A Scenario-based Approach for Feature Interaction Detection of Communication Software Systems

Ana Cavalli*, David Lee**, Stephane Maag*

*Institut National des Telecommunications
9 rue Charles Fourier
F-91011 Evry Cedex, France
Email : {Ana.Cavalli, Stephane.Maag}@int-evry.fr
**Bell Laboratories, Research China
15/FAero Space Great Wall Building
30 Hai Dian Nan Lu
Beijing, 100080
China
Email : lee@research.bell-labs.com

Abstract

Feature interactions in communication systems manifest themselves as unexpected behaviors when multiple features are invoked jointly. We present a new approach for feature interaction detection, based on the analysis of scenarios and structures of the systems. We study the theoretical foundation of the feature interaction and propose efficient algorithms for their detection. As a case study, we also present an experiment on a real system in order to illustrate our approach.

Keywords: *Telecommunication system, Feature interaction, Finite State Machine, Formal methods.*

1 Introduction

Researchers have been working on one of the most critical problems in the telecommunications domain: Service Interaction. In order to give an idea of the problem, let us imagine two users A and B, subscribers of the service *Call Forwarding on Busy Line*. The user A forwards his incoming calls to B who forwards his own to A. Now, suppose that A calls B who is on line. What happens to user A ? what happens in the telecommunication network ? The increase in the number of services that we are witnessing makes the resolution of these interaction problems even more complex and crucial. To manage this complexity, different methods and techniques have been proposed: analysis of interactions at the early

user-needs stage, or at the service specification stage, when designing and conceiving the environment supporting the services or at the service implementation stage [3].
As pointed out, for example in [4], several different types of interactions have been observed. This fact complicates the task of finding a universal method to detect all possible interactions. It is therefore necessary to integrate different methods to be applied at the early stages of development [6], as well as, during later stages, when implementing specific environments [1].

In this paper we present an analysis and a formal method for feature interaction detection. In order to create an effective detection algorithm, we work at the upper level of the service life cycle, that is, in the specification phase. In this case, there exist several suitable formalisms to model protocols. For example, extended finite state machines [15], transition systems [16], process algebras [2] and Petri nets [18]. Many works tackle this problem and thus several analysis lead to the creation of detection methods based on different kind of specifications. The study presented in [5] leads to the use of Message Sequence Charts scenarios. However, scenarios usually describe a partial behavior of the system while we would like to perform a global behavior analysis. So, there are methods allowing to detect some interactions (e.g. [11, 13]). Nevertheless, the services are specified from their requirements. In turn, requirements are expressed by properties. Thus we cannot detect each kind of interaction. Indeed, one of the main drawbacks of the properties oriented approaches is that it is often very difficult to describe all the system properties by logic expressions. Furthermore, when we add new features to a system, it is necessary to rewrite several properties. That is why the choice of our study is a formal approach based on finite state machines.

The analysis and detection of interactions use a specification of services based on Extended Finite State Machines (EFSMs). Work involving the same formal concepts is presented in [14]. However, the use of a restriction function on automata deprives the analysis of a specification part. A reason could be that the pruned part is useless to the feature interaction studies. Unfortunately, no satisfying method is given to show that this part is negligible to the feature interaction detection. Our work consists in the detection of interactions based on the comparison of automata traces after some projections. The main contribution of our work consists in the design of a new algorithm for the detection of interactions from the global behavior of a telecommunication system. Furthermore, we try to avoid the increasing number of redundant detections. This algorithm has been implemented and applied to the feature interaction detection on an intelligent network architecture. It describes five services on the Basic Call Service (BCS) implemented in SDL. We present here the results dealing with the Originating Call Service (OCS) and the Call Forward Unconditional (CFU).

The rest of the paper is organized as follows. Section 2 introduces basic concepts used to describe telecommunication system and services definitions. Section 3 presents the feature

interaction definition and algorithms with the application to the case study. Finally, Section 4 concludes the paper.

2 Preliminaries and definitions

The Extended Finite State Machines (in short EFSM) are the basis of the specification processes in SDL (Specification and Description Language). The goal of SDL is to specify system behaviors from the representation of their functional properties and to describe their effective behavior. Therefore, this language is particulary suited to study systems that can be described by using extended finite state machines. However, our technique can also be applied on other mathematic models such as transition systems, labelled transition systems, process algebras and Petri nets. In our work, EFSMs (modeled by using SDL) are deployed into Finite State Machines.

Definition 2.1 A *Finite State Machine* (in short FSM) is a 5-tuple Aut=$(s^0, D, \Sigma, Act_{Aut}, T)$ with:

- $s^0 \in \Sigma$ is the initial state of Aut,
- D is the set of the Aut final states,
- Σ is the set of Aut states,
- Act_{Aut} is the set of Aut actions,
- $T \subset \Sigma \times Act_{Aut} \times \Sigma$ defines the transition set.

