A New Algorithm for Service Interaction Detection

Ana Cavalli and Stéphane Maag

Institut National des Télécommunications
9 rue Charles Fourier
F-91011 Evry Cedex, France
phone : +33 1 60764474, fax : +33 1 60764711
e-mails : {Ana.Cavalli, Stephane.Maag}@int-evry.fr

Abstract This paper presents a new formal method for telephony services engineering allowing feature interactions detection. System and service specifications are provided using the specification language SDL. These specifications are simulated in order to obtain finite state machines. An algorithm allowing to compare service scenarios according to marked transitions is proposed. The algorithm has been implemented and applied to a case study, a telecommunication system on Intelligent Network architecture including the Basic Call Service (BCS) and two supplementary: Originating Call Service (OCS) and Call Forward Unconditional (CFU). The results of this application are presented.

Keywords: *Telecommunication service, Feature interaction, Finite State Machine, Intelligent Network.*

1 Introduction

The service interaction is one of the most critical problems in the telecommunications domain. In order to give an example of such problems, let two users A and B be subscribers of the service *Call Forward Unconditional*. A forwards his calls to B and the latter B forwards to A. Let us imagine a third user C who calls A or B. What happens to user C ? What happens for the telecommunication network ? The detection and resolution of these interactions become more and more complex and crucial. Indeed, the number of services (or features, we do not differentiate the two terms here) may be witnessed. In order 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].
Many types of feature interactions may be observed [5], this is the reason why it is very difficult to find 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 [9], as well as, during later stages, when implementing specific environments [1].

We may notice that the number of service interactions is growing up through all these stages. Therefore, in order to be efficient, we present in this paper an analysis and a formal method for feature interactions detection based at the upper level of the life cycle, that is the specification phase. There exists several suitable formalisms to model protocols. For example, extended finite state machines [15], transition systems [16], labelled transition systems [4], process algebras [2] and Petri nets [17]. 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 [6] leads to the use of Message Sequence Charts scenarios. However, a partial behavior of the system is described while we would like to perform a global behavior analysis. There are others methods allowing to detect some interactions (e.g. [7,8,13]) such as the ones using the feature requirements. In turn, requirements are expressed by properties. Nevertheless, 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 sometimes necessary to rewrite some properties. That is why the choice of our study turns on a formal approach based on finite state machines.

We use a specification of services based on Extended Finite State Machines (EFSMs) to analyze and detect feature interactions. Work involving the same formal concepts is presented in [14]. The authors use a restriction function on automata that deprives the analysis of a specification part. However, no satisfying method is given allowing to show that the pruned part is negligible to the feature interaction detection. Our work consist in the detection of interactions based on comparison of scenarios with marked transitions. The main contribution of our work consists in the design of a new algorithm for the detection of interactions from the behavior of a telecommunication system. 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 [11], however we present here only the results dealing with the Originating Call Service (OCS) and the Call Forward Unconditional (CFU).

The article is organized as the following. Section 2 introduces the basic notions and the definitions illustrating the problematic. Section 3 describes the feature interaction detection algorithm. Section 4 presents the application of the case studied on a real telecommunication system. And finally, Section 5 concludes the article.

2 Basics

The Extended Finite State Machines (abbreviated by 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 particularly suited to study systems that can be described by using the extended finite state machines. 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 1: A *Finite State Machine* (in short FSM) or *automaton*, is a quintuple $Aut = (s^0, D, \Sigma, Act_{Aut}, T)$ with:

- $s^0 \in \Sigma$ is the initial state of Aut,
- D is the set of 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 set of transitions. □

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 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 that $\mathcal{F} \in \mathcal{P}(\{F_0, F_1, \ldots, F_n\})$. F_0 is the BCS on which the other value-added features F_i $(i \in I = \{1, \ldots, n\})$ are connected. A telecommunication system T_k is a finite state machine such that $T_k = BCS \bigoplus_{i \in I_k \subset I} F_i$. In our work, we assume that the connections are specified and that the finite state machines are given.
When we analyze the 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 the functional aspects. *Execution sequences* (or *scenarios*) are a good and formal way in order to capture this kind of behavior. Next, we introduce this concept in our framework.

