An automatic off-line feature interaction detection method by static analysis of specifications

Hélène Jouve
France Telecom R&D
2 av. Pierre Marzin
F- 22307 Lannion
Tel: (+33) 29 60 53 954
helene.jouve@francetelecom.com

Pascale Le Gall
LaMI, Univ. d’Évry
523 pl. des Terrasses
F- 91000 Évry
Tel: (+33) 1 60 87 39 14
legall@lami.univ-evry.fr

Sophie Coudert
Institut Eurecom
BP 193
2229 route des crêtes
F- 06904
Sophia-Antipolis
sophie.coudert@enst.fr

Abstract
This paper presents an automatic off-line method for detecting interactions from service specifications. Services are specified with diagrams. The detection is based on a static analysis and reveals two main kinds of interactions: when two services are triggered at the same time or when the trigger of a service leads to the trigger of another one. It is implemented in Prolog and we present a conclusive case study. The interactions are computed in terms of subscriptions, state, triggering event and conflicting results. Our algorithms can be modified to take into account new concepts introduced by VoIP and new IP services.

Keywords: feature interaction, off-line detection, static analysis, Internet telephony.

1 Introduction

The feature interaction problem
Telecommunication systems are constantly changing due to the introduction of new services (features) and to the evolution of underlying technologies. [6], [8], [11], [4] give a general presentation of the use of service-oriented development to help the design of telecommunication systems by modularizing their functionalities. A service may be viewed as an optional unit modifying an underlying service\(^1\). The

\(^1\) As in many articles dealing with phone services, this pre-existent basic call service is called POTS for Plain Old Telephone Service. It ensures calls between two users.
particularity of such service-oriented systems is the emergence of interactions among services during their integration in the system because they are often designed separately. The mutual influence between two or more services makes the system behave differently from what has been specified for each service. Moreover, these interactions may be considered as cooperation or conflict, in other words, as good interactions or as bad ones. In order to propose care-free sets of services, network operators are interested in shortening the design cycle of services by developing ad hoc methodological approaches. This cycle is generally different from the one of classical software: a detection phase is often coupled with the first design steps. [5] gives a pragmatic example of such a service-oriented design method.

Formal methods are promising in detecting interactions. Verification techniques such as theorem-proving, model-checking or formal testing have been proposed for that purpose ([11], [18], [1], [10], [3], [21]). The main common drawback of these approaches is that they need costly efforts such as modeling telecommunication systems as automata and systematic exploration. However some works ([22], [5], [9], [15] ...) are based on static analysis methods. They consist of analyzing service specifications in order to reveal interactions. Most of them deal with pre-post formulas expressing properties by means of event-based transition rules.

Our approach The work we present here concerns a static method for detecting interactions. Our paper is an extended version of [15]: we add a significant case study and sketch out how we are extending our method to deal with VoIP services. [14] provides a more complete case study based on the benchmarks given by the two interaction detection contests ([12], [17]). The current method, working for phone services, is fully implemented and allows the fully automatic detection of interactions from diagram specifications2. Indeed, a classical way to specify feature is to synthesize behaviors under the form of representative message sequences ([19]). In our approach, we consider sequences of messages sent between the network and the user devices. Such a specification style based on diagrams is widely used (see for example [12], [17] with respectively Chisel sequence diagrams and state transition diagrams).

To simplify algorithms, we have defined our own framework of diagram-based specifications: we made variable scope, subscriptions to features and mechanisms of aliases clear. Finally, we model diagrams of services as simple trees (translated into terms to be analyzed). Indeed, we chose Prolog to take advantage of the symbolic facilities of logic programming. With re-

---

2In fact, this work is a significant part of two industrial and academic French projects MOECIF and ValiServ.
spect to the approaches using pre-post rules, dealing with diagrams offers a better consideration of state successions and a finer characterization of state equivalence. Interactions are basically made up of a configuration of service subscriptions, a triggering message and a state description. We restrict ourselves to the problem of interaction detection between two services.

Our call configurations are similar to possibly interacting configurations (PIC) as defined in [16] and to the notion of terminal assignments in [21]. PICs are hand built by an enumeration of all call configurations likely to bring about some interactions between two services. Terminal assignments associate real terminal to terminal variables involved in the pre-post rules specifying features. All pertinent terminal assignments are computed by enumerating all interesting combinations of terminal variables between two pre-post rules. Unlike to [16] and [21], we do not a priori build a set of potentially interesting configurations. On the contrary, call configurations are computed on-the-fly during the interaction detection process. Thus, we are able to only consider call configurations leading to real interactions.

The paper is structured as follows: Section 2 introduces the basic elements of our specification framework. Section 3 presents our method for interaction detection based on two implemented algorithms. Section 4 illustrates our approach on a representative case study. Section 5 sketches out to adapt our work to VoIP and especially to the SIP protocol.

2 Feature specifications

2.1 The example of the basic system

A service specification comprises a diagram expressing exchanges between the network and the connected phones, and some constraints expressing physical properties of the phones. Since we adopt a user point of view, we abstract the whole physical network and its components by a unique object, the network.

