.

Now, we are ready to define the feature interactions on the rule-based service specification. For each of three types of interactions mentioned above, we define the *undesirable states* each of which causes the corresponding interaction.

Definition 5: Let $S = \langle U, V, P, E, R, s_0\rangle$ be a given service specification. For any state $s \in RS(S)$, s is said to be a

(a) *deadlock state*: iff no rule $r \in R$ is enabled for any substitution θ at s.

(b) *loop state*: iff there exists a sequence of state transitions (s,e_1,s_2), (s_2,e_2,s_3),..., (s_n,e_n,s) and s_0 is not reachable from s.

(c) *non-deterministic state*: iff there exists a pair of rules $r, r' \in R$ such that r and r' are enabled for θ and θ' at s, respectively, and that $Ev[r\theta]=Ev[r'\theta']$.

A service specification S is called *safe* iff S has none of the above states.

For two service specifications S_1 and S_2, we say that S_1 *interacts* with S_2 iff both S_1 and S_2 are safe and $S_1 \oplus S_2$ is not safe.

3. Interaction Detection by Full Reachability Graph

A straightforward and conventional approach for detecting interactions between given two specifications S_1 and S_2 is to identify the undesirable states defined in Definition 5 by exploring all possible reachable states in $RS(S_1 \oplus S_2)$. This approach is very powerful since it can exactly detect any interactions. We show that this interaction detection can be performed by using full reachability graph.

3.1. Full Reachability Graph FRG

For a given service specification $S = \langle U, V, P, E, R, s_0\rangle$, the rule applications from initial state s_0 construct a finite state machine(FSM) consisting of all reachable states from s_0, in which a state moves to the next state by occurrence of an event. Since an FSM can be described by a labeled directed graph, here we directly define such an FSM as a directed graph. In the following, we represent a directed edge from s to s' labeled by e as a triple (s, e, s').

Definition 6: A labeled directed graph is defined as $G = \langle N, L, T\rangle$ where:

(a) N is a set of *nodes*.

(b) L is a set of *labels* attached to the directed edges.

(c) $T \subseteq N \times L \times N$ is a set of *directed edges*.

For any directed graph $G = \langle N, L, T\rangle$, a *directed path* ρ is a sequence of directed edges: $\rho=(s_1,e_1,s_2)$, (s_2,e_2,s_3),..., (s_n,e_n,s_{n+1}). For this, the node s_1 is called *head node* of ρ, while s_{n+1} is called *tail node* of ρ. n is called a *length* of ρ. A directed path is a *directed cycle* iff its

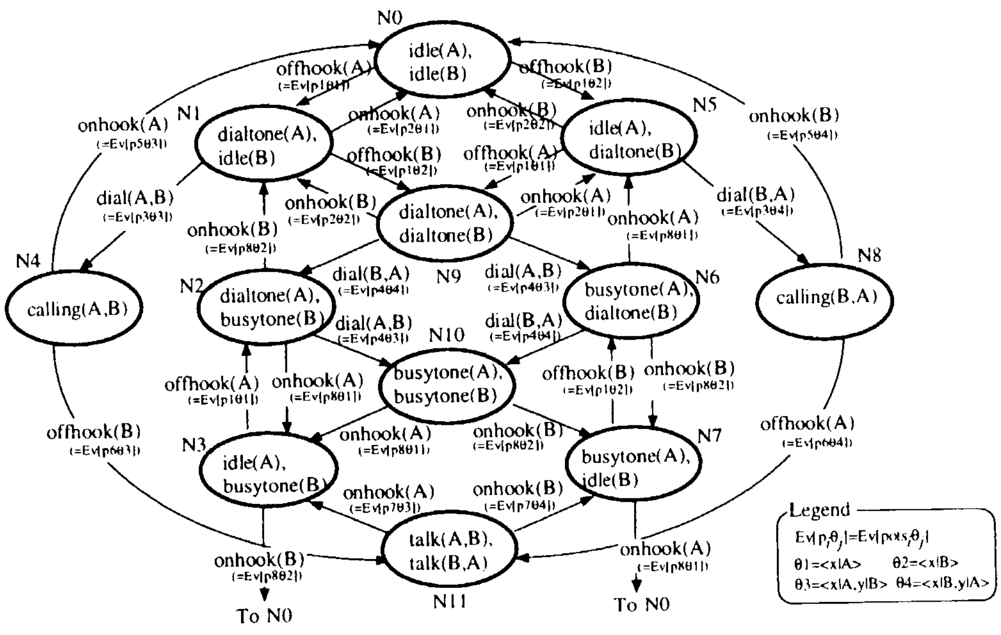


Figure 2. Full reachability graph for POTS specification

head node and tail node are identical. A node is called a *terminal* iff it has no outgoing edge.

Definition 7: Let $S = \langle U, V, P, E, R, s_0 \rangle$ be a service specification. A *Full reachability graph* for a given S is a labeled directed graph $FRG(S) = \langle N, L, T \rangle$ such that:

(a) $N = RS(S)$.

(b) L is a set of all instances of events.

(c) $T = \{(s, Ev[r\theta], s') \mid (s, Ev[r\theta], s')$ *is defined on* $S\}$.

