Modeling and Analysing Telephone Features Using HMSCs

Loïc Hélouët
IRISA
Campus de Beaulieu
35042 Rennes Cedex, France
33 2 99 84 74 06
helouet@irisa.fr

Ferhat Khendek
Electrical and Computer Engineering
Concordia University
1455 de Maisonneuve Bd
Montreal, Quebec, Canada H3G 1M8
(514) 848-3081
khendek@ece.concordia.ca

ABSTRACT

This paper introduces a formal framework for the description and simulation of telephone features using High-level Message Sequence Charts (HMSCs). For that purpose HMSCs are extended with variables to describe guards for the triggering conditions and feature requirements. This formal framework takes into consideration and defines formally two types of interactions: triggering conflicts and requirement violations. These two types of interactions are detected using a HMSC simulation technique.

Keywords

Message Sequqnce Charts, feature interaction, requirement analysis.

1 INTRODUCTION

It is generally admitted that the later an error is detected during software development, the more time and effort it will need to be corrected. Errors can even be introduced during early phases of the software life cycle, when requirements are inconsistent. Tools for analysis of requirements can be used, but need to introduce formal models at a stage where notations are generally informal. The use of scenarios is a good compromise: they are clear and can be easily understood, but yet allow for formal manipulations.

The emergence of new architectures for telephone systems has made the need for consistency checking at the requirement level a crucial problem. Telephone systems based on Intelligent Networks now allow users to add new functionalities (called features) to the initial set of services provided by their operator. The proposal of new services has rapidly become a commercial argument, imposing time-to-market constraints for the definition of new functionalities. Unfortunately, a new feature behaving correctly in isolation can interferes negatively with other features, or even prevent the network from working properly. This situation is called feature interaction, and is seen as a problem that may hinder Intelligent Networks and other new architectural frameworks from achieving their goal of rapid development and deployment of new features.

Different approaches have been proposed for tackling the feature interaction problem. A good overview and a classification of these approaches can be found in [5]. Formal detection techniques generally consist of modeling a system using a formal notation, and then exploring the state space of the system for undesirable properties. We propose a formal requirement modeling based on scenarios, allowing for feature interaction detection. Scenarios are intuitive, allow to hide details that are not significant at the requirement stage, and yet allow for formal processing of the models. A scenario approach based on interworkings (scenarios with synchronous communications) is proposed in [1]. The approach defined is this article is based on a standardized scenario language, Message Sequence Charts (MSCs), and supposes asynchronous communications, which allows more elaborated behaviours. MSCs define in a graphical and intuitive manner the behavior of a system under consideration. In our approach, the role of the network is minimized and the platform considered as one central entity that provides responses to user requests (this kind of modeling is often referred as "user oriented view" of telephone systems).

A telephone system is modeled by mean of several user views, which define the behaviour that each user expects from the network, and a set of features A feature defines a behavior that must replace the normal behavior and a goal (also called *requirement* [8]), that is expressed by a logical property on variables. In order to express these requirements, the MSC standard notation is extended with variables.

The framework proposed allows for the formal characterization of two kinds of interactions that can be detected through simulation. The first class of interactions is called "triggering conflict", and appears when two features activated by a single user can be triggered after the same sequence of events. It is characterized by a non-deterministic behavior in the system model. The second class of interactions is called "requirements violation", and appears when the requirement of a feature is violated in the context of other features.

The paper is structured as follows. Section 2 introduces standard MSCs and extends this notation with variables manipulation. Section 3 introduces the model used for describing features. An approach for feature interaction detection within this framework is given in section 4 and illustrated through examples, before conclusion.

2 Message Sequence Charts

One of the advantages of a scenario-based approach is to allow for a graphical and intuitive representation of a system. Compared to the interleaving semantics of state based models, scenarios provide a visual and intuitive representation of concurrency that ease the understanding of a specification. This section introduces MSC, a graphical formalism standardized and maintained by ITU-T [11] and an extension to allow for variables manipulation for the specific purpose of requirements modeling.

Extended bMSCs

A basic MSC (bMSC) describes the behavior of communicating entities, called instances. These instances are represented by vertical axes, and message exchanges are represented by arrows from the sending to the receiving instance. An event in a bMSC can be a sending event, a receiving event, an operation on a timer, or an internal action. A message sending precedes its reception, and events are ordered along an instance axis. Therefore, a bMSC defines a *causal order between events*. Until very recently, MSC did not allow for the use of variables and data. However MSC'2000 will extend MSC'96 with data and variables manipulation. In this paper, we define and use a slightly different extension for handling variables.

In the model described in Section 3, feature triggering is guarded by a predicate on *variables*. Variables provide information on the state of the system, and may be modified by the execution of events. A MSC event can be parameterized. One may also want to describe generic behaviors, i.e. generic responses from the network to the instances. For this purpose, *instance variables* are also introduced in our model. For an illustration purpose, let us consider the MSC in Figure 1. Events e_1 and e_6 in this specification update a single variable T. Event e_4 is performed by an instance specified by variable u. u depends on the value fixed when A dials a number and will be instantiated during the execution (simulation) of the specification.