We first illustrate our formalism on the example of the POTS diagram (Figure 1). The beginning of a communication involves a single phone, A. At the root node, the annotation “A : POTS” indicates that from this point, the variable A will represent a phone subscribing to POTS, and the label idle(A) specifies that A is in the idle state. Node properties are both required before messages labelling outgoing edges are sent and after messages labelling entering ones are sent. Most of the edges are labelled by a phone message and a network answer. Conventionally, a phone message such as A.call(B), begins with the phone name while a network message like start(disctone).A, ends with it. At each node, each potential future event is specified so that each possible transition is on the diagram. Whether B is
idle or not, there are two possible network answers to the message $A.call(B)$. Thus there are two edges with a shared part labelled $A.call(B)$. $B$ is introduced at this level, as a subscriber to POTS. If $B$ corresponds to an engaged equipment, then it meets the condition $\sim idle(B)$ and $A$ obtains the line busy tone at the next state (emitting($A$,lbtone)). For simplicity, we consider that dialing an incorrect number also leads to the line busy tone. Notice that $idle(B)$ and $\sim idle(B)$, its negative, are incompatible conditions and the transitions are deterministic. Each node is annotated by a number $n$ and an alphabetically ordered list of variables, the ones which are in its scope. Moreover, each node is implicitly prefixed by the service name POTS. On Figure 1, numbers from 1 (the root node) to 14 (the idle($B$) one) and lists of variables are given near each node. This numeration allows us to model intrinsic loop phenomena and to avoid specification redundancies. Thus, we could have replaced the node “12, ($A$, $B$)” by the alias “POTS, 11, ($B$, $A$)”. This alias with the list ($B$, $A$) would indicate that the rôles of $A$ and $B$ have to be exchanged. Indeed, when $B$ goes on hook, everything happens as if $A$ goes on hook, except that their rôles are exchanged. All other symbols involved in the POTS diagram naturally correspond to their usual meaning. For example, the network message $start$($disctone$).$A$ leads to a specific tone indicating to $A$ that there is no longer a user talking to it. By going on hook after the node 5, $A$ enters in an idle state and do not listen to any tone.

To specify services, some pieces of information are missing about state literals. A phone $A$ cannot satisfy, in the same state, the literals $talking(A,B)$ and $idle(A)$. This is specified by means of axioms, called constraints, of the form $talking(A,B) \Rightarrow \sim idle(A)$. They allow us to restrict the set of admissible reachable states. Clearly, diagrams lack modularity: they are pertinent to describe a feature, but do not allow to specify feature integration.
dealing with interaction detection, diagrams are convenient since a static analysis-based method only requires to compare two feature specifications.

2.2 Syntax

Among the basic data, a predefined symbol phone denotes the generic connected equipment. A data signature \( \Sigma = (S, \Omega) \) consists of a set \( S \) of sorts such that (phone \( \in S \)) and a set \( \Omega \) of operation definitions \( f : s_1 \times \ldots \times s_n \rightarrow s \) where \( f \) is an operation symbol with \( s_1 \times \ldots \times s_n \) an element of \( S^* \) and \( s \) an element of \( S \setminus \{ \text{phone} \} \). Examples of sort and operation definition sets are \{phone, tone\} and \{dialtone \( \rightarrow \) tone, disctone \( \rightarrow \) tone, \ldots\}. Connected equipments are only represented by variables in order to be anonymous and generic. We denote by \( V \) a set of variables of sort phone, provided with a total order (in fact, alphabetical). Terms with variables are built on \( \Sigma \) in the usual way. Specifications include some basic information:

**Definition 1** A feature signature \( \Theta = (\Sigma, F, M, E, P) \) consists of a data signature \( \Sigma = (S, \Omega) \), a set \( F \) of feature names, a set \( E \) of phone message names (sent to the network), a set \( M \) of network message names (sent to phones), and a set \( P \) of predicate names. Each element \( x \) in \( F, E, M \) or \( P \) is provided with an arity \( s_1 \ldots s_n \) over \( S^* \), denoted by \( x : s_1 \times \ldots \times s_n \) or \( x : \varepsilon \) if \( n = 0 \).

For \( t_1, \ldots, t_n \) terms of sort resp \( s_1, \ldots, s_n \) and for \( A \) a phone variable, let us denote by \( A : f(t_1, \ldots, t_n) \) a subscription meaning that the subscriber phone \( A \) has subscribed to \( f : s_1 \times \ldots \times s_n \) in \( F \), \( A.m(t_1, \ldots, t_n).A \) a phone message (resp. a network message) for \( m : s_1 \times \ldots \times s_n \in E \) (resp. \( \in M \)) meaning that the phone \( A \) sends (resp. receives) the message \( m(t_1, \ldots, t_n) \) to (resp. from) the network, \( p(t_1, \ldots, t_n) \) for \( p : s_1 \times \ldots \times s_n \in P \) an atomic property meaning that \( p(t_1, \ldots, t_n) \) holds.

A literal is either an atomic property \( l \), or its negative denoted by \( \sim l \). According to the context, we call any set of literals a state condition. \( \text{State}_\emptyset(V) \) denotes the set of all state conditions. A system state is a state condition such that for each atomic property \( l \), it contains either \( l \) or \( \sim l \).