Example 3: Figure 2 shows a full reachability graph for the POTS specification in Figure 1. In the figure, ovals represent nodes, arrows represent labeled directed edges. From the graph, we can see that 12 reachable states (represented by the nodes) and 30 state transitions (represented by the edges) are specified by POTS specification

3.2. Proof Rules for Interaction Detection

Using the full reachability graph FRG, we can easily identify the undesirable states in Definition 5 as follows.

Lemma 1: *The following properties are satisfied for FRG(S):*

(a) s is a terminal. $\Leftrightarrow$ s is a deadlock state.

(b) there exists a directed cycle starting from s, and there exists no directed path from s to s_0. $\Leftrightarrow$ s is a loop state.

(c) s has a pair of outgoing edges (s, e_1, s') and (s, e_2, s'') such that $e_1 = e_2$. $\Leftrightarrow$ s is a non-deterministic state.

Thus, in order to detect the interactions between given two specifications S_1 and S_2, we first construct FRG($S_1 \oplus S_2$), then identify the undesirable states using Lemma 1.

Although the interaction detection using FRG is quite simple and powerful, it may suffer from the *state explosion problem.* That is, the size of the FRG exponentially grows when the number of users and the number of rules in the specification become large (numerical studies will be presented in Section 5). Therefore, the application of the detection using FRG is limited to relatively simple services with small number of users.

4. State Reduction by Permutation Symmetry

In order to cope with the state explosion problem, we try to reduce the size of FRG without losing the necessary information for the interaction detection. The key idea to achieve this reduction is to utilize a relation among states called *permutation symmetry.* Although the basic idea of the permutation symmetry was originally proposed in several fields such as colored Petri Net verification[7], we show in this paper that the state reduction is successfully realized by means of a specific constraint on telecommunication services.

4.1. Key Idea

In the telecommunication services, there exists a specific constraint that "*all subscribers of a service X are guaranteed to be able to use the same functionality of X*". For example, if users A and B are subscribers of POTS, then both A and B can use the same functionalities of POTS. Let us consider that A uses POTS's dialing functionality to B: "Suppose that A receives a dialtone and B is idle. At this time, if A dials B, then A will be calling B". Similarly, when B uses this functionality, the situation is: "Suppose that B receives a dialtone and A is idle. At this time, if B dials A, then B will be calling A." We can easily convince that two situations are *symmetrical* with respect to users, that is, the one can be inferred from the other only by swapping A and B.[*]

In terms of our service specification, we can observe the symmetry on states and state transitions. For example, consider the following two states s_1 and s_2:

s_1 = *dialtone(A), idle(B), busytone(C)* $\qquad$ s_2= *dialtone(C), idle(A), busytone(B)*

We see s_1 and s_2 are symmetrical, in the sense that s_2 is obtained from s_1 just by substituting A for C, B for A, C for B. In other words, letting $U=\{A,B,C\}$, there exists a *permutation* $\phi;U \rightarrow U$ such that $\phi(A)=C$, $\phi(B)=A$, $\phi(C)=B$ from s_1 to s_2.

Now, let us apply the following rule to s_1 and s_2:

pots3: dialtone(x), idle(y) [dial(x,y)] calling(x,y).

pots3 is enabled for $\theta 1 = \langle x|A, y|B\rangle$ at s_1, and is also enabled for $\theta 2 = \langle x|C, y|A\rangle$ at s_2.

pots3θ1: dialtone(A), idle(B) [dial(A,B)] calling(A,B).

pots3θ2: dialtone(C), idle(A) [dial(C,A)] calling(C,A).

As a result, the following next states s_1' and s_2' are obtained from s_1 and s_2, respectively.

s_1' = *calling(A,B), busytone(C)* $\qquad$ s_2'=*calling(C,A), busytone(B).*

We can observe that there also exists the same permutation ϕ from s_1' to s_2'. Roughly speaking, this fact implies that state transition (s_2, *dial(C,A)*, s_2') can be reproduced from (s_1, *dial(A,B)*, s_1') in terms of ($\phi(s_1)$, ϕ(*dial(A,B)*), $\phi(s_1')$). Therefore, we need no longer to store either states s_2, s_2' or transition (s_2, *dial(C,A)*, s_2') in the reachability graph.

From this observation, we reach a hypothesis in general case that if we have a state transition *(s,e,s')*, then we also have $(\phi(s),\phi(e),\phi(s'))$ for any permutation ϕ on U. If the

[*] Note that the symmetry cannot be applied to systems such like "only supervisor C is allowed to execute a command Y, others are not", since the symmetry among users is broken.

hypothesis is true, then it is sufficient to have only states s and s' and a transition (s,e,s') in a reachability graph, since we can infer any symmetric states and transitions from s, s' and (s,e,s'). Actually we can verify all interactions defined in Definition 5 by using a new reachability graph, which is much smaller than FRG (to be shown in Section 4.5).

4.2. Permutation Symmetry

In this section, we formally define the permutation symmetry. Throughout this section, we assume that a service specification $S = \langle U, V, P, E, R, s_0\rangle$ is given unless especially specified. First, we define all permutations with respect to users.

