Using Symbolic Model Checking to Detect Service Interactions in Telecommunication Services

Takayuki Hamada, Tatsuhiro Tsuchiya, Masahide Nakamura, and Tohru Kikuno

Department of Informatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
{t-hamada,t-tutiya,masa-n,kikuno}@ics.es.osaka-u.ac.jp

Abstract. Feature interaction is a kind of inconsistent conflict between multiple communication services. In this paper we present an automatic method for detecting feature interactions in service specifications. This method is based on symbolic model checking which can perform verification by symbolically representing the search space with binary decision diagrams. Experimental results show that the method outperforms a previous method based on explicit state traversal, in terms of time and memory required for detection.

1 Introduction

Feature interaction is a kind of inconsistent conflict between multiple communication services, which was never expected from the single services' behavior. In practical service development, the analysis of interactions has been conducted in an ad hoc manner by subject matter experts. This leads to time-consuming service design and testing without any interaction-free guarantee.

To tackle this problem, we propose a formal approach for detection of feature interaction. The detection process checks if interactions occur or not between given multiple services. The proposed approach uses *symbolic model checking* as its basis.

Model checking is a powerful technique for verifying systems that are modeled as a finite state machine. In model checking, the properties to be checked are expressed in temporal logic. For realistic designs, the number of states of the system can be very large and the explicit traversal of the state space may become infeasible. *Symbolic model checking* has proven to be successful for overcoming this problem. This method uses Boolean functions to represent the state space. Since Boolean functions can be often represented by Ordered Binary Decision Diagrams (OBDDs) very compactly, the symbolic model checking method can reduce the memory and time required for analysis. By manipulating the Boolean functions, the method can determine whether or not a system meets a given property that is specified using CTL [1], a branching time temporal logic.

I. Chong (Ed.): ICOIN 2002, LNCS 2343, pp. 641–651, 2002.

In this paper, we investigate how we can detect feature interactions by using SMV, a well-known symbolic model checking tool. We propose a systematic method for translating given specifications of telecommunication services into the input language of SMV. Using this method, automatic detection of feature interactions can be carried out. To illustrate the effectiveness of the approach, we show the results of applying it to the specifications of practical telecommunication services.

Plath and Ryan [10] also proposed the use of SMV for feature interaction detection. Their work considered more detailed specifications than ours, but it entails describing different CTL formulae by hand, depending on the services and properties to be checked.

In contrast, our method adopts a more abstract model, thus allowing us to represent four major types of feature interactions by only two formulae. The downside of adopting a high level model is that only non-subtle interactions can be detected, which might easily be resolved in a low-level design by, for example, prioritizing services. We think, however, that the method is still useful, since knowing the possibility of such interactions can help identify error-prone parts of the design.

2 Preliminaries

In order to formalize the feature interaction detection problem, we present fundamental definition in this section.

2.1 Services

For the formalization, we have to first prepare the services. From ITU-T recommendation [11] (*ITU-T Recommendations Q.1200 Series* - Intelligent Network Capability Set 1 (CS1)) and Bellcore's feature standards [12] (*Bellcore* - LSSGR Features Common to Residence and Business Customers I,II,III), we have selected the following seven services (features):

Call Forwarding (CF): This service allows the subscriber to have his incoming calls forwarded to another number. Suppose that x subscribes to CF and that x specifies y to be a forwarding address. Then, any incoming call to x is automatically forwarded to y.

Originating Call Screening (OCS): This service allows the subscriber to specify that outgoing calls be either restricted or allowed according to a screening list. Suppose that x subscribes OCS and that x puts y in the OCS screening list. Then, any outgoing call to y from x is restricted, while any other call from x is allowed. Suppose that x receives dialtone. At this time, even if x dials y, x receives busytone instead of calling y.

Terminating Call Screening (TCS): This service allows the subscriber to specify that incoming calls be either restricted or allowed according to a screening list. Suppose that x subscribes TCS and that x puts y in the TCS

screening list. Then, any incoming call from y to x is restricted, while any other call to x is allowed. Suppose that y receives dialtone. At this time, even if y dials x, y receives busytone instead of calling x.