Definition 2: Let Aut be a finite state machine. A *scenario* s in Aut is a concatenation of elements in T, where data have been removed. Only primitives named *feature primitives* are kept. In our paper, we study feature interactions from the control part of the system behaviors. Therefore, we avoid the generation of multiple behaviors that are the same (e.g. user A or B offhooks and onhooks). We represent every Aut's scenario by $S(Aut)$. □

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 Algorithm

In order to analyze feature interactions, we need to obtain the whole studied system behavior. We generate all the behaviors in deploying and executing every cycle in the finite state machines.

3.1 Acyclic paths and simple loops generation in a Finite State Machine

The behaviors are usually cyclic in telecommunication systems, that is, when all the users onhook, the current state becomes the initial state, and we obtain the system like a huge strongly connected component. Therefore, for the automata analysis, we stop the scenarios execution when we reach the initial state back, defining then final states. In order to obtain every scenario of a FSM Aut, we first transform it in a Direct Acyclic Graph (DAG) Aut' in which the nodes are either Aut nodes or Strongly Connected Components (SCC) in Aut. To compute the partitions of finite state machines into SCCs, we apply the method developed by Tarjan [18] implemented in our laboratory. By a depth-first search, we obtain the acyclic paths in the finite state machine Aut'. Into the scenarios, some nodes represent some SCCs and then we need to deploy them in order to analyze the whole behavior. We generate for this reason every simple loop. We call simple loops the cycles which do not contain other cycles. We apply the following algorithm to the strongly connected components.

Scenario-Generation 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 1 and to the state $v = 1$. We obtain the set $\gamma(1) = \{(1, a, 2, b, 3, c, 1), (1, a, 2, b, 3, d, 4, e, 1), (1, a, 2, b, 3, d, 4, f, 5, g, 3)\}$. 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 1 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

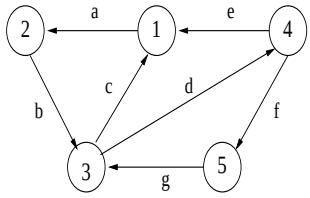


Figure1. A strongly connected component.

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. ■

We obtain simple loops that allow to analyze interactions from the FSM scenarios. Nevertheless, in the general case, the set of all these scenarios has a infinite size, indeed, the SCC cycles may be executed infinitely. The method used here consists in, first of all, obtaining the acyclic paths in a DAG and in the second hand, analyzing the automaton SCC. This is detailed in the following.

3.2 Scenarios Analysis

Let our system be composed by two services F_1 and F_2 (different from the BCS) with some different primitives. The specification of this system is written in SDL and with ObjectGEODE [19] we generate three finite state machines A_1, A_2 and G. They respectively represent the feature F_1 and the BCS, the feature F_2 and the BCS, and the global system containing F_1, F_2 and the BCS.
Our work follows the idea that there exists a feature interaction between F_1 and F_2 if and only if the behavior of G does not respect the expected and defined behaviors of A_1 and A_2, according to Section 2.
In an informal manner, here is the procedure allowing the feature interaction detection we propose:

- First, we want to verify that the scenarios included in A_1 and A_2 are present in the scenarios set of G (having renamed the states if necessary). If this is not the case, it means that the service behavior, whose the scenario is the cause, has been modified. A feature interaction occurs.
- In the second hand, if the inclusion in the last step is verified, we also have to check that the scenarios of G non-executed previously can be produced by A_1 and A_2. The mixed execution of actions of A_1 and A_2 have to generate them. These scenarios correspond to the configurations where both services are active. Indeed, the other scenarios are checked by the previous step. The mixed execution of A_1 and A_2 may generate scenarios where only one service is activated, which are undesired. Therefore, we remove transitions that prevent to services from beeing triggered or do not cause the services to be triggered. In A_1 and A_2, we mark such transitions.

The first step of the procedure corresponds of preserving the A_1 and A_2 behaviors in the global system. The second step allows to guarantee that both features are activated in the scenarios of G not taken into account previously, and that these latters may be generated by the mixed execution of A_1 and A_2 actions. This combination is obtained by the *independent product* $A_1 * A_2$ of A_1 and A_2. The independent product allows to obtain all the interleaved behaviors of the automata A_1 and A_2.