Definition 8: Let $Perm(U)$ denote a set of all permutations $\phi;U \to U$. Each element ϕ of $Perm(U)$ is called a *permutation symmetry.*

Example 4: Let $U = \{A, B, C\}$. Then,

$$Perm(U) = \left\{ \begin{bmatrix} A & B & C \\ A & B & C \end{bmatrix} \begin{bmatrix} A & B & C \\ A & C & B \end{bmatrix} \begin{bmatrix} A & B & C \\ B & A & C \end{bmatrix} \begin{bmatrix} A & B & C \\ B & C & A \end{bmatrix} \begin{bmatrix} A & B & C \\ C & A & B \end{bmatrix} \begin{bmatrix} A & B & C \\ C & B & A \end{bmatrix} \right\}$$

Each permutation symmetry of *Perm(U)* specifies a bijection ϕ from U to U. For example, $\begin{bmatrix} A & B & C \\ C & A & B \end{bmatrix}$ specifies a bijection ϕ such that $\phi(A) = C, \phi(B) = A, \phi(C) = B$.

Next, we extend $\phi \in Perm(U)$ for states, rules and substitutions as follows.

Definition 9: Let $\phi \in Perm(U)$ be a permutation symmetry. Then, for any instance $p(a_1, a_2, \ldots, a_k)$ of a predicate with $p \in P, a_i \in U$, we define $\phi(p(a_1, a_2, \ldots, a_k)) = p(\phi(a_1), \phi(a_2), \ldots, \phi(a_k))$. At this time,

(a) For any state s, we define $\phi(s)$ to be a state obtained by applying ϕ to each instance of predicate in s.

(b) For any instances $r\theta$ of rule $r \in R$, we define $\phi(r\theta)$ to be an instance of rule obtained by applying ϕ to each instance of predicate in $r\theta$. Similarly, we define $\phi(Pre[r\theta])$, $\phi(Post[r\theta])$ and $\phi(Ev[r\theta])$.

(c) For any substitution $\theta = \langle x_1|a_1, \ldots, x_n|a_n\rangle$, $x_i \in V, a_i \in U$, we define $\phi(\theta) = \langle x_1|\phi(a_1), \ldots, x_n|\phi(a_n)\rangle$.

Two states s_1 and s_2 are *symmetrical* iff there exists $\phi \in Perm(U)$ such that $\phi(s_1) = s_2$.*

For a given state s, the *symmetric class* of s, denoted by $[s]$, is a set of all states symmetrical with s. For $[s]$, s is called a *representative* of $[s]$.

Example 5: Consider again the example discussed in Section 4.1. Then, for a permutation symmetry $\phi = \begin{bmatrix} A & B & C \\ C & A & B \end{bmatrix} \in Perm(U)$, we can see that (a) $\phi(s_1) = s_2$ and $\phi(s_1') = s_2'$, (b) $\phi(pots3\theta1) = pots3\theta2$, and (c) $\phi(\theta1) = \langle x|\phi(A), y|\phi(B)\rangle = \langle x|C, y|A\rangle = \theta2$.

Lemma 2: *Let a permutation symmetry* $\phi \in Perm(U)$ *be given. Then, for any rule* $r \in R$

* The relation *symmetrical* (s_1 *sym* s_2) is an equivalence relation on states since (i) s_1 *sym* s_1 (reflexive), (ii) s_1 *sym* s_2 implies s_2 *sym* s_1 (symmetric) and (iii) s_1 *sym* s_2 and s_2 *sym* s_3 imply s_1 *sym* s_3 (transitive).

and any substitution θ, $\phi(r\theta) = r\phi(\theta)$ *holds. Also,* $\phi(Pre[r\theta]) = Pre[r\phi(\theta)]$, $\phi(Post[r\theta]) = Post[r\phi(\theta)]$ *and* $\phi(Ev[r\theta]) = Ev[r\phi(\theta)]$ *hold.*

Intuitively, Lemma 2 implies that it does not matter whether we use a permutation symmetry ϕ before or after the instantiation of the rule r based on the substitution θ. Now, we are ready to provide an important theorem for a state transition.

Theorem 1: *For any permutation symmetry* $\phi \in Perm(U)$, *the following property is satisfied for any state transition on S:*

$$(s, Ev[r\theta], s') \text{ is defined on } S \Leftrightarrow (\phi(s), \phi(Ev[r\theta]), \phi(s')) \text{ is defined on } S.$$

Proof: ($\Rightarrow$) Assume that the state transition $(s, Ev[r\theta], s')$ is defined on S. Since r is enabled for θ at s, all predicates of $Pre[r\theta]$ are included in s from Definition 3. From (a) and (b) of Definition 9, all predicates of $\phi(Pre[r\theta])$ are clearly included in $\phi(s)$. From Lemma 2, all predicates of $Pre[r\phi(\theta)]$ are included in $\phi(s)$. This implies that r is enabled for $\phi(\theta)$ at $\phi(s)$.