The approach described in this paper aims at modeling networks from the user point of view. Internal details of the network are hidden. The network is modeled in terms of interactions (requests/responses) with the users. Therefore, specifications will only contain bMSCs defining message exchanges between one user and the network (i.e. a bMSC with only two instances). Any communication with other instances is performed through instance variables.

A bMSC can be defined as a tuple $M = (E, \leq, Var, VI, \alpha, v, A, I)$, where:

- E is a set of events,
- $\leq$ is a partial ordering on these events,
- Var is a set of global variables,
- VI is a set of user variables,
- α is a function associating an instance and an action name to each event ($\alpha = E \longrightarrow A \times I \cup VI$),
- v is a function assigning a set of modifications on variable values to each event,
- A is a set of action names,
- I is the set of instances of the bMSC, i.e. the name of the user and the network.

For an event e, we write $\phi(e) = i$ when $\alpha(e) = (a, i)$. Intuitively the meaning of the mapping ϕ is "event e is performed by instance i". To illustrate our definitions, let us consider again the example in Figure 1. This bMSC describes a simple communication between a user A and the network N. Notice that the variable T is set to *true* by event e_1, and set to $false$ by event e_6. Event e_4 is performed by an instance whose name depends on the choice of actions performed by instance A. Furthermore, we have:

- $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$,
- $\leq = \{(e_1, e_2); (e_1, e_3); (e_1, e_4); (e_1, e_5); (e_1, e_6); (e_2; e_3); (e_2, e_4); (e_2, e_5); (e_2, e_6); (e_3, e_4); (e_5, e_6)\}$,
- $Var = \{T : Boolean\}$,
- $I = \{A, N\}$,
- $VI = \{V\}$,
- $\alpha = \{(e_1, !dial(u), A); (e_2, ?dial(u), N); (e_3, !call(u), N); (e_4, ?call(u), u); (e_5, !ringing, N); (e_6, ?ringing, A)\}$,
- $v = \{(e_1, T := true); (e_6, T := false)\}$.

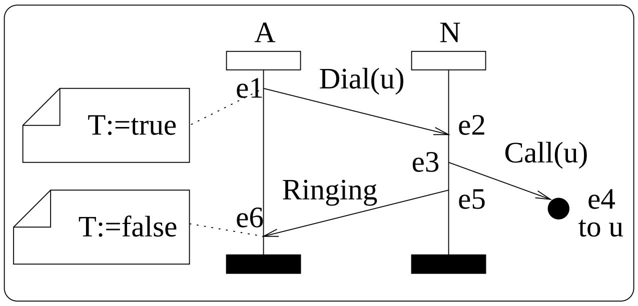

Figure 1: bMSC with a state variable and a variable instance.

HMSCs

As bMSC only describe finite and simple behaviors, a higher-level notation is needed to define more elaborated specifications. HMSCs are a kind of high-level graphs that combine bMSCs using parallel, loop or sequence operators. As the simulation framework defined further in this paper uses the results of [7], the set of operators is restricted to loop and sequence. A HMSC is defined as a graph $H = (N, \longrightarrow, M, l, End)$, where:

- N is a set of nodes,
- $\longrightarrow$ is a set of edges,
- M is a set of basic MSCs,
- l is a labeling function assigning a bMSC or a type to each node,
- $l : N \longrightarrow M \cup \{end, start, connect\}$, End associates a reference to a node of a normal behavior, indicating how the specification should behave when this end node is reached.

A detailed description of HMSCs is given in [9].

3 Modeling Telephone Systems with MSCs

This section describes MSC models of the Plain Old Telephone System (POTS) and few features as well as a semantic interpretation of these models. MSCs were already used as an input format for an axiomatic approach of feature detection in [2]. In this approach, simple MSCs with synchronous communications were used as a convenient way of defining a transition system for axiomatic rules. The approach presented hereafter considers iterated behaviours and asynchronism, and deals with axiomatics and behavioral aspects of features.

A MSC Model for POTS and Features

The main idea behind the model proposed hereafter is to define a set of possible behaviors for each user of the network. A telephone network may be performing concurrently more than one task for one user. For example, in the normal behavior, when a user u is dialing, the network has to process the call, and concurrently send a busy answer as a response to any call from other users. The first task can be seen as normal message exchanges between a user and the network. The second one describes a reaction of the network to an external stimulus (i.e. a message that is not sent by u), according to the state of u. Notice that an external stimulus to the network representation of u only exists if there is a corresponding message sent by another user u' to its network representation.

Definition 1 A *Model* of a telephone network is a set $\mathcal{M} = \underset{u \in Users}{\{B_u\}}$ of behaviours where each B_u describes a set of message exchanges between a user u and the telephone network.

Definition 2 A *Behavior* is a couple $B_u = (N_u, \mathcal{F}_u)$ where N_u is the set of communications between a user u and the network when no feature is triggered. $\mathcal{F}_u$ describes a set of *features*, i.e. a set of interactions that may replace the normal behavior.