□

We call $\mathcal{A}$ the set of the Finite State Machines. An action of a FSM is composed by a primitive and data, as for instance *offhook(A)* which means that user A has offhooked. This notion of Finite State Machine allows us to obtain the description of telecommunication system behaviors from SDL specifications. We use here the usual notion of telecommunication system in the sense that it provides a set $\mathcal{F}$ of features to some customers such as $\mathcal{F} \in \mathcal{P}(\{F_0, F_1, \ldots, F_n\})$. F_0 is the Basic Call Service (BCS) on which the other value-added features F_i $(i \in I = \{1, \ldots, n\})$ are connected. Each service F_i contains a set of feature primitives P_{F_i} which support the transitions in the FSM. For example, the feature primitive *offhook* to the action *offhook*(A) for the BCS, where A is the subscriber. We call *Prim* and *prim* the two functions giving, respectively, the feature primitive of an action and the set of primitives from a set of actions in a finite state machine. For instance, *prim(offhook(A))=offhook*.

Definition 2.2 A *telecommunication system* T_k for $I_k \subset I$ is a finite state machine such as $T_k = BCS \bigoplus_{i \in I_k} F_i$. If $I_k = I$, we call T_k the *global telecommunication system.* □

The connections $\bigoplus$ are defined in the model presented in [17, 20] according to each F_i specification. In our work, we assume that the connections are specified and that the final finite state machines are given.

Next, we introduce some notions concerning undesirable properties of systems. A *livelock* in T_k is a loop that can be reached from the initial state. When the system gets into a livelock, it will loop around indefinitely without any outgoing actions. This is an underisable system behavior. In the case of telecommunication systems the users may not act normally. Another undesirable system behavior is *deadlock.* As a matter of fact, both livelocks and deadlocks are to be checked for system design, and are well studied in protocol validation and verification research [7].

Definition 2.3 We say that a system is *valid* when it is free of deadlocks and livelocks. □

We shall not digress here and proceed with our discussion of feature interaction analysis with the assumption that each telecommunication system is valid.

When we analyze system features, we often mention the notion of *behavior.* However, this behavior is not always formalized. There are several aspects that may define it. For instance, the *functional* aspects, which concern the sequences of possible states/events, and the *non-functional* aspects, with regard to any real time and performance aspects of the behavior. In this paper, we restrict our analysis to functional aspects. Systems will be defined as finite state machines. *Execution sequences* are a good and formal way in order to capture this kind of behavior. Next, we introduce this concept in our framework.

Definition 2.4 For a finite state machine $Aut{=}(s^0, D, \Sigma, Act_{Aut}, T)$, we define an *execution sequence* s as an element of T^*.
For a finite state machine Aut, we represent all its execution sequences by the set $E(Aut)$. □

The behavior of a whole system depends on the different features. In order to analyze feature interactions, we need to perceive them as an observer that tracks only a feature F_i. This is performed with respect to the corresponding feature primitives. Next, we define the appropriate projections where primitives not related with the corresponding set of features are hidden.

Definition 2.5 Let $I_p \subseteq I$ and $a \in Act_{T_p}$. We define the function $\pi_{I_p} : Act_{T_p} \longrightarrow Prim(Act_{T_p}) \cup \{\varepsilon\}$ as:

$$\pi_{I_p}(a) = \begin{cases} prim(a) & \text{if } prim(a) \in \bigcup_{i \in I_p} P_{F_i} \\ \varepsilon & \text{otherwise.} \end{cases}$$

Where ϵ represents the empty-move.

This function may be extended to execution sequences as follows. We consider the function $\Pi_{I_p} : E(T_p) \longrightarrow (Prim(Act_{T_p}) \cup \{\varepsilon\})^*$, defined as

$$\Pi_{I_p}((y_0, a_1, y_1, a_2, y_2, \ldots, y_{m-1}, a_m, y_m, \ldots)) = (y_0, \pi_{I_p}(a_1), y_1, \ldots, y_{m-1}, \pi_{I_p}(a_m), y_m, \ldots).$$

□

We also call Π_{I_p} the *canonical function* applied to execution sequences sets in order to ease the reading of this paper.

Our approach is based on the analysis of sequences of actions that represent the system behavior. In these telecommunication systems, some behaviors may be redundant in the sense that the messages exchanged between the switch and the users are composed by the same primitives. Indeed, when no services are activated, the behaviors are equal either user A calls B or B calls A. That is why, in our work, we use scenarios sets as defined below.

Definition 2.6 Let $E(Aut)$ be the set of execution sequences of Aut. We define $S(Aut)$, the *scenarios set* of Aut, as the set of elements belonging to $E(Aut)$ in which each action is replaced by its primitive by using the function *prim*.

□

Let us remark that the projection functions introduced in Definition 2.5 may be extend to deal with scenarios set in the expected way.

The feature interactions may have different causes and thus many definitions. In our work, feature interaction refers to situations where different features or instances of the same features influence each other. Generally, they can be observed when two telecommunication systems affect each other [12]. This mechanism is developed in the following section.