**Definition 2** A call configuration is a set \( C = \{ c_1, \ldots, c_n \} \) of subscriptions such that for each \( t \) of sort phone, for each subterm of any \( c_i \) in \( C \), there exists in \( C \) a subscription of the form \( t : f \).

Suppose that the Terminating Call Screening (TCS) service is introduced as \( TCS : \text{phone} \), the subscription \( A : TCS(B) \) indicates that \( A \) subscribes

---

\(^3\) \( S^* \) denotes all the words on \( S \) (including the empty word).
to TCS with B as forbidden origin of incoming calls\textsuperscript{4}. \{A : TCS(B), B : POTS\} is a call configuration while \{A : TCS(B)\} is not an acceptable one since nothing is told about the subscriptions of B. The subscription to POTS is the minimal one. Most of services are defined from this knowledge. This can be generalized by introducing a partial order relation over features. \(F_1 \leq F_2\) means\textsuperscript{5} that the specification of \(F_2\) is based on the knowledge of the one of \(F_1\). By assumption, \(\text{POTS} \leq F\) for all services \(F\).

To ease static analysis, diagrams are labelled rooted trees represented as finite terms. To express loop schemas or reduce diagram size, nodes are numbered, allowing alias nodes pointing at an existing numbered node. Diagrams describe system behavior by representing typical exchanges between the network and phones. They are close to Chisel diagrams ([2], [19]), specifying an alternation of messages from phones to the network and from the network to phones. Different types of label are defined for nodes and edges using an arbitrary set \(N\) (in practice \(\mathbb{N}\) the set of natural numbers) to systematically number nodes.

**Definition 3** Let us consider a feature signature \(\Theta\) for a service \(F\).

- An alias for \(F\) is a pair \(((G,n), (v_1, \ldots, v_k))\) consisting of a node identifier \((G,n)\) with \(n \in N\) and \(G \leq F\) a feature, and a list \((v_1, \ldots, v_k)\) of variables in \(V\). The set of aliases for \(F\) is denoted by \(\text{Alias}_{\Theta}(V)\).
- A state label is either a state condition or an alias for \(F\).
- An edge label is a tuple made up of a set of subscriptions (possibly empty and called declaration of phones), a phone message (the only optional element, called an input), a state condition and a set of network messages \(S\) (called an output). The set of all edge labels is denoted by \(\text{EdgeLabel}_{\Theta}(V)\).

A state label is made up of literals and partly characterizes the current state of the system, while an alias is a link to a referenced node. The node identifier appearing in an alias for \(F\) has to refer to a node of a service preceding \(F\) or \(F\) itself. \(k\) is the number of variables under the scope of the referenced node. Edge labels express input and output messages leading to a system state evolution. Most of the times, outputs (network messages) occur in direct reaction to inputs (phone message) according to some conditions on the state. Conditions on edge labels are required for the given input/output message exchange but not in the source node of the transition. We can now characterize diagrams of feature specifications:

\textsuperscript{4}If one wants to express that \(A\) prevents incoming calls from both \(B\) and \(C\), then \(A\) has to subscribe twice to the TCS service: \(A : TCS(B)\) and \(A : TCS(C)\).

\textsuperscript{5}When there is no ambiguity, we will simply identify the signature feature name to the name of the feature under specification.
Definition 4 A diagram $D = (\text{Node}, \text{Edge}, \text{label}, I, \text{nodelabel}, \text{edgelabel})$ for a feature signature consists of: a rooted tree $^6$, label : Node $\rightarrow N$ an injective application, $I$ the root set of declaration of phones and nodelabel and edgelabel the labelling functions respectively from $N$ to $\text{State}_\Theta(V) \cup \text{Alias}_\Theta(V)$ and from $\text{Edge}$ to $\text{EdgeLabel}_\Theta(V)$ such that each node labelled by an alias is a leaf of the rooted tree.

The application label is the numeration of nodes. Occurring phone variables should be declared by a subscription. Variables used in the root node are introduced by the root set of declarations. Other variables are declared on the edge where they appear for the first time in scenarios: a kind of local variables. At each node $n$, the set of free variables occurring in the subdiagram $sd$ stemming from $n$ contains all variables which are not under the scope of a subscription declaration occurring in $sd$.

Let us now introduce two well known services: CFB (Call Forward when Busy) and TCS. Figure 2 presents their diagrams. The subscription $B : \text{CFB}(C)$ leads $B$ to have incoming calls forwarded to the phone $C$ when $B$ is busy. The root node coincides with an intermediate POTS node. It ensures that the initial state of a service specification is reachable from a network with phones only subscribing to POTS. The edges labelled by $A.\text{call}(B)$ introduce $B$ and $C$ with the subscriptions $B : \text{CFB}(C)$ and $C : \text{POTS}$. The diagram of CFB consists of splitting the POTS call cases: adding edges with the message $A.\text{call}(B)$ and some conditions on the state of $B$ and $C$. A set of conditions labelling an edge is a conjunction of conditions. When they label edges with the same message they have to be mutually exclusive to ensure determinism. According to the three cases, CFB is triggered or not. In particular, the first left branch stemming from the $\text{dialing}(A)$ node is