Definition 3: Let $Aut_i = (s_i^0, D_i, \Sigma_i, Act_i, T_i), \quad i = 1, 2$, be two finite state machines, we note $*$ the *independent product* between the two Aut_i, defined by:
$A_1 * A_2 = Aut = ((s_1^0, s_2^0), D_1 \times D_2, \Sigma_1 \times \Sigma_2, Act_1 \cup Act_2, T)$
with T obtained such that:
$(s_1, x_1, s'_1) \in T_1 \Longrightarrow \forall s_2 \in \Sigma_2, \quad ((s_1, s_2), x_1, (s'_1, s_2)) \in T$
$(s_2, x_2, s'_2) \in T_2 \Longrightarrow \forall s_1 \in \Sigma_1, \quad ((s_1, s_2), x_2, (s_1, s'_2)) \in T$

□

We illustrate this product in Figure 2.

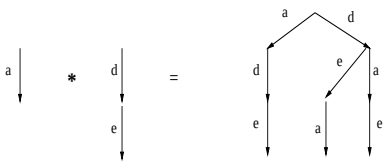


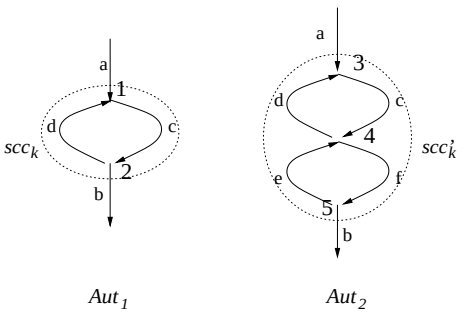
Figure2. Example of independent product $*$.

We need to check the inclusion of one set of scenarios in another set of scenarios. However, strongly connected components may exist. As follows, we give a method based on two SCCs obtained with scenario-generation algorithm in order to improve this analysis.
For every strongly connected component scc_i in Aut_1, let P_i^j be the j execution sequences from the initial state to scc_i. These P_i^j have to reach a $scc_i^{'}$ in Aut_2 such that the set of simple loops of scc_i is included in $scc_i^{'}$. Furthermore, each state of the simple loops of these both strongly connected components must have the same outcoming and incoming transitions. Otherwise, $S(Aut_1) \not\subseteq S(Aut_2)$. Figure 3 shows an example where $S(Aut_1) \not\subseteq S(Aut_2)$.

By this technique, we check whether a set of scenarios is included in another one by analyzing the strongly connected components.

3.3 Feature Interactions Detection Strategy

We expound here our feature interaction detection method. First, we check whether the behaviors defined by each isolated automaton A_1 and A_2 are present in the finite state machine G. After that, we verify that the other scenarios in



The simple loops of scc_k in Aut_1 are included in the one of scc'_k. However, in the state 2, the transition c is incoming, and b outcoming which is not the case for state 4. We infer $S(Aut_1) \nsubseteq S(Aut_2)$.

Figure3. Automata with different sets of scenarios.

G, that is $S(G) \setminus \{S(A_1) \cup S(A_2)\}$ are included in $S(A_1 * A_2)$. As mentioned before, marked transitions in A_1 and A_2 do not have to be executed. The marked automaton A_i is noted $\overline{A_i}$. Formally, we can describe our feature interaction algorithm as follows.

Interactions-Detection Algorithm
Input: A_1, A_2 and G, three FSMs respectively representing F_1 and the BCS; F_2 and the BCS, F_1, F_2 and the BCS.
Output: "Interaction detected" or "No interaction detected".
1- **if** $S(A_1) \subset S(G)$ **and** $S(A_2) \subset S(G)$ **and** $S(G) \setminus \{S(A_1) \cup S(A_2)\} \bar{\subset} S(\overline{A_1} * \overline{A_2})$
2- **then** "no interaction detected"
3- **else** "interaction detected".