3 Feature Interaction Detection

In order to analyze feature interactions, we need to obtain the whole studied system behavior. As we have previously mentioned, our method tackles interactions by observing the scenarios from sets of services in a global telecommunication system. In this section, we present a new approach to detect them, and we are specially careful to avoid the increasing number of redundant detections.

3.1 Defining Feature Interactions

Let T_p and T_q be two valid telecommunication systems. We say that T_q has no feature interactions with T_p if and only if the behavior of T_p is present in T_q. In order to satisfy this informal definition, we compare every T_p and T_q scenario such that an interaction is detected when a scenario belonging to $S(T_p)$ is not present in $S(T_q)$ in which the actions whose primitives are in $\bigcup_{i \in I_q \setminus I_p} P_{F_i}$, have been replaced by ϵ. Indeed, in this case, this means that a T_p behavior has been modified. We can formalize all of this as follows.

Definition 3.1 Let $I_p \subset I_q$ be two subsets of I. We say that a valid telecommunication system T_q *has no feature interactions* with a valid telecommunication system T_p if and only if $S(T_p) = \Pi_{I_p}(S(T_q))$. A valid telecommunication system T_q is *interaction-free* if and only if $\forall I_p$ such as $I_p \subset I_q \subseteq I$, we have $S(T_p) = \Pi_{I_p}(S(T_q))$. □

We assume in the previous definition that T_q and T_p are different from the FSM composed only by the Basic Call Service. We show the use of this definition in the next example.

Example 3.1: Let us assume that we have n features F_i in the global telecommunication system and we wish to verify that the system composed by F_1, F_2 and F_3 is interaction-free. We denote by T_3 the finite state machine representing this system. In respect with Definition 3.1, we have 6 I_p sets for $I_q = \{1, 2, 3\}$, that is we need to check six equalities with $I_p \in \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}\}$ upon T_3.

Let us remark that, even if we can decide whether T_q is interaction-free or not, this method does not still allow us to be convinced that it is also *totally interaction-free*, that is, the systems T_p are also interaction-free. In order to analyze all the behaviors, we also need to verify that all these T_p are interaction-free. In Example 3.1, the number of equalities to check is 12. They are the six mentioned above and the cases where T_q is $\{1, 2\}$, $\{1, 3\}$ and $\{2, 3\}$. We notice that the number of checked equalities quickly becomes important if we wish to detect feature interactions among the n features. This number may be reduced by using a lattice structure.

Definition 3.2 We define the complete lattice $L = (\{T_k \mid T_k = BCS \bigoplus_{i \in I_k} F_i,\ I_k \subseteq I\},\ \leq)$ such as:

- the bottom is the empty set $\emptyset$,
- the top is the global telecommunication system GT,
- $T_p \leq T_q$ if and only if $I_p \subseteq I_q$.

□

The following result states that the notion introduced in Definition 3.1 presents a transitive behavior in the scope of the previous lattice.

Proposition 3.1 Let T_p, T_q and T_r be three finite state machines belonging to L such as $T_p \leq T_q \leq T_r$. If $S(T_p) = \Pi_{I_p}(S(T_q))$ and $S(T_q) = \Pi_{I_q}(S(T_r))$ then we have $S(T_p) = \Pi_{I_p}(S(T_r))$.

Proof: In $\Pi_{I_q}(S(T_r))$, every primitive belonging to $\bigcup_{i \in I_r \setminus I_q} P_{F_i}$ is, by definition of Π_{I_q}, substituted by ε in $S(T_r)$. As $T_p \leq T_q$, we have $\bigcup_{i \in I_p} P_{F_i} \subset \bigcup_{i \in I_q} P_{F_i}$. Then, each element of $\bigcup_{i \in I_p} P_{F_i}$ still remains in $\Pi_{I_q}(S(T_r))$. The other primitives have been replaced by the empty-move. Now, if we apply Π_{I_p}, the elements belonging to $\bigcup_{i \in I_q \setminus I_p} P_{F_i}$ are substituted by ε, and only the elements of $\bigcup_{i \in I_p} P_{F_i}$ are kept. Finally, the set $\Pi_{I_p}(\Pi_{I_q}(S(T_r)))$ for $I_p \subseteq I_q \subseteq I_r$, is equal to $\Pi_{I_p}(S(T_r))$ and by assumptions we have $S(T_p) = \Pi_{I_p}(\Pi_{I_q}(S(T_r)))$. Therefore, $S(T_p) = \Pi_{I_p}(S(T_r))$. ■

This proposition allows to reduce the number of necessary checked equalities. Indeed, in this lattice L we only have to detect the interactions peers to peers in order to detect them in an upper finite state machine T_p in L. Informally, it means that a valid telecommunication system T_p is interaction-free if every T_m such as $T_m \leq T_p$ is interaction-free.

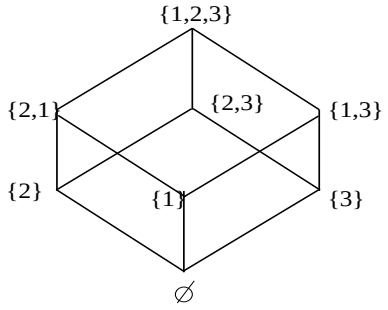

Figure 1: Lattice of the subsets of $\{1, 2, 3\}$.