\begin{tikzpicture}[scale=0.8]
  \node at (0,0) {\text{A-POTS}}; \node at (3,0) {\text{A-call(B)}}; \node at (6,0) {\text{A-onhook}};
  \node at (0,-3) {\text{B:CFS(C)}}; \node at (3,-3) {\text{B:CFB(C)}}; \node at (6,-3) {\text{C:POTS}};
  \node at (0,-6) {\text{start(rbtone)}.A}}; \node at (3,-6) {\text{start(rgtone).B}}; \node at (6,-6) {\text{start(lbtone).A}};
  \node at (0,-9) {\text{POTS(A)}}; \node at (3,-9) {\text{POTS(B)}}; \node at (6,-9) {\text{POTS(C)}};
  \node at (0,-12) {\text{POTS}(A)}; \node at (3,-12) {\text{POTS}(B)}; \node at (6,-12) {\text{POTS}(C)};
\end{tikzpicture}

\begin{tikzpicture}[scale=0.8]
  \node at (0,0) {\text{Dialing}(A)}; \node at (3,0) {\text{CFS}(A)}; \node at (6,0) {\text{A-\text{onhook}}};
  \node at (0,-3) {\text{B:TCS(A)}}; \node at (3,-3) {\text{A-call(B)}}; \node at (6,-3) {\text{C:POTS}};
  \node at (0,-6) {\text{Idle}(A)}; \node at (3,-6) {\text{Idle}(B)}; \node at (6,-6) {\text{Idle}(C)};
  \node at (0,-9) {\text{POTS}(A)}; \node at (3,-9) {\text{TCS}(A)}; \node at (6,-9) {\text{POTS}(C)};
  \node at (0,-12) {\text{POTS}(A)}; \node at (3,-12) {\text{TCSmsg(A)}}; \node at (6,-12) {\text{POTS}(C)};
\end{tikzpicture}

Figure 2: CFB (left) and TCS (right) diagrams.

^6A directed graph is a couple $(\text{Node}, \text{Edge})$ where $\text{Node}$ and $\text{Edge}$ are respectively the set of nodes and the set of edges (couples (source node,target node)). A rooted tree is a connected directed graph with a root: an identified node such that there is no cycle in the graph, any node being reachable from the root by a consecutive edge sequence.
similar to the corresponding POTS branch while the other ones are new and describe the feature execution. The branch labelled by \( \sim \text{idle}(B) \text{ idle}(C) \) means that if \( B \) is not idle and \( C \) is idle, then a communication between \( A \) and \( C \) is starting: \( A \) is listening to a ringing back tone and \( C \) to a ringing tone. There are two subdiagrams equivalent, up to some variable renaming, to the POTS subdiagram whose node identifier is \((\text{POTS}, 5)\). To make these similarities explicit, we use aliases, graphically represented by triangles. TCS prevents incoming calls from phones chosen by the subscriber. With the subscription \( B : \text{TCS}(A) \), the POTS communication attempt in response to \( A.\text{call}(B) \) is replaced by a new refusal answer, using a specific recorded message. We define feature specifications as diagrams provided with some additional properties on system states:

**Definition 5** Let \( \Theta \) a feature signature. A constraint is a formula \( l \Rightarrow l' \) where \( l \) and \( l' \) are literals over \( \Theta \) such that phone variables occurring in \( l' \) also occur in \( l \). A feature specification \( F = (D, C) \) over \( \Theta \) is composed of a diagram \( D \), denoted \( D_F \), and a set \( C \) of constraints over \( \Theta \), denoted \( C_F \).

Constraints are invariants on the set of reachable states. For example, \( \text{talking}(X, Y) \Rightarrow \sim \text{dialing}(Y) \) is a POTS constraint specifying that a phone cannot be both talking and dialing at the same time. Each feature inherits constraints from \( \text{POTS} \) and adds new ones, mainly when introducing new literal symbols (as \( \text{playing} \) for TCS). The restriction on the use of variables allows to deduce from a state condition all literals necessarily true w.r.t a set of constraints, which is only possible if \( l' \) does not introduce fresh variables.

We have mentioned some criteria to qualify a specification \( F = (D_F, C_F) \) as well-formed. They concern the proper use of variables, alias nodes (use of variables and reference to defined nodes) and the consistent use of atomic properties within \( D_F \) w.r.t constraints in \( C_F \). We will not detail these criteria here, but all of them are required to compute interactions.