We note here $S(G) \setminus \{S(A_1) \cup S(A_2)\} \bar{\subset} S(\overline{A_1} * \overline{A_2})$ the inclusion which is true if and only if $\forall s \in S(G) \setminus \{S(A_1) \cup S(A_2)\}$ such that s does not execute marked transitions and reaches a final state of A_1 or A_2, we have $s \in S(A_1 * A_2)$. Indeed, if s does not reach neither a final state in A_1 nor in A_2, it means that s in G have not executed a excepted behavior in A_1 or A_2. That is a feature interaction occurs. In order to avoid the combinatory explosion problem when $\overline{A_1} * \overline{A_2}$ is generated, the inclusion $\bar{\subset}$ will be verified on-the-fly in $\overline{A_1}$ and $\overline{A_2}$.
This feature interaction detection algorithm is applied to a pair of services. It means that we analyze two-way interactions. However we also may be confronted with three-way (and even more) interactions although pairs of these features do not show any interaction. Let us imagine a party A, subscriber of the *Unlisted Number* service (the subscriber may hide his call number to a called party), that calls a party B, subscriber of an *Automatic Recall* of the last caller (the subscriber may automatically recall the last caller). User B is also a subscriber of *Itemized Billing* service. If every feature is activated and B uses the *Automatic Recall*, A's number appears on B's phone bill. This is clearly a violation of the intention of A's *Unlisted Number*. These cases are less common than two-way interactions but we may extend our algorithm in order to tackle this kind of interactions. Let A_1, A_2, A_3 and G be four finite state machines respectively containing the ser-

vices F_1, F_2, F_3 and $\{F_1, F_2, F_3\}$. The last automaton G represents the global system. The reasoning is then the same than for the interactions-detection algorithm except that we focus on the G scenarios composed by behaviors belonging to A_1, A_2 and A_3 interleaved together. For this reason, we use the independent product between A_1, A_2 and A_3.

Property 2: Let A_1, A_2 and A_3 be three finite state machines, we have:
commutativity: $A_1 * A_2 = A_2 * A_1$,
associativity: $(A_1 * A_2) * A_3 = A_1 * (A_2 * A_3)$.

In order to extend our algorithm to three-way interactions detection we then use the independent product $A_1 * A_2 * A_3$ (also written $\bigotimes_{i\in\{1,2,3\}} A_i$) represented by $(A_1 * A_2) * A_3$. As before, we mark the transitions allowing the deactivation or non-activation of a service. The following algorithm allows to detect such interactions.

Three-Way-Interactions-Detection Algorithm
Input: A_1, A_2, A_3 and G, four FSMs that respectively represent F_1 and the BCS; F_2 and the BCS, F_3 and the BCS, F_1, F_2, F_3 and the BCS.
Output: "Interaction detected" or "No interaction detected".
1- **if** $S(A_1) \subset S(G)$ **and** $S(A_2) \subset S(G)$ **and** $S(A_3) \subset S(G)$ **and** $S(G) \setminus \{S(A_1 * A_2) \cup S(A_1 * A_3) \cup S(A_2 * A_3)\} \bar{\subset} S(\overline{A_1} * \overline{A_2} * \overline{A_3})$
2- **then** "no interaction detected"
3- **else** "interaction detected".

This algorithm allows to check whether the isolated behaviors of F_1, F_2 and F_3 are present in the global system G. Furthermore, the fourth inclusion verifies if the execution of F_1, F_2 and F_3 in a same G scenario has a sense with regard to the A_1, A_2 and A_3 specifications. Let us remark that Three-Way-Interactions-Detection Algorithm does not guarantee the Two-Way interactions detection covered with the Interactions-Detection Algorithm. Indeed, the scenarios only containing pairs of services behaviors are not covered there. By this way, we also may study four-way (and more) interactions in extending our method. We give in the following a method allowing to detect interactions between n features. This is an extension of Three-Way-Interactions-Detection Algorithm in which we detect interactions when n services are active.

N-Way-Interactions-Detection Algorithm
Input: $A_1, A_2, \ldots, A_n$ and G, $n+1$ FSMs that respectively represent F_1 and the BCS; F_2 and the BCS,... F_3 and the BCS, and the BCS with $F_1, \ldots, F_n$.
Output: "Interaction detected" or "No interaction detected".
1- **if** $\bigcup_{i\in\{1,\ldots,n\}} S(A_i) \subset S(G)$
and $S(G) \setminus \{ \bigcup_{i\in\{1,\ldots,n\}} S(\bigotimes_{j\in\{1,\ldots,n\}, j\neq i} A_j)\} \bar{\subset} \mathrm{S}(\bigotimes_{i\in\{1,\ldots,n\}} \overline{A_i})$

2- **then** "no interaction detected"
3- **else** "interaction detected".