Hence, from Definition 3 and Lemma 2, the next state s^* of $\phi(s)$ is obtained as follows:

$$\begin{aligned} s^* &= \phi(s) - Pre[r\phi(\theta)] + Post[r\phi(\theta)] \\ &= \phi(s) - \phi(Pre[r\theta]) + \phi(Post[r\theta]) \\ &= \phi(s - Pre[r\theta] + Post[r\theta]) = \phi(s') \end{aligned}$$

Therefore, a state transition $(\phi(s), Ev[r\phi(\theta)], \phi(s'))$ is defined on S. From Lemma 2, $(\phi(s), Ev[r\phi(\theta)], \phi(s')) = (\phi(s), \phi(Ev[r\theta]), \phi(s'))$, as required.

($\Leftarrow$) It follows by means of inverse ϕ^{-1}.

Theorem 1 clearly explains the hypothesis discussed in Section 4.1 is true. That is, if we have a state transition $t=(s,e,s')$ on S, then we can confirm that all of t's symmetric transitions $(\phi(s), \phi(e), \phi(s'))$'s are defined on S. We can easily extend Theorem 1 for the sequences of state transitions.

Theorem 2: *Let* $S = \langle U, V, P, E, R, s_0 \rangle$ *be a service specification. For any permutation symmetry* $\phi \in Perm(U)$, *all of the following properties hold.*

(i) a sequence of transitions $(s_1,e_1,s_2), (s_2,e_2,s_3), \ldots, (s_n,e_n,s_{n+1})$ *is defined on S.* $\Leftrightarrow$
a sequence of transitions $(\phi(s_1),\phi(e_1),\phi(s_2)), (\phi(s_2),\phi(e_2),\phi(s_3)), \ldots, (\phi(s_n),\phi(e_n),\phi(s_{n+1}))$ *is defined on S.*

(ii) s' is reachable from s. $\Leftrightarrow$ $\phi(s')$ *is reachable from* $\phi(s)$.

(iii) If $s_0=\phi(s_0)$ *then [* $s \in RS(S) \Leftrightarrow \phi(s) \in RS(S)$ *].*

Proof: Property(i) follows by repeated use of Theorem 1. Property(ii) is a direct consequence of Property(i). Property(iii) follows from Property(ii) supposing that $\phi(s_0)=s_0$.

Thus, if we have only one transition sequence τ on S, then we can confirm the existence of all transition sequences symmetrical with τ by Theorem 2.

4.3. Self Symmetry on Initial State

If the initial state s_0 is *symmetrical with itself*, i.e., for any $\phi \in Perm(U)$, $s_0=\phi(s_0)$, then we can utilize the strong property $s \in RS(S) \Leftrightarrow \phi(s) \in RS(S)$ by Theorem 2(iii). This means that if we know a state s is reachable from the initial state, all of states symmetrical with s are also reachable. Generally, in the telecommunication services, it is reasonable to assume that all users are idle at the initial state. That is, we may describe s_0 as follows:

$$s_0 = idle(a_1), idle(a_2), \dots, idle(a_n).$$

Clearly, it satisfies $s_0=\phi(s_0)$ for any $\phi \in Perm(U)$. However, a question might arise: "How should we deal with the case that each user subscribes to the different supplementary service?" For example, consider the case that user A is a subscriber of CW (Call Waiting), B subscribes to DT (Denied Termination) and C is a terminal on a police station which uses EMG (Emergency Call). Then, the initial state s_0 might be:

$$s_0 = idle(A), idle(B), idle(C), CW(A), DT(B), EMG(C).$$

which is asymmetrical with itself. In order to cope with this case, we put on an assumption that all users *can* subscribe to any services. This assumption is quite reasonable since we never see such a situation that only user C cannot subscribe to a service X. Based on this assumption, we start with an initial state in which none of users subscribe to any services yet. Also, we add the following two rules to each service X for the registration/withdrawal of X.

X-reg:	*idle(x), yet-X(x)*	*[regist-X(x)]*	*idle(x), X(x).*
X-wdr:	*idle(x), X(x)*	*[wdraw-X(x)]*	*idle(x), yet-X(x).*

yet-X(x) means that user x does not yet subscribe to the service X, while $X(x)$ means x is a subscriber of X. X might be CW, DT, EMG and so on. By doing this, we can obtain an initial state in which each of users has a possibility to subscribe to any service, for example:

$$s_0 = idle(A), idle(B), idle(C), \; yet\text{-}CW(A), yet\text{-}CW(B), yet\text{-}CW(C), \; yet\text{-}DT(A), \\ yet\text{-}DT(B), yet\text{-}DT(C), \; yet\text{-}EMG(A), yet\text{-}EMG(B), yet\text{-}EMG(C)$$

which clearly satisfies $s_0=\phi(s_0)$ for any $\phi \in Perm(U)$, as required.

Using this consideration, in the following we assume that $s_0=\phi(s_0)$ for any $\phi \in Perm(U)$ for a given service specification $S = \langle U, V, P, E, R, s_0\rangle$.

4.4. Symmetric Reachability Graph SRG

Here, we define a reachability graph called symmetric reachability graph SRG.