### 3 Our method of interaction detection

Our detection method combines two major steps. The first step extracts the triggering informations of a feature \( F \) from its specification \( (D_F, C_F) \). It identifies the points in \( D_F \) which differ from the POTS diagram. While detecting these divergences, the algorithm collects contextual information such as subscriptions, states and triggering message. Moreover, \( D_F \) is annotated with intentions. Let us illustrate this notion considering the subscriptions \( B : \text{CFB}(C) \) and \( C : \text{TCS}(A) \). When \( A \) calls \( B \) in a busy state, CFB asks the network to react as if it had received the intentional message \( A.\text{call}(C) \).
instead of the initial message \textit{A.call(}B\textit{)}. But an interaction occurs with \\ \textit{A.call(}B\textit{)} since the simulated message \textit{A.call(}C\textit{)} could originate the refusal message \textit{TCSmsg} to \textit{A} while CFB requires a connection between \textit{A} and \textit{C}. We qualify such interactions as indirect (called semantical in [22] or Sequential Action Interaction in [7]). Some other information to complete the computed intentional informations is state condition (concerning the status of \textit{idle(}C\textit{)}). In few words, the first step consists of computing the pertinent triggering informations given with the intentional information.

The second step computes interactions by exploiting triggering information and associated intentional information. This algorithm is thus an interaction elicitation algorithm. In particular, it automatically infers the critical call configuration and triggering message leading to a given interaction.

\section*{3.1 First step : computation of triggering information}

To compute the triggering information of \textit{F}, we compare \textit{D\textsubscript{F}} to the default diagram POTS. The exploration is done according to a breath first strategy and to the building of a variable renaming from variables of \textit{D\textsubscript{F}} to the ones of \textit{D\textsubscript{POTS}}. Renaming allows to superpose the diagrams while preserving the rôles of the variables inside each one. The principles are the following ones:

- We look for a node in POTS coinciding with the root node of \textit{D\textsubscript{F}}, up to a variable renaming \(\lambda\) in order to initiate the diagram comparison.
- The comparison is pursued by comparing the immediate subdiagrams. Diagram paths are recursively pairwise compared. The variable renaming \(\lambda\) is extended to the new encountered variables while descending the compared paths. We extract the set of paths such that the first edge of the path cannot be found in POTS. The edge source is the activation point, the edge phone message is the triggering message, the union of the edge condition and the source node label is the triggering condition, and the subdiagram linked to the edge target is the activation subdiagram.
- We elicit intentional messages by looking for a network answer in POTS similar to those of the activation subdiagram. If the search is successful, the phone message at the origin of the corresponding POTS subdiagram is the intentional message we are looking for. To each intentional message is associated an intentional condition: the union of the condition labelling the corresponding POTS edge and its source node label. We also search for intentional messages inside the activation subdiagram.

To conclude, triggering information is basically made up of a call configuration, a triggering message, a triggering condition, an annotated acti-
ination diagram\(^7\), intentional message and condition. The information computed for CFB is synthesized in the following table. Confronting CFB with POTS identifies the CFB dialing\((A)\) node as activation point. From this node, the diagram contains two branches sharing the \(A.call(B)\) message and the \(\sim idle(B), \text{dialing}(A)\) condition but respectively with the \(\text{idle}(C)\) and \(\sim \text{idle}(C)\) conditions. The triggering message is \(A.call(B)\). Computed intentional messages are in the first case \(A.call(C)\) (with the \(\text{idle}(C)\) condition) and in the second case \(A.call(C)\) (with the \(\sim \text{idle}(C)\) condition).

<table>
<thead>
<tr>
<th>Call configuration</th>
<th>Triggering msg.</th>
<th>Condition</th>
<th>Intentional msg.</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A : \text{POTS}) (C : \text{POTS}) (B : \text{CFB}(C))</td>
<td>(A.call(B)) (\sim \text{idle}(B), \text{dialing}(A)) (\text{idle}(C))</td>
<td>(A.call(C)) (\text{idle}(C)) (\text{dialing}(A))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(A : \text{POTS}) (C : \text{POTS}) (B : \text{CFB}(C))</td>
<td>(A.call(B)) (\sim \text{idle}(B), \text{dialing}(A)) (\text{idle}(C))</td>
<td>(A.call(C)) (\sim \text{idle}(C)) (\text{dialing}(A))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the TCS diagram, the two POTS branches for the \(A.call(B)\) message are replaced by a new branch: the network sends the playing of a recorded refusal message for which any intentional message cannot be found.

<table>
<thead>
<tr>
<th>Call configuration</th>
<th>Triggering msg.</th>
<th>Condition</th>
<th>Intentional msg.</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A : \text{POTS}) (B : \text{TCS}(A))</td>
<td>(A.call(B)) (\text{dialing}(A))</td>
<td>none (\text{none})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.2 Second step: elicitation of interactions

To detect interactions between two features \(F_1\) and \(F_2\) (possibly \(F_1 = F_2\)), we apply static analysis to their respective triggering information. Our search is decomposed into three main steps according the kind of interactions we are looking for. At first glance, there are three principal cases explaining that invocation of \(F_1\) causes an interaction with \(F_2\). First, \(F_1\) and \(F_2\) share the same triggering message (direct interaction). Second, the intentional message associated to the triggering message of \(F_1\) coincides with the triggering message of \(F_2\) (indirect interaction). Third, the activation of \(F_1\) is likely to activate \(F_2\) via real or intentional messages. The third step simply amounts to apply in the first and second cases at each edge of the activation diagram.