As before, this last algorithm does not guarantee k-Way-Interactions-Detection for $k \in \{2, \ldots, n-1\}$. However, when n increases, the number of N-Way interactions decreases. This method is used with a number n of features that allows to limit the number of k-Way interactions detection and thus improving the computation. Indeed, in order to detect every feature interaction between each service, it is necessary to tackle each k-Way interaction. Our approach enables to resolve this last point.
In order to apply our Interactions-Detection Algorithm, we illustrate a case of study in the following.

4 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 networks architecture [10]. The services integrated in this system are 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 system is described using the SDL language [11] for the call treatment, the services invocation and for the customers management. The framework of this specification is described in Figure 4.

The global system whose we give few significantly numerals in Figure 5, has been simulated in using the exhaustive simulation mode and a startup file in order to obtain a complete reachability graph. The startup file allows to restrict the simulation and thus to avoid the combinatory explosion problem. Indeed, it is impossible to obtain the complete reachability graph, huge memory size is necessary for such a graph. Therefore, the startup file of the specification allows to activate only the Originating Call Service (OCS) and the Call Forward Unconditional (CFU), and to put the three others inactive. We also may reduce the number of the customers as three, U_1, U_2 and U_3.

Our tool needs three input files. The first one represents an automaton where OCS is lonely active, the second one illustrates the system with only CFU active, and the last one represents the global system containing the two services above. In order to mark the transitions of the two first finite state machines, a static insertion of a signal *fake* is provided in the SDL specification. This signal is sent when the service is unused as it is shown on Figure 6.

With the ObjectGEODE tool, we have simulated the three configurations and have obtained the three finite state machines A_1, A_2 and G whose the details are given in Figure 7.

We may underline that these three finite state machines A_1, A_2 and G do not contain neither deadlocks, nor livelocks. If this is the case, it would meant that the behavior is not correct, and then, an analysis of these scenarios reaching these

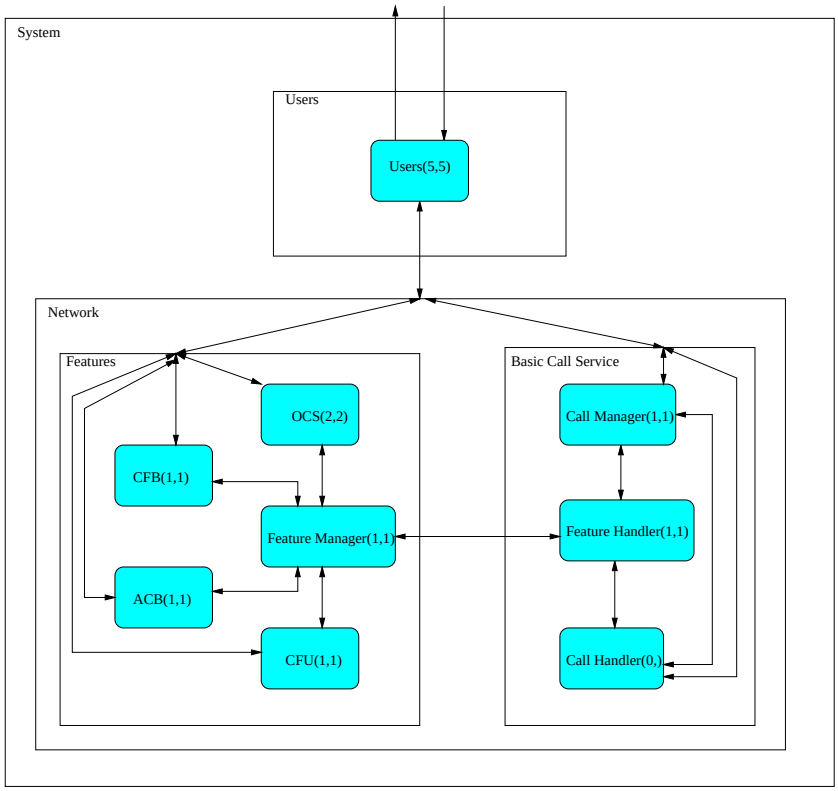


Figure4. Global Framework.

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

Figure5. Metrics of the service specification.