Definition 10: Let $S = \langle U, V, P, E, R, s_0\rangle$ be a service specification. A *symmetric reachability graph* for a given S is a labeled directed graph $SRG(S) = \langle N, L, T\rangle$ such that

(a) $N = \{[s] \mid s \in RS(S)\}$ (that is, N is a partition of $RS(S)$),

(b) L is a set of instances of events, and

(c) $T = \{([s], Ev[r\theta], [s']) \mid (s, Ev[r\theta], s^*)$ *is defined on* S *and* $s^* \in [s']\}$.

In SRG, each node represents a symmetric class $[s]$ with a representative (state) s, and each arc outgoing from $[s]$ represents a state transition which occur at the representative s. Also, from the assumption in Section 4.3, Theorem 2(iii) and the above (a), each node $[s]$ in SRG(S) implies that all states belonging to the class $[s]$ are reachable from the initial state s_0.

The following algorithm constructs a SRG. In the algorithm, we define *Waiting* to be a set of nodes.

SRG construction algorithm.

Input: A service specification $S = \langle U, V, P, E, R, s_0\rangle$

Output: A symmetric reachability graph $SRG(S) = \langle N, L, T\rangle$

Procedure:

```
Waiting := {[$s_0$]} ; N := {[$s_0$]}
Repeat
    select [$s$] from Waiting ;
    for any $r \in R$ which is enabled for some $\theta$ at $s$ {
        generate the next state $s'$ by applying $r$ to $s$
        add $Ev[r\theta]$ to $L$ ;
        if $[s^*] \notin N$ s.t. $s' \in [s^*]$ then {
              add $[s']$ to $N$ ; add $[s']$ to Waiting ;
              add $([s], Ev[r\theta], [s'])$ to $T$ ; }
        else   add $([s], Ev[r\theta], [s^*])$ to $T$ ; }
    delete $[s]$ from Waiting ;
Until Waiting = $\emptyset$
```

Example 6: Figure 3 shows a symmetric reachability graph SRG for POTS specification in Figure 1. Each of nodes N4, N5, N6 and N7 represents a symmetric class with two states, while each of other nodes represents a class with only one state. For example, N4 actually represents two symmetrical states "dialtone(A),idle(B)" and "dialtone(B),idle(A)", while N0 does one state "idle(A),idle(B)". Compared with FRG in Figure 2, SRG has smaller number of nodes(8 nodes) and edges(20 edges) (though it might be a small reduction in this example).

From Theorems 1, 2 and Definition 10, the following lemma holds:

Lemma 3: *The following properties are satisfied for SRG(S).*
(i) Each directed path p in SRG(S):

$$p = ([s_1], e_1, [s_2]),\ ([s_2], e_2, [s_3]), \ldots, ([s_n], e_n, [s_{n+1}])$$

has, for each state $s_1{}^* \in [s_1]$, *a corresponding sequence of state transitions* τ *on S:*

$$\tau = (s_1{}^*, \phi_1(e_1), s_2{}^*),\ (s_2{}^*, \phi_2(e_2), s_3{}^*), \ldots, (s_n{}^*, \phi_n(e_n), s_{n+1}{}^*),$$

where $\phi_i \in Perm(U)$ *and* $s_i{}^* \in [s_i]$ $(1 \le i \le n+1)$.
(ii) Each sequence of transitions τ *on S:*

$$\tau = (s_1, e_1, s_2),\ (s_2, e_2, s_3), \ldots, (s_n, e_n, s_{n+1}),\ \text{where } s_1 \in RS(S)$$

has a corresponding directed path in SRG(S):

$$p = ([s_1{}^*], \phi_1(e_1), [s_2{}^*]),\ ([s_2{}^*], \phi_2(e_2), [s_3{}^*])), \ldots, ([s_n{}^*], \phi_n(e_n), [s_{n+1}{}^*]),$$