For \(i = 1, 2\), let \(CC_i, TM_i, C_i\) be respectively a call configuration, a triggering message and a triggering condition associated with \(F_i\). Let \(IM_i\) and \(IC_i\) denote the intentional message and associated condition when they exist. The comparison begins with the search of a variable renaming \(\lambda\) between the two variable sets for \(F_1\) and \(F_2\) such that it unifies the (triggering or intentional) messages of \(F_1\) and \(F_2\). It is also required that triggering conditions are compatible. It means that \(\lambda(C_1) \cup \lambda(C_2)\) is a consistent state condition w.r.t. the constraints of both \(F_1\) and \(F_2\). As soon as the comparison

---

\(^7\)Due to lack of space, such diagrams will be left implicit in the sequel.
of messages provides us with such a variable renaming, there is an interaction. An interaction is composed of a call configuration \( CC \) already defined, a triggering message \( TM \), and a state condition \( TC \), already computed as \( \lambda(C_1) \cup \lambda(C_2) \):

- **Direct interactions** (\( \lambda(TM_1) = \lambda(TM_2) \)): \( TM = \lambda(TM_1) \).
- **Indirect interactions** (\( \lambda(IM_1) = \lambda(TM_2) \) or reciprocally). To observe an interaction, we then require that the conditions \( \lambda(IC_1) \) and \( \lambda(C_2) \) are consistent w.r.t. constraints of \( F_1 \) and \( F_2 \). In other words, when \( F_1 \) is triggered, the simulated behavior meets the triggering condition of \( F_2 \) triggering. \( TM \) is the message \( \lambda(TM_1) \) at the origin of the intentional message \( \lambda(IM_1) \). Such an interaction can be understood as: “in the configuration \( CC \), when \( F_1 \) is triggered on the message \( TM_1 \), all happens as if the network had received the message \( IM_1 \), at the origin of an indirect \( F_2 \) triggering”.

We do not detail the case of indirect or direct interactions occurring with messages inside an activation diagram.

We can decide whether a detected interaction is sensible or not. If after a joint triggering of \( F_1 \) and \( F_2 \), the two activation diagrams are similar, then the interaction is perceived as *invisible*. Else, it is a *visible* one and there really exists a difference between service requirements.

The interactions between TCS and CFB are automatically computed from their diagrams and their associated constraints.

### 4 A case study

In this section, we present the results of a case study ordered by France Telecom (the historical French phone company), considered as a benchmark to evaluate our technique and involving three services: a voice mail service (VM), call waiting (CW) and an automatic recall service (RBF; for Ring Back when Free). They have been chosen because they were complex enough not to be considered as toy examples and not to allow experiments to be run by hand. These three services have been derived from informal specifications of France Telecom services.

Our aim was to check whether the found interactions were the good ones or not, and whether all interactions were found or not. We received a real positive feedback with regard to the high level of qualitative description of interactions and with regard to the fact that our method is fully automatic,
without any additional knowledge given by an expert (like indications on the
number of phones to consider with or without subscriptions to some services).

4.1 The three services and their triggering informations

RBF allows the subscriber $A$ to automatically recall a phone $B$ that was
not idle at the first call: $A$ is offered to recall $B$, as soon as $B$
is idle again and before 30 minutes passed (choice materialised by the $RBFmsg$
message). When $B$
is idle again, the network calls $A$. If $A$
go off hook, the network calls $B$ (if still idle). The RBF dia-
gram (Figure 3) contains 11 nodes, and 26 when all aliases re-
moved. A constraint
is added, specifying
that playing the specific message $RBFmsg$ is incompatible with being idle.
Computing of the triggering information for RBF gives two activation points:
the call back phase and the initialization of call back (when the subscriber
calls a non idle phone). For the automatic call (first column of the table),
RBF is not triggered by any phone message but by a change of state. RBF
needs sorts to specify times and durations (with operations like $+$, $\ldots$). As
our formalism lacks expressivity, we represent $currentdate$
by a constant. For
initialisation of automatic call (second column in the table), instead of emit-
ting the line busy tone, $A$ plays a message advertising that it can put $B$ on
automatic call.

![Figure 3: diagram of RBF](image)

<table>
<thead>
<tr>
<th>Subscription</th>
<th>Automatic call</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A: RBF$</td>
<td>$A: RBF$</td>
</tr>
<tr>
<td></td>
<td>$B: POTS$</td>
<td>$B: POTS$</td>
</tr>
<tr>
<td>State cond.</td>
<td>$idle(A) idle(B)$</td>
<td>$dialing(A)$</td>
</tr>
<tr>
<td></td>
<td>$currentdate &lt; dateofcall(B, A) + 30mn$</td>
<td>$\sim idle(B)$</td>
</tr>
<tr>
<td>Triggering msg.</td>
<td>none</td>
<td>$A.call(B)$</td>
</tr>
<tr>
<td>Net. msg.</td>
<td>$start(ringingtone), A$</td>
<td>$startplaying(RBFmsg), A$</td>
</tr>
<tr>
<td>Activation diag.</td>
<td>describes the automatic call back an intentional message (when $A.offhook$ and $idle(B)$) $A.call(B)$ $idle(B)$</td>
<td>initialisation of the automatic call back</td>
</tr>
</tbody>
</table>
CW is the feature allowing its subscriber to accept a second call when he is already in communication. The subscriber hears a special tone CWTone and can then switch between the two conversations or let the second caller give up by hanging up. When the subscriber is in conversation with one of the phones he has on line, the other is on hold. The diagram contains 32 nodes, including 13 aliases (2 of them are loops), and after removal of unnecessary aliases, 50 nodes (including 16 loops). We obviously cannot treat such a specification by hand.