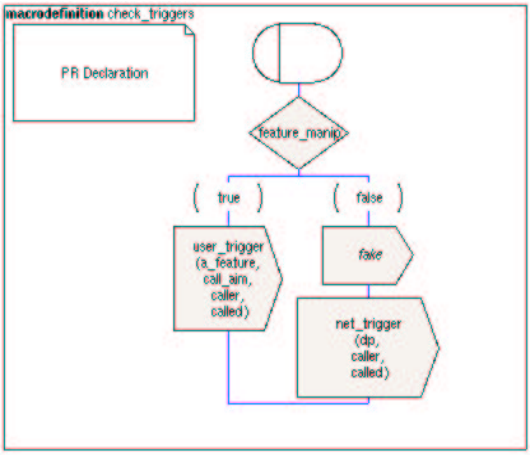


Figure6. Marks of the transitions in the SDL specification.

	A_1	A_2	G
Configurations	*U_2 can activate and deactivate CFU to U_3.*	*U_1 can activate and deactivate OCS with U_3 in his blacklist.*	*U_1 and U_2 may respectively activate and deactivate OCS and CFU.*
Number of States	17081	17629	18379
Number of Transitions	17571	18123	18907

Figure7. The three input automata allowing the interaction detection.

deadlocks or livelocks would be necessary. From the formal method described in Section 3.2, we check $S(A_1) \subset S(G)$ and $S(A_2) \subset S(G)$. Our tool does not detect any anomalies when we verify both inclusions, it means that the F_1 and F_2 isolated behaviors are correctly respected in the global system. The A_1 and A_2 transitions have previously been marked and the implementation of our algorithm respectively provides us 228,157 and 507 scenarios in A_1, A_2 and G. The third inclusion takes into account 285 scenarios which we need to reproduce from the A_1 and A_2 behaviors in $A_1 * A_2$. We note that 124 errors invalidate the inclusion. Therefore, we have analyzed the obtained results in order to observe the interactions we detected. Every scenario implied in the detection represents the same interaction, indeed, we reach the wished final states, but the *fake* transition is executed. We have studied the schema of the detected communication and we observe that the interaction is produced when the user U_1 is in line with U_3 whereas this latter is in the U_1's blacklist.

5 Conclusion

The contribution of our work is the definition of a formal method allowing to analyze the global behavior of a system containing several services in order to detect the feature interactions. We extend our algorithm in order to detect three-way interactions. This work has led to the implementation of a tool taking three finite state machines in input and allowing to detect the interactions. We applied our algorithm to a real case study in a system with an Intelligent Network architecture. The experimentation has been realized on the system containing the Call Forward Unconditional service and the Originating Call Service. We detect 124 scenarios that lead to the same interactions. One of the perspectives is to restrict the scenarios analysis in order to avoid the execution of all the scenarios which create a redundant interaction.
Furthermore, it is possible to study the desirable interactions with our method. Indeed, this could be obtained as a generalization of our algorithm by marking the transitions with different criteria. This would allow to reach objectives such as desirable interactions.

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*, 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*, March 1989.
4. E. Brinksma. A Theory for Derivation of Tests. In *IFIP WG6.1 8th Int. Symp. On Protocol Specification, Testing and Verification*, 1998. North Holland.
5. 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.
6. 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.
7. M. Frappier, A. Mili, and J. Desharnais. Defining and Detecting Feature Interactions. In R.S. Bird and L.G.L.T. Meertens, editors, *IFIP TC 2 Working Conference on Algorithmic Languages and Calculi.* Chapman and Hall, 1997.
8. P. Gibson, G. Hamilton, and D. Méry. A taxonomy for Triggered Interactions using Fair Object Semantics. In M.Calder and E.Magill, editors, *Feature Interaction in Telecommunications and Software Systems VI.* IOS Press, 2000.
9. 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*, 2000.
10. ITU-T. General Recommendations on Telephone Switching and Signalling, Intelligent Network, Q-Series Intelligent Network Q.1200-Q.1290. Technical report, ITU-T, 1993.
11. ITUT-T. Recommandation Z.100: CCITT Specification and Description Language (SDL). Technical report, IUT-T, 1999.
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-PSTV, France*, 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. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Universitat Bonn, 1962.
18. R. Tarjan. *Depth-First Search And Linear Graph Algorithms*, volume 1. SIAM J. Computer, June 1972.
19. Verilog. *ObjectGEODE Simulator, Reference Manual*, 1997.