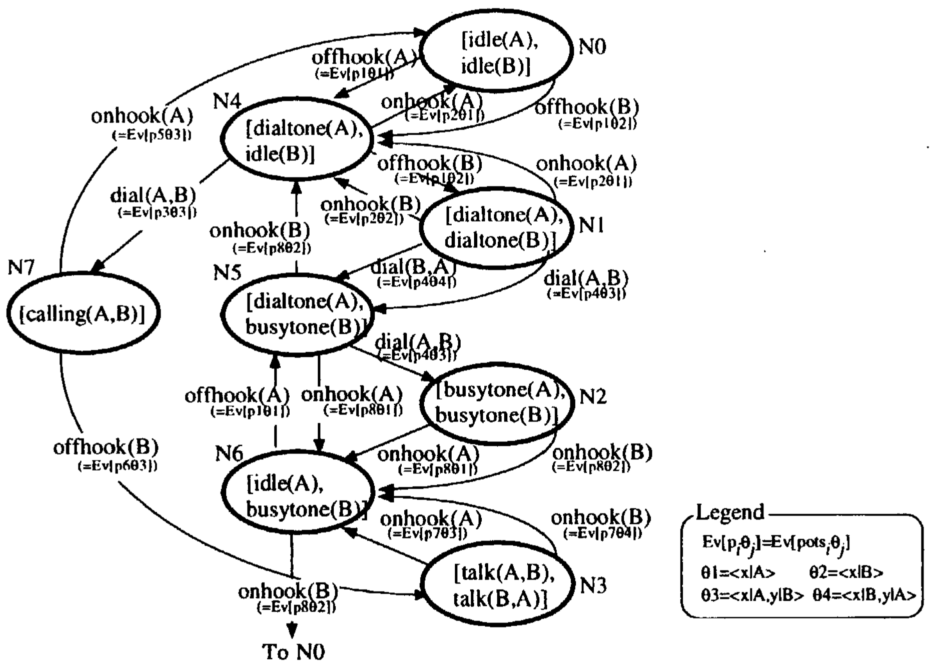


Figure 3. Symmetric reachability graph for POTS specification

where $\phi_i \in Perm(U)$ *and* $s_i^* = \phi_i(s_i)$ $(1 \le i \le n+1)$.

Lemma 3(i) tells us that a directed path from $[s]$ to $[s']$ in SRG(S) implies the existence of a sequence of transitions on S from each $s^* \in [s]$ to some $s'^* \in [s']$. On the other hand, Lemma 3(ii) tells us that a sequence of transitions from s to s' on S implies the existence of a directed path from $[s^*]$ to $[s'^*]$ in SRG(S), where $s \in [s^*]$ and $s' \in [s'^*]$.

4.5. Proof Rules for Interaction Detection

Using the symmetric reachability graph SRG, we can identify the undesirable states in Definition 5 as follows:

Theorem 3: *The following properties are satisfied for SRG(S).*

(a) $[s]$ is a terminal. $\Leftrightarrow$ each $s^* \in [s]$ is a deadlock state.

(b) there exists a directed cycle starting from $[s]$, and there exists no directed path from $[s]$ to $[s_0]$. $\Leftrightarrow$ each $s^* \in [s]$ is a loop state.

(c) $[s]$ has a pair of outgoing edges $([s], e_1, [s'])$ and $([s], e_2, [s''])$ such that $e_1 = e_2$. $\Leftrightarrow$ each $s^* \in [s]$ is a non-deterministic state.

Proof: Properties (a)(b) directly follow from Lemma 1 and Lemma 3.

Property (c) : ($\Rightarrow$) Assume that there exists a pair of the edges $([s], e, [s'])$ and $([s], e, [s''])$ in SRG(S). From Definition 10, there exists a pair of transitions (s,e,ss'), (s,e,ss'') such that $s \in RS(S)$, $ss' \in [s']$, $ss'' \in [s'']$. From Theorem 1, for any $\phi \in Perm(U)$, there exists a pair of transitions $(\phi(s),\phi(e),\phi(ss'))$, $(\phi(s),\phi(e),\phi(ss''))$. Since $s \in RS(S)$, we have $\phi(s) \in RS(S)$ from the assumption in Section 4.3 and Theorem 2(iii). Hence, from Definition 5, $s^* = \phi(s) \in [s]$ is a non-deterministic state, as required.

($\Leftarrow$) Since s* is a non-deterministic state, there exists a pair of transitions (s^*,e,ss'), (s^*,e,ss'') such that $s^* \in RS(S)$. From Definition 10(a), there exists a node $[s]$ such that s = $\phi(s^*)$ in SRG(S). From Theorem 1, there exists a pair of transitions $(\phi(s^*),\phi(e),\phi(ss'))$, $(\phi(s^*),\phi(e),\phi(ss''))$ on S. From Definition 10(c), SRG(S) has a pair of edges $([\phi(s^*)],\phi(e),[s'])$, $([\phi(s^*)],\phi(e),[s''])$, where $\phi(ss') \in [s']$, $\phi(ss'') \in [s'']$. Thus, SRG(S) has a pair of edges $([s], e^*, [s'])$, $([s], e^*, [s''])$, as required.

Thus, in order to detect the interactions between two specifications S_1 and S_2, we first construct SRG($S_1 \oplus S_2$), then identify the undesirable states using Theorem 3.

5. Experimental Evaluation

5.1. Preliminaries

We have performed two experiments in order to evaluate the interaction detection using SRG. For the experiments, we have developed a software constructing both SRG and FRG for any given rule-based specification. Also, based on [15][16], we have prepared the rule-based specifications for the following six practical services (features): CW (Call Waiting), CF (Call Forwarding), DC (Direct Connect), DO (Denied Origination), DT (Denied Termination) and EMG (Emergency call). All of them commonly include POTS. The experiments have been performed on the UNIX workstation Sun SS-UA1.

5.2. Experiment 1(effectiveness and efficiency)

The objective of Experiment 1 is to evaluate the proposed method from the following two viewpoints:

(a) *effectiveness*: whether SRG can exactly identify all interactions detected by FRG,

(b) *efficiency*: how much reduction is attained by using SRG.

First, we checked whether each of six specifications prepared in Section 5.1 is safe or not by constructing FRG (and SRG). As a result, we have found that all services except EMG are safe, while EMG contains the loop states, which is known as the interaction of EMG with itself.* Next, we combined each pair of the remaining five services, then tried to detect the interactions between any two services by constructing FRG and SRG. For all specifications, the number of users is assumed to be three. Table 1 summarizes the result.