When the subscriber to VM is busy or does not answer to a call, a welcome message invites the caller to record a message in the subscriber’s message box. The diagram of VM contains 9 nodes. VM uses a sort for played message such as VMwelcome...TMtimeout is a constant specifying the delay at the end of which the subscriber B is considered as not responding. VM is activated at one point in two different ways: when the subscriber is busy or does not answer after the delay (see the table below).

<table>
<thead>
<tr>
<th>Subscription</th>
<th>A : CW B : POTS C : POTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>State cond.</td>
<td>talking(A, B)</td>
</tr>
<tr>
<td>Trigger. msg.</td>
<td>C.call(A)</td>
</tr>
<tr>
<td>Net. msg.</td>
<td>start(CWTone).A startplaying(ringbacktone).C</td>
</tr>
<tr>
<td>Activ. diag.</td>
<td>describes the double comm. if A accepts the second call</td>
</tr>
</tbody>
</table>

**Figure 4:** CW : activation

<table>
<thead>
<tr>
<th>busy subscriber</th>
<th>non answering subscriber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscription :</td>
<td>B : VM A : POTS C : VM A : POTS</td>
</tr>
<tr>
<td>State:</td>
<td>dialing(A) ~ idle(B) emitting(A, ringbacktone) emitting(C, ringingtone) currentdate &lt; dateof(A.call(C)) + TMtimeout</td>
</tr>
<tr>
<td>Trigg. message</td>
<td>A.call(B) non</td>
</tr>
<tr>
<td>Activation diag.</td>
<td>recording of a message recording of a message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VM</th>
<th>RBF</th>
<th>CW</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>(5) (6)</td>
<td>(2) (4)</td>
</tr>
<tr>
<td>RBF</td>
<td>-</td>
<td>(1) (3)</td>
</tr>
<tr>
<td>CW</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Figure 5:** Interactions

**4.2 Detected interactions**

6 interactions are automatically found as shown in Figure 5. A cell in the tabular is referenced by a pair \((f, f')\) where \(f\) and \(f'\) are feature names resp. used in column and line. A cell \((f, f')\) contains either a possibly empty set of numbers or a dash. The numbers reference interactions in the sequel. The dashes concern symmetric interactions. We do not detail interactions (5) and (6).

**Interaction 1 (RBF vs RBF):** two subscribers to RBF, B and A, have called a busy POTS subscriber X; two automatic calls have been initialized. But, if X becomes idle again during

**Figure 6:** Interaction 1
the delay, which of $A$ and $B$ should recall $X$? This choice is represented by the network answers \texttt{start(ringingtone).A} and \texttt{start(ringingtone).B}.

**Interaction 2 and 3 (RBF vs VM and RBF vs CW):** when an RBF subscriber $A$ calls a VM subscriber $B$, if $B$ is busy, according to specifications, both an automatic call of $B$ should be initialized for $A$ (\texttt{startplaying(VMwelcome).A}) and it should be offered $A$ to record a message for $B$ (\texttt{startplaying(RBFmsg).A}). The interaction between RBF and CW is quite similar, thus not detailed. Interaction 2 had been discovered late in the France Telecom design process.

**Interaction 4 (CW vs VM)** $B$, subscriber to CW and VM is called while busy, so VM should be triggered. But, as he is talking to $C$ (notice the role played by constraints), CW should be triggered too.

Figure 7: Interactions 2 and 3

Figure 8: Interaction 4

We assert that the key issue when dealing with feature interaction detection is not to find a large number of interactions. For example, \cite{21} computes 14 interactions between TCS and CFBL, but does not give any hint about the meaning of each detected interaction. We prefer to compute few but pertinent interactions. What does “pertinent” mean for us? We simply refer to the expertise of service designers\(^{8}\). Our results are very satisfactory because the number of found interactions is reduced and because they can be managed and understood by an expert. A good method of interaction detection has to be automatic without involving extra knowledge, and has to provide tractable information about the computed interactions. In \cite{14}, we have applied our method on all the services given in the international benchmarks \cite{12,17} in order to fit our approach with the intuition. We have also compared our results with previous works related to these benchmarks in so far as precise descriptions are available. Unlike most of other works, the main advantage of our approach is that our interaction detection method includes the computation of the corresponding call configuration and returns knowledge useful to understand the interactions.

5 Adapting our technique to VoIP services

We shortly describe the way we are currently adapting our work to the VoIP service challenge. We do not care neither about problems caused by real time

\(^{8}\)Notably at France Telecom
Voice traffic over IP, nor about Quality of Service. Some other works ([21], [20]) are also concerned with VoIP services.