Definition 3.3 A valid telecommunication system T_q is *totally interaction-free* if and only if $S(T_m) = \Pi_{I_m}(S(T_n))$ for all (T_m, T_n) in the set of the lower bounds of T_q, such as the greatest lower bound and the least upper bound of (T_m, T_n) in the lattice L are respectively T_m and T_n. □

We remove cases with the empty set $\emptyset$ and $T_m = T_n$, indeed such equalities are always verified.

Example 3.2: We take back the example above in which we have six I_p sets. According to Definition 3.1, we need to check 12 equalities. In Figure 1, we represent the lattice of all subsets of the set $\{1, 2, 3\}$ under the ordering relation of set inclusion, as related to Definition 3.2 of the lattice L. From the proposition above, in order to obtain the same result than in Example 3.1, we use the transitivity and then, only 9 equalities have to be analyzed instead of 12. This example shows how Definition 3.3 significantly decreases the number of checked equalities when we have n features.

Nevertheless, this method of detection leads us to think over obtaining the scenarios of a finite state machine. The task is not so easy and we develop this aspect in the following.

3.2 Scenarios and feature interactions

As we note in Definition 3.3, our feature interactions detection method uses sets of scenarios in finite state machines. In order to ease the FSM analysis, we can notice that it is possible to operate the projection directly on the automata to obtain the same detections. We define Ψ_{I_p}, a function that has the same effect than Π_{I_p} except that it is applied on automata, instead of sets of scenarios.

Definition 3.4 Let $I_p \subset I_q$ be two subsets of I and $T_q = (s^0, D, \Sigma, Act_{T_q}, T)$ be a FSM. We define the function Ψ_{I_p} such as
$\Psi_{I_p} : \mathcal{A} \longrightarrow \mathcal{A}$
$\Psi_{I_p}(T_q) = (s^0, D, \Sigma, \Pi_{I_p}(Act_{T_q}), T')$,
where $(s, a, s') \in T'$ if and only if $\exists (s, b, s') \in T$ such as $a = \pi_{I_p}(b)$.
□

This definition allows us to give the following proposition.

Proposition 3.2 For a finite state machine $Aut = (s^0, D, \Sigma, Act_{Aut}, T)$ and $I_p \subseteq I$, we have $\Pi_{I_p}(S(Aut)) = S(\Psi_{I_p}(Aut))$.

Proof: We prove it by double inclusion. Let $s = (y_0, p_1, y_1, \ldots, y_{m-1}, p_m, y_m)$ be a scenario belonging to $S(Aut)$. The action primitives composing s belong to $Prim(Act_{Aut})$. By definition of $\Psi_{I_p}(Aut)$, action primitives composing $\Pi_{I_p}(s)$ belong to $\Psi_{I_p}(Aut)$. Moreover, $\Pi_{I_p}(s) = (y_0, \pi_{I_p}(p_1), y_1, \ldots, y_{m-1}, \pi_{I_p}(p_m), y_m)$. Therefore, $\Pi_{I_p}(s) \in S(\Psi_{I_p}(Aut))$, that is $\Pi_{I_p}(S(Aut)) \subseteq S(\Psi_{I_p}(Aut))$.
Now, let s be an element of $S(\Psi_{I_p}(Aut))$. Let us note that the transitions of s belong to $\Sigma \times \Pi_{I_p}(Act_{Aut}) \times \Sigma$. We may obtain $s' \in S(Aut)$ such that $\Pi_{I_p}(s') = s$ because s is provided from Aut for the same reason than above. Then, $S(\Psi_{I_p}(Aut)) \subseteq \Pi_{I_p}(S(Aut))$. ■

This proposition allows us to ease the verification of Definition 3.3 for a valid telecommunication system. Indeed, instead of analyzing scenarios of finite state machines, we may use the automata structures in different cases.

3.2.1 Deterministic Finite State Machines

We need to check the equality $S(T_p) = \Pi_{I_p}(S(T_q))$ for $T_p \leq T_q$. According to Proposition 3.2, we may check $S(T_p) = S(\Psi_{I_p}(T_q))$. The set $S(\Psi_{I_p}(T_q))$ of all the scenarios describes the automaton language. In the case where after projections, the finite state machines are deterministic, the problem is solvable in polynomial time by using trace equivalence [8]. When the equality holds, we say that T_p and $\Psi_{I_p}(T_q)$ are *isomorphic* (see [15] for a good survey on the topic). We assume the deterministic structure after projecting. Indeed Ψ_{I_p} may generate a non-deterministic finite state machine (NDFSM) as shown in Figure 2.

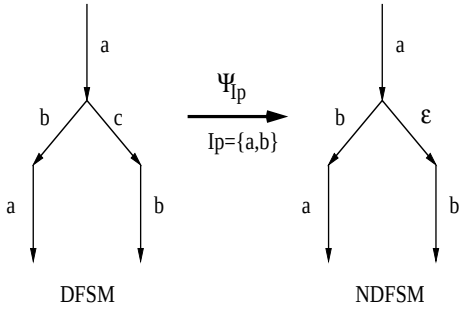

Figure 2: From deterministic FSM to NDFSM with Ψ_{I_p}.