Denied Origination (DO): This service allows subscriber to disable any call originating from the terminal. Only terminating calls are permitted. Suppose that x subscribes to DO. Then, any outgoing call from x is restricted. Even if x offhooks when the terminal is idle, x receives busytone instead of dialtone.

Denied Termination (DT): This service allows subscriber to disable any call terminating at the terminal. Only originating calls are permitted. Suppose that x subscribes to DT. Then, any incoming call to x is restricted. Even if another user y dials x, y receives busytone without calling x.

Direct Connect (DC): This service is a so-called *hot line* service. Suppose that x subscribes to DC and that x specifies y as the destination address. Then, by only offhooking, x is directly calling y. It is not necessary for x to dial y.

Emergency Call (EMG): This service is usually deployed on police and fire stations. In the case of an emergency incident, the call will be held even when the caller mistakenly onhooks. Suppose that x is a police station on which EMG is deployed, and that y has made a call to x and is now busy talking with x. Then, even when y onhooks, the call is on hold without being disconnected. Followed by that, if y offhooks, the held line reverts to a connected line and y can talk with x again. In order to disconnect the call, x has to onhook.

2.2 Specifications

To formalize the feature interaction detection problem, we have to describe services in a certain way. There are a number of researches concerning service description to formulate the interaction problem. In this paper, we adopt a variant of State Transition Rules (STR) [5,9], a rule-based service specification language. Other examples of such a language include, for example, declarative transition rules [3].

Notation. First, we define the syntax notation of the specification. 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.
(f) s_0 is the (*initial*) state.

Each rule $r \in R$ is defined as follows:

$$r : pre{-}condition\ [event]\ post{-}condition.$$

Pre(post)−condition is a set of *predicates* $p(x_1, \ldots, x_k)$'s, where $p \in P$,$x_i \in V$ and k is called *arity* which is a fixed number for each p. Especially, precondition can include *negations* of predicates such as $\neg p(x_1, \ldots, x_k)$'s which implies $p(x_1, \ldots, x_k)$ does not hold. *Event* is a predicate $e(x_1, \ldots, x_k)$, where $e \in E$, $x_i \in V$.

Figure 1 shows an example of a specification. This specification represents the Plain Old Telephone Service (POTS). Additional communication features, such as those described in the previous subsection, can be described by modifying this specification (for example, adding rules or predicate symbols).

$U = \{A, B\}$
$V = \{x, y\}$
$P = \{idle, dialtone, calling, path, busytone\}$
$E = \{onhook, offhook, dial\}$
$R = \{$
$pots1 : idle(x)\ [offhook(x)]\ dialtone(x).$
$pots2 : dialtone(x)\ [onhook(x)]\ idle(x).$
$pots3 : dialtone(x)\ ,\ idle(y)\ [dial(x, y)]\ calling(x, y).$
$pots4 : dialtone(x)\ ,\ \neg idle(y)\ [dial(x, y)]\ busytone(x).$
$pots5 : calling(x, y)\ [onhook(x)]\ idle(x)\ ,\ idle(y).$
$pots6 : calling(x, y)\ [offhook(y)]\ path(x, y)\ ,\ path(y, x).$
$pots7 : path(x, y)\ ,\ path(y, x)\ [onhook(x)]\ idle(x)\ ,\ busytone(y).$
$pots8 : busytone(x)\ [onhook(x)]\ idle(x).$
$pots9 : dialtone(x)\ [dial(x, x)]\ busytone(x).$
$\}$
$s_0 = \{idle(A), idle(B)\}$

Fig. 1. Rule-based specification for POTS.

State Transition Model. Next, we define the state transition specified by the rule-based specification.

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 r based on 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. We represent pre-condition, event and post-condition of rule r as $Pre[r]$, $Ev[r]$ and $Post[r]$, respectively.