VoIP. Voice over IP concerns voice traffic using the Internet Protocol (IP). Circuit switching is a quite expensive solution for carrying voice and is not so suitable for new requirements such as exchanges of email, video, instant messaging, etc. IP combines data transactions with voice transport and defines a lot of useful features, involving for example, mail, phone calls, etc. For example, stored voice messages can be accessed from a mailer. Nowadays, with VoIP, more intelligence is placed inside the user devices connected to the network. A lot of features can be locally triggered rather than being subscribed and executed by the network. IP phone, browser, telephony software, etc. are such devices.

SIP. We consider applications running in the connected user devices, and so, protocols of the application layer. SIP, Session Initiation Protocol ([13]) is a client-server protocol used for the management of sessions (setup, modification and ending). Some SIP entities (proxy, redirect server, user agent, etc) are clients, some other ones are servers or play both rôles. SIP opens sessions between User Agents (UA): each SIP user device implements the capabilities of a User Agent entity. A client is an entity emitting SIP requests (also called methods) such as the invite request, used to ask for the establishment of a session between two user agents. A server answers to SIP requests with responses (trying, ringing, ok, moved temporarily, etc). We do not detail here all the SIP entities. A user owns a SIP address, which takes the form of an email address. One or more (usual or temporary) IP addresses can correspond to this address. Proxy servers and redirect servers are entities whose main rôles are to assign IP addresses to SIP ones and to carry out appropriate actions. A proxy relays the requests to the suitable IP addresses while a redirect server informs the client of new addresses.

We briefly introduce two examples of services: a "one number service" (ONS) and a service of "temporarily forward incoming calls" (TFIC). The ONS feature allows the owner of a SIP address to be contacted at several IP addresses. In the specification focus (branch (a)) of Figure 9, a user agent A corresponds to the SIP address Asip and to the IP address Aip. It has to contact the Bsip SIP address but does not know the corresponding IP address. The invite request sent by Aip in direction of Bsip is relayed by a proxy server to the IP addresses to which Bsip has registered: Bip1 and Bip2. In the same time, Aip is informed by the proxy a session initiation is being attempted.
Let us remark that IP addresses play the rôles of the phone variables and that the notion of subscription has been modified. Roughly speaking, SIP addresses inherit from a kind of subscription in order to associate feature with SIP addresses. In the previous example, the Bsip address benefits from the ONS service with two numbers Bip1 and Bip2. We denote this by Bsip : ONS(Bip1), Bsip:ONS(Bip2), with ONS as a feature name and Bip1 and Bip2 as parameters. The service of Temporarily Forward Incoming Calls (TFIC) allows the owner of a SIP address to inform the one who wants to get in contact, that he/she has moved from the usual SIP address to another SIP address. In the focus (edge (1) followed by edge (2)) of Figure 10, the caller is the IP address Aip, the called SIP address is Bsip. The owner of Bsip has moved temporarily to a place where he/she can be contacted with the Bsip’ SIP address, formalised by Bsip : TFIC(Bsip’). TFIC involves a redirect server. Contrary to the proxy used for ONS, the redirect server doesn’t relay the request to the IP address to contact but informs the caller that the SIP address to contact has changed, with a moved-temporarily response. Aip can then contact Bsip’ instead of Bsip.

**Interactions**  By adapting our approach, we can find two interactions between ONS and TFIC. The first one can occur when they are triggered at the same time, and the other one when TFIC is first executed and then ONS can be triggered inside the activation diagram. The first one is found by comparing branch (a) of ONS to the branch (1) of TFIC. When the request Aip.invite(Asip, Bsip) is sent, the network can send the response moved-temporarily(Asip, Bsip, Bsip’). Aip or the set of messages trying.Aip, invite(Asip, Bsip). Bip1, invite(Asip, Bsip). Bip2. There are clearly two different reactions specified for such an invite request when the SIP address Bsip benefits from ONS and TFIC. Let us suppose that TFIC has been triggered (edge (1)). Aip will then send a new invite request, to Bsip’ instead of Bsip. The comparison of (edge (2)) of TFIC to branch (a) of ONS reveals that with TFIC, Aip would receive a ringing response and with ONS, a trying response. This occurs when the SIP address Bsip has TFIC with Bsip’ as temporary address, and when Bsip’ has subscribed to ONS. This is the second interaction.
6 Conclusion

In this paper, we have proposed a new off-line detection method for detecting interactions from telecommunication service specifications described by means of diagrams. Our diagrammatic notations resemble state-based and Chisel-like notations and deal with variable declaration, variable scope, loop mechanisms and aliases. The detection method is based on static analysis technics and is implemented in Prolog with two algorithms. The first one compares a service specification to the basic system specification in order to find the triggering information of the service. The second one compares the triggering information of two services in order to find interactions between them. We also give results for a short case study based on conventional services. The results are conclusive: the descriptions of detected interactions are legible and comprehensible such that designers of services at France Telecom have been convinced. They appreciate the full automatic approach and the precise description of the computed interactions including a call configuration. We show on a short example how our method can be adapted to VoIP and especially to services with SIP. This adaptation is promising but our formalism has to become more generic (w.r.t. a more flexible notion of data).

References