In the table, DLK, LOP and NDT denote the existence of deadlock states, loop states, non-deterministic states, respectively. |N| and |T| represent the number of nodes and edges in the graph, respectively. Time(s) represents the execution time of the software for the construction of the graph.

From the table, it can be seen that all interactions detected by FRG are exactly identified by SRG. Also, we see that SRG attains about 80% reduction of FRG in both the graph size and the execution time (e.g., as for the number of nodes in CW+CF, 1-17610/102746=0.83). From these, we can say that our detection method by means of SRG attains an adequate cost reduction for the practical interaction detection with preserving the effectiveness of FRG.

5.3. Experiment 2(scalability)

In order to investigate the applicability of the proposed method to more complex services with many users, we compare SRG with FRG from the following viewpoints:

(a) scalability w.r.t. # of users: impact of the number of users on the graph size.

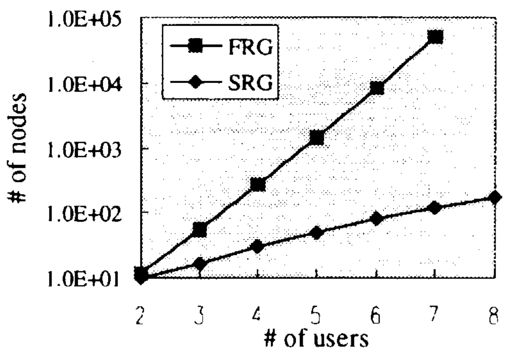
(b) scalability w.r.t. # of features: impact of the number of features on the graph size.

First we compare the scalability w.r.t. # of users. Concretely, for a fixed service specification, we observe the growth of the graph size by varying the number of users. We have selected the POTS specification described in Figure 1 and varied the number of users from 2 to 8. Figure 4 shows the result. Note the logarithmic scale on the vertical axis. It can be seen that, for the increase of the number of users, the size of FRG exponentially grows.

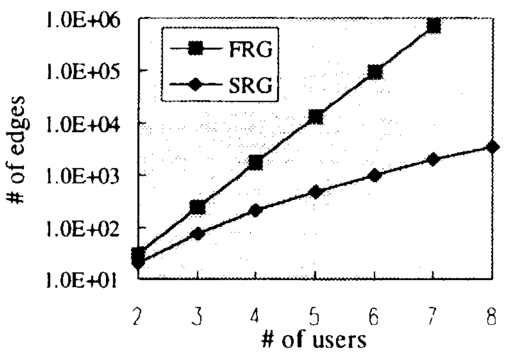
Table 1. Result of Experiment 1

Service	FRG(S)						SRG(S)					
Spec. S	DLK	LOP	NDT	|N|	|T|	Time(s)	DLK	LOP	NDT	|N|	|T|	Time(s)
CW+CF			x	102746	446124	4706.4			x	17610	76732	913.8
CW+DC				9592	38424	270.4				1684	6838	55.6
CW+DT			x	7120	39036	187.2			x	1344	7470	43.9
CW+DO				3480	16560	89.3				668	3234	20.7
CF+DC				65410	243876	1851.5				11065	41456	364.7
CF+DT			x	38584	206868	1006.1			x	6662	35902	213.2
CF+DO				17775	80538	447.6				3087	14079	93.5
DC+DT				5390	27510	69.6				954	4956	17
DC+DO			x	4654	23490	57.9			x	820	4202	14.2
DT+DO				1450	9180	15.7				300	1936	4.9
EMG		x		522	2802	3.5		x		116	646	1.3

* Suppose that both A and B are the subscribers of EMG, and that they are talking. Then, neither A and B can disconnect the call, and the call is trapped in a loop between the talking state and the hold state.

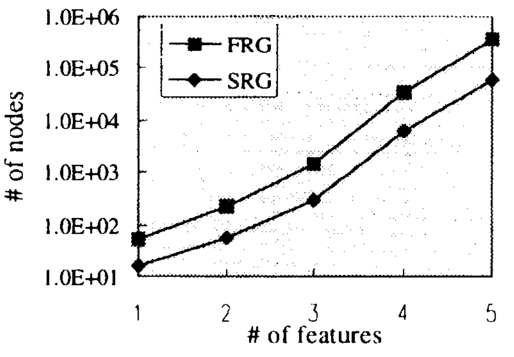


(a)

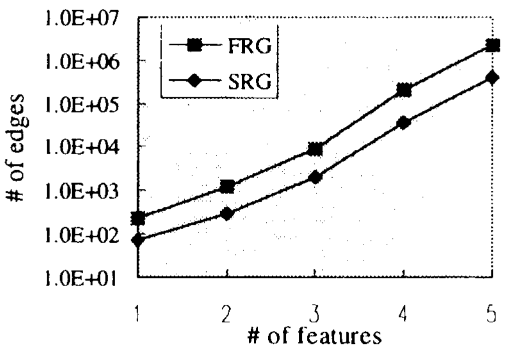


(b)

Figure 4. Graph size w.r.t. # of users: (a) number of nodes (b) number of edges



(a)



(b)

Figure 5. Graph size w.r.t. # of features: (a) number of nodes (b) number of edges

And finally, we couldn't construct FRG with 8 users due to the memory overflow. On the other hand, the size of SRG grows much more slowly than that of FRG. The larger the number of users is, the more SRG attains the significant reduction. From this, we can say that our detection method by means of SRG is much more scalable than that of FRG with respect to the number of users.