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

Let s be a state. We say that rule r is *enabled* for θ at s, denoted by $en(s, r, \theta)$, iff all instances in $Pre[r\theta]$ hold at s (i.e., all instances are included in s). Let $\hat{Pre}[r\theta]$ be the subset of $Pre[r\theta]$ that is obtained by removing all negations of

instances of predicates from $Pre[r\theta]$. When $en(s, r, \theta)$ holds, the *next state*, s' of s, can be generated by deleting all instances in $\hat{P}re[r\theta]$ from s and adding all instances in $Post[r\theta]$ to s; that is,

$$s' = (s \backslash \hat{P}re[r\theta]) \cup Post[r\theta]$$

At this time, we say a *state transition* from s to s' caused by an event $Ev[r\theta]$ is defined on S.

Example 1. Suppose that $r = pots4$ in Figure 1, $\theta = \langle x|A, y|B\rangle$ and $s = \{dialtone(A), dialtone(B)\}$. At this time, $Pre[r\theta] = \{dialtone(A), \neg idle(B)\}$, $Post[r\theta] = \{bustytone(A)\}$ and $en(s, pots4, \theta)$ holds. If subscriber A dials B, then a state transition occurs, thus resulting in $s' = \{busytone(A), dialtone(B)\}$.

2.3 Feature Interactions

In this paper, we focus primarily on the following three types of interactions. These are very typical cases of interactions and are discussed in many papers (e.g., [2,3,4,6,9]):

- **deadlock:** Functional conflicts of two or more services cause a mutual prevention of their service execution, which result in a deadlock.
- **loop:** The service execution is trapped into a loop from which the service execution never returns to the initial state.
- **violation of invariant:** The invariant property, which is asserted by each service, is violated by the service combination.

Example 2. (Deadlock) Suppose that both A and B subscribe to *EMG* and are talking to each other. Here, if A onhooks, the call is on hold by B's *EMG*. At this time, if A offhooks, the call reverts to the talking state. On the other hand, if B onhooks, the call is also held by A's *EMG* without being disconnected. Symmetrically, this is true when B onhooks first. Thus, neither A nor B can disconnect the call. As a result, the call falls into a trap from which it never returns to the idle state.

Example 3. (Violation of invariant) Suppose that (1)A is an *OCS* subscriber who restricts the outgoing calls to C, and (2)B is *CF* subscriber who sets the forwarding address to C. At this time, if A dials B, the call is forwarded to C, so A will be calling C. This nullifies A's call restriction to C.

3 Proposed Method

3.1 SMV Programs

SMV (Symbolic Model Verifier)[7] is a software tool for symbolic model checking; it is publicly available and has been especially successful in verifying hardware

systems. In this section, we describe how we can use SMV to detect feature interactions.

In SMV, services (features) are described in a special language called the *SMV language*. We refer to a service description written in the SMV language as an *SMV program*.

An SMV program describes both the state space and the property to be verified. The property is expressed in a temporal logic called CTL (Computation Tree Logic). The model checker extracts a state space and a transition system represented as an OBDD from the program and uses an OBDD-based search algorithm to determine whether the system satisfies the property. If the property does not hold, the verifier will produce an execution trace that shows why the property is falsified.

```
MODULE main
VAR   request:boolean;
      state:{ready, busy};
INIT  state = ready
TRANS (state = ready & request)
      & next(state) = busy
SPEC  AG(request -> AF state = busy)
```

Fig. 2. An SMV program.

Figure 2 shows an example of an SMV program. The keyword `VAR` is used to declare variables. The variable `request` is declared to be a Boolean in the program, while the variable `state` can take on the symbolic values `ready` or `busy`.

The property to be checked is described as a formula in CTL under the keyword `SPEC`. The SMV model checker verifies that all possible initial states satisfy the CTL formula. In this case, the property is that invariantly if `request` is true, then eventually the value of `state` is `busy`.