Therefore, the feature interaction detection between two deterministic FSMs is easily computed, for example by using a depth-first search. Unfortunately, this is not the case with non-deterministic finite state machines.

3.2.2 Non-deterministic Finite State Machines

We suppose here that one of the two FSMs is non-deterministic. We cannot use isomorphism as before in this case. Then, the first reflex we may have is to transform the NDFSM into a deterministic one [19]. This reasoning is interesting and also applicable when we have small automata ($|\Sigma| \leq 10^6$). After projecting we only have to check isomorphism, as before. However, most of the time, the obtained finite state machine has a huge number of states, resulting from an exponential increase in the number of states in the worse case. That is why we need to use other steps allowing to reduce the complexity of the analysis.

Our approach here study Strongly Connected Components (SCC) of the NDFSM. First, we obtain the SCC partitions of a finite state machine *Aut* in a linear time complexity with respect to the set of states and to the set of transitions [19]. Therefore, we find every *simple* loop (that is, loops which do not contain any other loops) by the following algorithm.

Input: N a strongly connected component in *Aut*, v a state in N.
Output: A set of scenarios in N starting from v.
1- Let v_i, be a node belonging to N, $(v_i, t_j, v_j) \in T_{Aut}$, where v_j is a node in N,

2- $\gamma(v_i) = \bigcup_{(v_i,t_j,v_j)\in T_{Aut}} (v_i, t_j, v_j)\gamma(v_j)$ where $\gamma(v_j) = \{\Lambda\}$ if v_j has already been visited by $\gamma(v_i)$. $\backslash * \ \{\Lambda\}$ illustrates the empty scenario $\backslash *$.
3- return $\gamma(v)$

We apply this last algorithm to the SCC represented in Figure 3 and to the state $v = 1$. We obtain the set $\gamma(1) = \{(1, a, 2, b, 3, c, 1), (1, a, 2, b, 3, d, 4, e, 1), (1, a, 2, b, 3, d, 4, f, 5, g, 3)\}$.

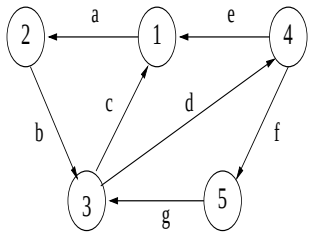

Figure 3: A strongly connected component.

Every simple loop in N is obtained from this last set by projecting the segments of scenarios. That is, in each element of $\gamma(v)$, we extract all the scenarios beginning from a state p and terminating to the same state p. In the previous case, we obtain the simple loops set $\{(1, a, 2, b, 3, c, 1), (1, a, 2, b, 3, d, 4, e, 1), (3, d, 4, f, 5, g, 3)\}$.

Proposition 3.3 This last algorithm allows to obtain all the simple loops in a strongly connected component.

Proof: By the SCC definition, all states are reachable from any states. From one state, we execute all the possible transitions. We thus may reach from this state all the other states in a recursive way and we eventually reach this same state. Therefore, all the simple loops are generated by any state.

■

The sets of simple loops corresponding to each strongly connected component are useful to reduce the analysis of NDFSM in the sense that we compare each simple loops set in the Direct Acyclic Graphs (DAG) of T_p and $\Psi_{I_p}(T_q)$. Nevertheless, we need to make two SCC match.

Definition 3.5 We say that two strongly connected components scc_1 and scc_2, which respectively belong to two finite state machines Aut_1 and Aut_2, *match* if there exists a scenario belonging to both Aut_1 and Aut_2, which reaches scc_1 and scc_2.

□

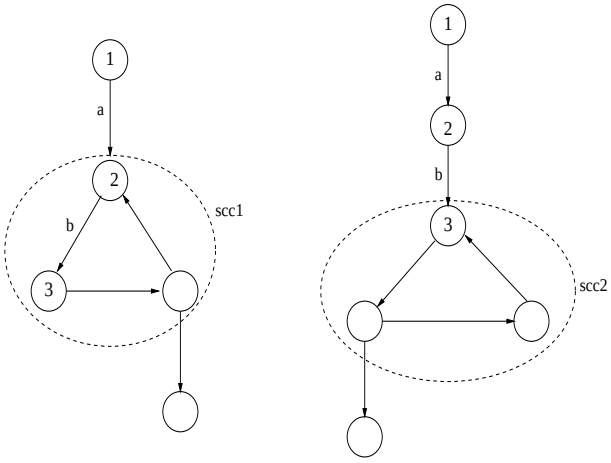

Figure 4: Two matching strongly connected components.

We may see in Figure 4 that the SCCs scc_1 and scc_2 match. Indeed, the scenario $(1, a, 2, b, 3)$ (renaming the states if necessary) allows to reach these two SCCs. The next result allows us to reduce the difficulty of the finite state machines analysis in the case where at least one of them is non-deterministic.