Next, we evaluate the scalability w.r.t. # of features. For this, we first prepare the POTS specification and fix the number of users (here we set it to be three). Then, by incrementally adding the specifications DO, DT, DC, EMG to POTS in this order, we observe how the graph size grows for each addition of the feature.

Figure 5 shows the result. Unfortunately, for the increase of the number of features, the size of SRG exponentially grows as well as the size of FRG. However, it can be seen that SRG constantly achieves more than 80% reduction of FRG (e.g., as for the number of nodes with 5 features, 1-58832/348868=0.83). In this sense, SRG is slightly more scalable than FRG with respect to the number of features.

From Experiment 2, we conclude that the proposed detection method using SRG is more scalable than the conventional methods using FRG especially for the increase of the number of users. As a result, it can be applied to the interaction detection for more complex services with many users.

6. Discussion and future works

In this paper, we have proposed a new interaction detection method using the permutation symmetry. By using the proposed SRG, we can realize not only exact interaction detection for given service specifications, but also efficient state space reduction especially for the increase of the number of users. The scalability with respect to the number of features is unfortunately not so good. We consider this is because we put an assumption that every user can subscribe to any service, in order to eliminate asymmetry on the initial state. We are now extending the proposed method so that SRG can deal with the asymmetric initial state by focusing the *subset* of all permutations. This will enable us to choose any initial state as we like. Then, regarding the symmetric initial state (presented in this paper) as a union of several states, we try to divide the original problem into several smaller subproblems. We expect this divide-and-conquer approach will significantly improve the scalability with respect to the number of users.

We discussed only three types of interactions in this paper. However, SRG can be surely applied to the detection of any other types of interactions if the interactions can be defined on the finite state machine. The application to other interactions is also one of our future work.

References

[1] Cameron, E.J., Griffeth, N.D., Lin, Y-J., Nilson, M.E., Schnure W.K. and Velthuijsen, H., "A feature interaction benchmark for IN and Beyond," *Proc. of Second Workshop on Feature Interactions in Telecommunications Systems*, pp.1-23, IOS Press 1994.

[2] Cameron, E.J., Cheng K., Lin, F-J, Liu, H. and Pinheiro B, "A formal AIN service creation, feature interactions analysis and management environment: an industrial appliction", *Proc. of Fourth Workshop on Feature Interactions in Telecommunications Systems*, pp.342-346, July, 1997.

[3] Capellmann, C., Combes, P., Pettersson., J., Renard, B. and Ruiz, J.L., "Consistent interaction detection - A comprehensive approach integrated with service creation", *Proc. of Fourth Workshop on Feature Interactions in Telecommunications Systems*, pp.183-197, July, 1997.

[4] Gammelgaard, A. and Kristensen E.J., "Interaction detection, a logical approach," *Proc. of Second Workshop on Feature Interactions in Telecommunications Systems*, pp.178-196, 1994.

[5] Harada, Y., Hirakawa, Y., Takenaka, T. and Terashima, N., "A conflict detection support method for telecommunication service descriptions," *IEICE Trans. Commun.*, Vol. E75-B, No.10, Oct., 1992.

[6] Holzmann, G.J., "The model cheker SPIN", *IEEE Trans. on Software Engineering*, Vol.23, No.5, pp.279-295, May 1997.

[7] Jensen, K., "Coloured Petri Nets," *EATCS Monographs on Theoretical Computer Science*, Vol1-2, Springer Verlag, 1992.

[8] Khoumsi, A., "Detection and resolution of interactions between services of telephone networks", *Proc. of Fourth Workshop on Feature Interactions in Telecommunications Systems*, pp.78-92, July, 1997.

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

[10] McMillan, K.L., "Symbolic Model Checking", Kluwer Academic Publishers, 1993.

[11] Nakamura, M., Kakuda Y, and Kikuno T., "Petri-net based detection method for non-deterministic feature interactions and its experimental evaluation," *Proc. of Fourth Workshop on Feature Interactions in Telecommunications Systems*, pp.138-152, July, 1997.

[12] Ohta, T. and Harada Y., "Classification, detection and resolution of service interactions in telecommunication services," *Proc. of Second Workshop on Feature Interactions in Telecommunications Systems*, pp.60-72, 1994.

[13] Tsang, S., Magill, E.H. and Kelly, B., "An investigation of the feature interaction problem in networked multimedia services", *Proc. of Third Communication Networks Symposium*, pp.58-61, 1996.

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

[15] ITU-T Recommendations Q.1200 Series., "Intelligent Network Capability Set 1 (CS1)", Sept. 1990.

[16] Bellcore, "LSSGR Features Common to Residence and Business Customers I, II, III," Issue 2, July 1987.