In this example, the transition relation is specified directly by a Boolean formula over the current and next versions of the state variables. Similary, the set of initial state is specified by another Boolean formula over the current version of state variables. These two formulas are accomplished by the `TRANS` and `INIT` statements, respectively.

The initial states are a set of states where the Boolean formula defined in the `INIT` statement holds. The transition relation is a set of the pairs of the current state and the next state that satisfy the Boolean formula defined in the `TRANS` statement. The expression `next(x)` is used to refer to the variable `x` in the next state.

3.2 Translating Service Specifications into SMV Programs

In this subsection, we show how to translate a given service specification into an SMV program. This process consists of three steps.

First, necessary variables are declared. Basically, we use one Boolean variable for each instance of a predicate. The variable represents whether or not the corresponding instance of the predicate holds. For example, suppose that $P = \{idle(x), path(x, y)\}$ and $U = \{A, B\}$. Then the variable declaration part will be

```
VAR  idle_A : boolean;  idle_B : boolean;
     path_A_B : boolean;  path_B_A : boolean;
```

The second step is to produce the INIT part. In this part, the initial state is specified by a Boolean formula over the variables that evaluates to true exactly for the initial state. For example, when $s_0 = (idle(A), idle(B))$, the INIT part will be

```
INIT  idle_A = 1 & idle_B = 1 & path_A_B = 0 & path_B_A = 0
```

The third step is to specify the transition relation by giving a Boolean formula over the variables and the next version of the variables.

The formula is expressed by a disjunction of many subformulas each of which represents an instance of each rule. Given an instance i of a rule, its corresponding formula F_i is

$$\bigwedge_{p \in Pre[i]} p \wedge \bigwedge_{p \in Post[i]} p' \wedge \bigwedge_{p \in \hat{P}re[i] \setminus Post[i]} \neg p' \wedge \bigwedge_{p \notin \hat{P}re \cup Post} (p \leftrightarrow p').$$

where p' denotes the next version of an instance p of a predicate. In the SMV language, this formula must be expressed as a formula over the declared variables. For example, consider rule $idle(x)$, $\neg idle(y)$ $[dial(x, y)]$ $path(x, y)$ and substitution $(x, y) = (A, B)$. Then the above formula is represented in SMV as

```
idle_A = 1 & idle_B = 0
& next(idle_A)=0 & next(idle_B)=idle_B
& next(path_A_B)=1 & next(path_B_A)=path_B_A
```

Thus the formula that represents the transition relation is

$$\bigvee_i F_i \vee (\bigwedge_i \neg F_i \wedge \bigwedge_p (p \leftrightarrow p'))$$

The subformula $\bigwedge_i \neg F_i \wedge \bigwedge_p (p \leftrightarrow p')$ is necessary, since the transition relation must be *total*; that is, the next state must be specified for any states. This requirement stems from the fact that both the CTL semantics and the CTL model checking algorithm depend on this assumption. Intuitively, the subformula signifies that if no transition is possible, then the next state will be the same as the current state.

3.3 CTL Formulas

The property to be verified by model checking must be described in CTL. CTL is a branching time temporal logic. Here we only use two temporal operators: ***AG*** and ***EF***.

The formula ***AG*** p holds in state s iff p holds in all states along all sequences of states starting from s. Clearly, the invariant property is expressed in CTL as ***AG*** I where I is an invariant property intended to be satisfied.

EF p holds in state s iff p holds in state s if p holds in some state along some state sequence starting from s. Thus, the freedom from deadlock and loop is described as CTL formula ***AG EF*** $initial\_state$, where $initial\_state$ represents the initial state.

4 Experimental Results

In order to evaluate the effectiveness of the proposed method, we conducted the experimental evaluation through interaction detection for practical services. For comparison purposes, we used two methods: the proposed method, which analyzes the state space symbolically, and a previous method[4], which searches all reachable states explicitly from the initial state.

For each of the seven services prepared in the previous section, we have created a rule-based service specification. In the following, we attempt to provide a reasonable invariant property intended to be satisfied. We let I_X denote the invariant property for service X.