Proposition 3.4 Let T_p and T_q be two finite state machines such as $I_p \subset I_q$. We have $S(T_p) \neq \Pi_{I_p}(S(T_q))$ if one of the following conditions holds:
- there exists some strongly connected components having no SCC to match with,
or
- each element of the simple loops set of each SCC from T_p has to be executable in the matched SCC from T_q.

Proof:
- If it exists some loops in a FSM that can not be executed in the other FSM, it means that the behaviors of the two automata are not equal, thus the equality may not be verified.
- If this second point arises, it means that the sets of the scenarios are not the same. In this case, a scenario cannot be executed in one of the FSM.

■

Let us note that if the previous result cannot be applied, then the determinization of the automata is necessary. Actually, we can perform it step by step begining with the direct acyclic graphs, by using Proposition 3.4, and finally finish with a complete determinization of the finite state machine.

3.3 Application to a Case Study

In this section, we describe the experimental results obtained by an implementation of our algorithm. We present an application of the tool developed in our laboratory to a real

telecommunication system upon an intelligent network architecture [9]. This system is described usinf the SDL language [10] and includes several services (the Originating Call Service (OCS), Terminating Call Service (TCS), Call Forward Unconditional (CFU), Call Forward on Busyline (CFB) and the Automatic Call Back (ACB)). The model is described such as it allows the parallel execution of different calls and also calls instanciated by the network. The environment sends some messages to the users' process (Users) which is modeled by instances of SDL processes. The process Users represents the combination between the phone line, the terminal and the customer. The global system whose we give few significantly numerals in Figure 5, has been simulated in using the exhaustive simulation mode of the ObjectGEODE tool [21] and a startup file in order to obtain a complete reachability graph. The startup file allows us to restrict the simulation and thus to avoid the combinatory explosion problem.

Lines	3,098
Blocks	4
Process	9
Procedures	12
States	88
Signals	50
Macro definition	12
Timers	0

Figure 5: Metrics of the service specification.

	T_1	T_2	T
Configurations	BCS$\bigoplus$OCS	BCS$\bigoplus$CFU	The global system
Number of States	56072	60186	210453
Number of Transitions	57952	62228	242126

Figure 6: The three input automata allowing the interaction detection.

We apply our method with the goal to detect feature interactions between two services *Call Forward Unconditional* (CFU) and *Originating Call Screening* (OCS) in a telephonic system. As illustrated in Figure 6, the obtained finite state machines contain thousands of states and cannot be illustrated in this article. That is why we present here a simplified specification of the system. The global system T is represented in Figure 9 (see appendix). We may also extract two FSMs T_1 and T_2 where OCS and CFU are respectively specified (Figures 5 and 6 in appendix).

In order to verify that the behaviors of each feature are respected in T, we need to check here two equalities: $S(T_1) = S(\Pi_{\{ocs\}}(T))$ and $S(T_2) = S(\Pi_{\{cfu\}}(T))$. Our tool respectively detects 723 and 344 scenarios implying the feature interactions. Some of them may be illustrated by the present finite state machines. Indeed the scenario (*offhook, dialtone, digit, route, route, onhook*) invalidates $S(T_1) = S(\Pi_{\{ocs\}}(T))$. Therefore, we conclude that a feature interaction has been detected between OCS and CFU. We analyze deeply the result and see that one of the main cause is due to the scenario part (*route, route*) which correponds to a re-route of the call. These scenarios correspond to the same interaction instance. This means that when a call is redirected and the feature OCS is present in the system, an interaction occurs. This may be explained by the configuration in which a user A, having activated OCS with the user C in its black list, calls B who

unconditionally forwards the call to C. In this case, A is connected to C without respecting the black list requirements.

4 Conclusion

The contribution of our work is the definition of a formal method allowing the analysis of the global behavior of a system containing several services in order to detect the eventual feature interactions. We are careful in order to reduce the number of necessary checked equalities allowing the feature interaction detections. The contribution is also noticeable by the fact that we do not use constraints or pruning functions. In this paper, we also present a new approach that reduces the task of the detection in the case where one or several finite state machines are non-deterministic. An implementation of the algorithms has been made and they have been applied to a real telecommunication system upon an intelligent network architecture. We have detected feature interactions between two services and have given an explanation of the results.

References

[1] M. Amer, T. Gray, A. Karmouch, and S. Mankovskii. Feature-Interaction Resolution Using Fuzzy Policies. *Feature Interactions in Telecommunications and Software systems VI*, pages 94–112, 2000.

[2] J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[3] T.F. Bowen, F.S. Dworak, C.H. Chow, N.D. Griffeth, G.E. Herman, and Y.-J. Lin. The Feature Interaction Problem in Telecommunication Systems. *Proceedings of the Seventh IEE International Conference on Software Engineering for Telecommunications Systems*, pages 58–63, March 1989.

[4] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen. A Feature Interaction Benchmark for IN and Beyond. Technical report, Bellcore and PTT Research, 1993.