CF: There is no invariant property respected for CF. Therefore, we give an invariant formula $I_{CF} = true$.

OCS: A reasonable invariant property is considered to be "If x puts y in the OCS screening list (denoted by $OCS(x, y)$), x is never calling y at any time". Therefore, we give an invariant formula $I_{OCS} = \neg OCS(x, y) \vee \neg calling(x, y)$.

TCS: A reasonable invariant property is considered to be "If x puts y in the TCS screening list (denoted by $TCS(x, y)$), y is never calling x at any time". Therefore, we give an invariant formula $I_{TCS} = \neg TCS(x, y) \vee \neg calling(y, x)$.

DO: A reasonable invariant property is considered to be "If x subscribes to DO (denoted by $DO(x)$), x never receives dialtone at any time". Therefor, we give an invariant formula $I_{DO} = \neg DO(x) \vee \neg dialtone(x)$.

DT: A reasonable invariant property is considered to be "If x subscribes to DT (denoted by $DT(x)$),, y is never calling x at any time". Therefor, we give an invariant formula $I_{DT} = \neg DT(x) \vee \neg calling(y, x)$.

DC: There is no invariant property respected for DC. Therefore, we give an invariant formula $I_{DC} = true$.

EMG: There is no invariant property respected for EMG. Therefore, we give an invariant formula $I_{EMG} = true$.

In the experiment, we put the following assumption.

Table 1. Result of interaction detection.

Service Spec.	Unsafety	Violation
EMG	Detected	None
CF+DC	None	None
CF+DT	None	Detected
CF+DO	None	None
CF+OCS	None	Detected
CF+TCS	None	Detected
DC+DT	None	Detected
DC+DO	None	None
DC+OCS	None	Detected
DC+TCS	None	Detected
DT+DO	None	None
DT+OCS	None	None
DT+TCS	None	None
DO+OCS	None	None
DO+TCS	None	None
OCS+TCS	None	None

(a) All users can subscribe to all services.
(b) At the initial state, all users are idle and no user subscribes to any service yet.

This assumption is quite reasonable for telecommunication services. In order to achieve Assumption (a), a pair of rules for the subscription registration and its withdrawal is added to each service specification.

The experiments have been performed on a Linux workstation with a 700 MHz Pentium III processor and 512MByte memory. We varied the number of users from three to five.

4.1 Results of Detection

First, we check if each of the seven specifications is *safe*, that is, free from deadlock and loop. As a result, we have found that all services except EMG are safe, while EMG contains the loop states as shown in Example 2 which is interaction of EMG itself. Next, we have combined each pair of the remaining six services, then tried to detect the interactions between any two services.

Table 1 summarizes the results. In this table, the column 'Unsafety' shows whether deadlock or loop states are identified (*detected*) or not (*none*), and the column 'Violation' shows whether violating invariant properties states are identified (*detected*) or not (*none*). The results were the same regardless of the number of users.

Table 2. Times required for detection (in seconds)

Service	Proposed method			Previous method		
Spec.	3 users	4 users	5 users	3 users	4 users	5 users
EMG	0.59	118.46	N/A	0.55	13.01	298.00
CF+DC	11.01	N/A	N/A	258.68	N/A	N/A
CF+DT	5.83	825.75	N/A	141.66	N/A	N/A
CF+DO	5.94	1066.58	N/A	62.58	N/A	N/A
CF+OCS	6.93	942.96	N/A	518.22	N/A	N/A
CF+TCS	6.88	931.91	N/A	516.64	N/A	N/A
DC+DT	0.84	12.74	514.94	10.27	1371.97	N/A
DC+DO	0.61	11.13	544.26	8.56	1125.81	N/A
DC+OCS	1.00	20.95	11131.60	39.62	N/A	N/A
DC+TCS	1.07	20.89	3239.47	39.43	N/A	N/A
DT+DO	0.48	5.31	75.61	2.37	67.63	1877.40
DT+OCS	0.64	8.81	929.61	15.84	N/A	N/A
DT+TCS	0.65	8.80	760.80	15.87	N/A	N/A
DO+OCS	0.58	9.34	N/A	9.64	1767.65	N/A
DO+TCS	0.59	9.26	N/A	9.64	1767.82	N/A
OCS+TCS	1.02	16.42	4429.52	63.31	N/A	N/A