[5] R. Dssouli, S. Some, J. W. Guillery, and N.Rico. Detection of Feature Interactions with REST. In L.M.S. Logrippo P. Dini, R. Boutaba, editor, *4th Int. Workshop on Feature Interactions in Telecommunications Networks and Software Systems*, pages 271–283. IOS Press, June 1997. Netherland.

[6] J. Hassine, T. Kikuno, L. Logrippo, and M. Nakamura. Feature Interaction Filtering with Use Case Maps at Requirements Stage. *Feature Interactions in Telecommunications and Software systems VI*, pages 163–178, 2000.

[7] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International Editions, 1991.

[8] J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relations to Automata.* Addison-Wesley Publishing Company, 1969.

[9] ITU-T. General Recommendations on Telephone Switching and Signalling, Intelligent Network, Q-Series Intelligent Network Q.1200-Q.1290. Technical report, ITU-T, 1993.

[10] ITUT-T. Recommandation Z.100: CCITT Specification and Description Language (SDL). Technical report, IUT-T, 1999.

[11] B. Jonsson, T. Margaria, G. Naeser, J. Nystrom, and B. Steffen. Incremental Requirement Specification for Evolving Systems. In M. Calder and E. Magill, editors, *Feature Interactions in Telecommunications Systems VI*, pages 145–162. IOS Press, 2000.

[12] D. O. Keck and P. J. Kuehn. The Feature and Service Interaction Problem in Telecommunications Systems: A Survey. In *IEEE Transactions on Software Engineering*, volume 24, pages 779–795. TSE, October 1998.

[13] F. Khendek and A. Sefidcon. A Pragmatic Approach for Feature Interaction Detection in Intelligent Networks. In *Proceedings of the IEEE International Conference on Computer Communications and Networks*, October 1999. Boston, Massachusetts.

[14] T. F. LaPorta, D. Lee, Y.-J. Lin, and M. Yannakakis. Protocol Feature Interactions. *Proc. FORTE XI-PSTV XVIII, France*, pages 59–74, 1998.

[15] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. *The Proceedings of IEEE*, 84(8):1090–1123, August 1996.

[16] R. Milner. *Communication and Concurrency.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[17] E. Najm and O. C. Dahl. Specification and Detection of IN Service Interference Using LOTOS. In Richard L. Tenney, Paul D. Amer, and M. Ümit Uyar, editors, *FORTE*, IFIP Transactions, pages 53–69, 1993.

[18] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice Hall, First edition, 1981.

[19] R. Tarjan. *Depth-First Search And Linear Graph Algorithms*, volume 1. SIAM Journal Computer, June 1972.

[20] K. J. Turner. An Architectural Foundation For Relating Features. In L.M.S. Logrippo P. Dini, R. Boutaba, editor, *Proc. 4th.International Workshop on Feature Interactions in Telecommunication Networks and Software Systems*, pages 226–241. IOS Press, 1997. Netherlands.

[21] Verilog. *ObjectGEODE Simulator, Reference Manual*, 1997.

A Appendix

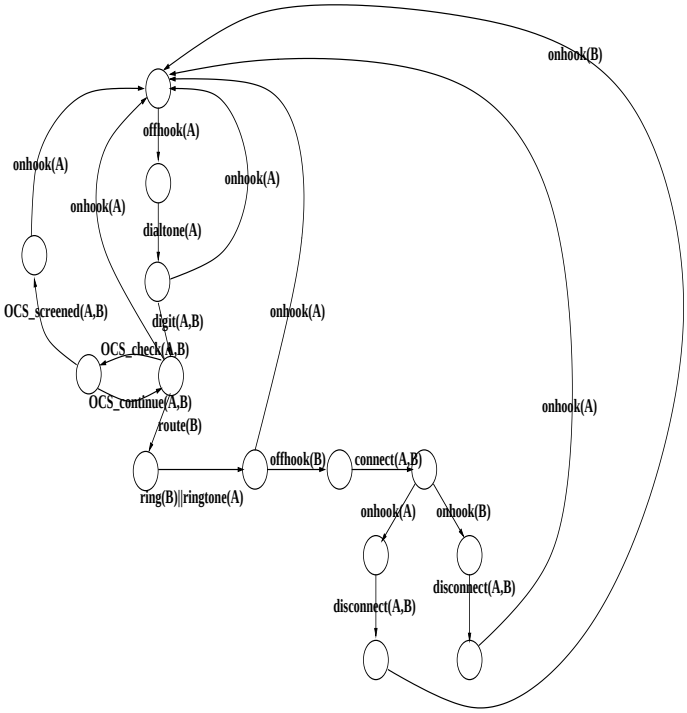

Figure 7: The finite state machine T_1: the feature OCS on the BCS.