4.2 Performance

Next, we evaluate the performance of the proposed method. For each of the two methods, we investigate how much time is needed to perform the interaction detection. The measurement was performed in the same setting in the previous experiment of detection quality.

Table 2 shows the results. In this table, an N/A indicates that data was not collected because of memory shortage.

In this table, one can see that for all combinations of each pair of six specifications, CF, DC, DT, DO, OCS, and TCS, the proposed method outperformed the previous method, in terms of time and memory required to perform the interaction detection. For example, consider the case of DT+TCS. In this case, the proposed method completed the detection process within around 13 minutes when the number of users is five. In contrast, the previous method was not able to carry out detection even when the number of users is four.

Exceptionally, the previous method outperformed the proposed method for specification EMG. This can be explained as follows. The number of reachable states is quite small for the case of EMG. Thus, the previous method can complete detection with very small amount of time. In symbolic model checking, however, an OBDD that represents the transition relation must be constructed before state space traversal. In this case, the OBDD is very large, thus its construction consumes long time, in spite of the small reachable state space.

5 Conclusions

In this paper, we proposed to use symbolic model checking to detect feature interactions in telecommunication. We present a method for translating service specifications into the input language of the SMV system. We implemented this method and, by applying it to practical services, showed the effectiveness of the proposed approach. Future research includes, for example, the examination of other model checking techniques. Specifically, the use of symmetry and partial order equivalence for state space reduction has already proven to be effective when explicit state representation is used [8]. We think that combining these techniques with the proposed approach deserves further study.

References

1. E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal-logic specifications," *ACM Trans. Programming Languages and Systems*, vol.8, no.2, pp.244-263, 1986.
2. R. Dssouli, S. Some, J. W. Guillery, and N. Rico, "Detection of feature interactions with REST," *Porc. of Fourth Workshop on Feature Interactions in Telecommunications Systems*, pp.271-283, July 1997.
3. A. Gammelgaard, E. J. Kristensen, "Interaction detection, a logical approach," *Porc. of Second Workshop on Feature Interactions in Telecommunications Systems*, pp.178-196 1994.
4. Y. Harada, Y. Hirakawa, T. Takenaka, and N. Terashima, "A conflict detection support method for telecommunication service descriptions," *IEICE Trans. Commun.*, vol.E75-B, no.10, Oct. 1992.
5. Y. Hirakawa and T. Takenaka, "Telecommunication service description using state transition rules," *Proc. of IEEE Int'l Workshop on Software Specification and Design*, pp.140-147, Oct. 1991
6. A. Koumsi, "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.
7. K. L. McMillan, *Symbolic Model Checking*, Kluwer Academic, 1993.
8. M. Nakamura and T. Kikuno, "Exploiting symmetric relation for efficient feature interaction detection," *IEICE Trans. on Information and Systems*, vol.E82-D, No. 10, pp.1352-1363, 1999.
9. T. Ohta and Y. Harada, "Classification, detection and resolution of service interaction in telecommunication services," *Porc. of Second Workshop on Feature Interactions in Telecommunications Systems*, pp.60-72 1994.
10. M. Plath and M. Ryan, "Plug-and-play features," In W. Bouma, editor, *Feature Interactions in Telecommunications Systems V*, IOS Press, pp. 150-164, 1998.
11. ITU-T Recommendations Q.1200 Series, *Intelligent Network Capability Set 1 (CS1)*, Sept. 1990.
12. Bellcore, *LSSGR Feature Common to Residence and Business Customers I,II,III*, Issue 2, July 1987.