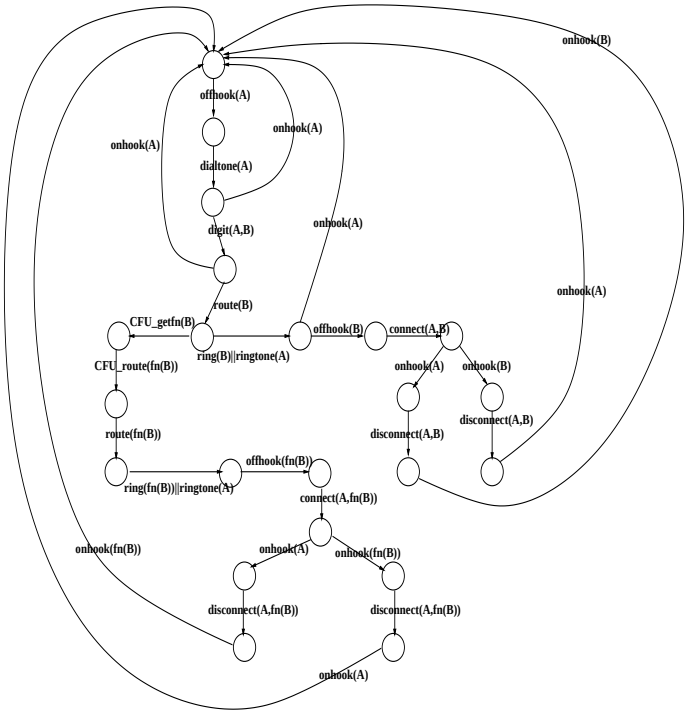

Figure 8: The finite state machine T_2: the feature CFU on the BCS.

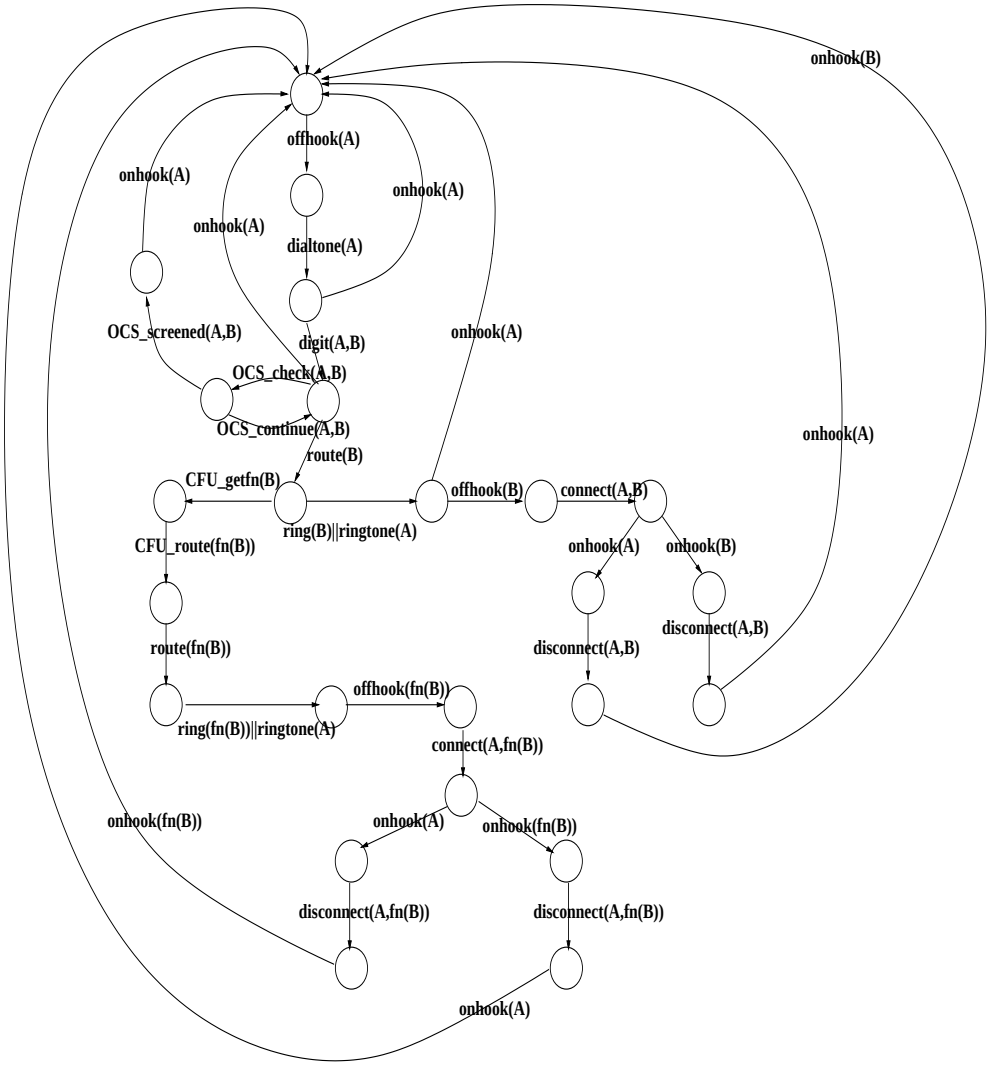

Figure 9: Global finite state machine T.