Definition 3 A *Normal Behavior* is a couple $N_u = (P_u, \mathcal{R}_u)$. P_u is an HMSC called the *normal protocol* that defines the normal exchanges between user u and the network. $\mathcal{R}_u = \{\underset{h \in 1..H}{(g_h, H_h)}\})$ is a set of guarded HMSCs, called the *reactions* of the network. Figure 3 shows an example of the normal behavior of a user as described by the POTS specification of [6].

Definition 4 A *reaction* is a response of the network to an external stimulus depending on the state of user u. g_h is a guard, i.e. an expression on a set of variables that can be evaluated to true or false. Each HMSC H_h describes how the network should react to an external message when the guard g_h is true. H_h should not contain messages originating from user u, and should not contain any loop.

An example of reaction is the expected response from the network to a user when the called subscriber is busy. The busy answer must be sent without warning the busy subscriber (except if a feature specifies another behavior). Let us suppose that a variable *A_Busy* is set to true whenever user A is busy. Then, this expected reaction in a normal behavior can be modeled by a couple $(g_1 = A\_busy, H_{AB})$, where H_{AB} is the HMSC of Figure 2. Reactions can be concurrent (when two different users call a busy subscriber, the network can respond concurrently to them).

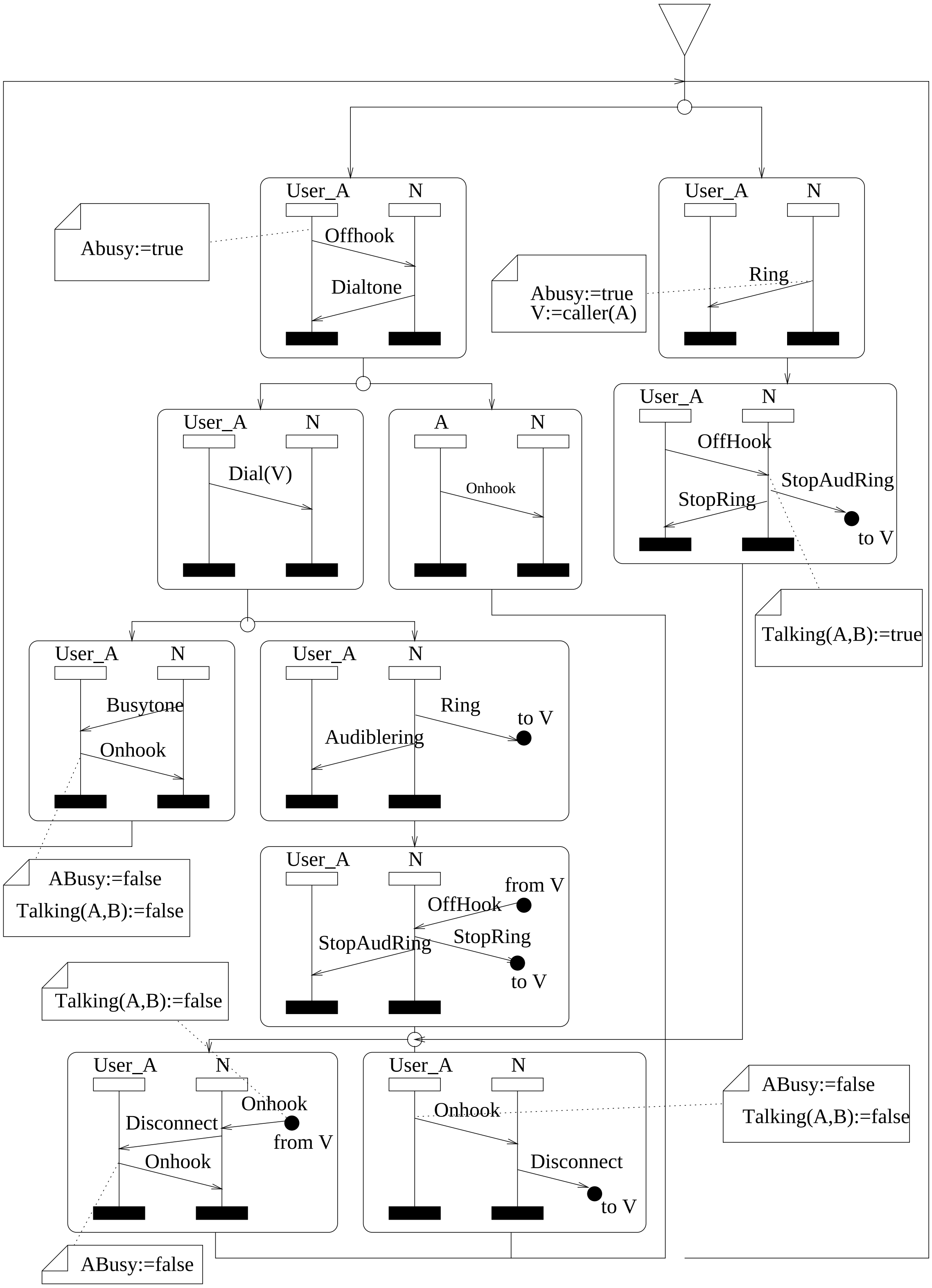

Figure 3: HMSC specification of the POTS.

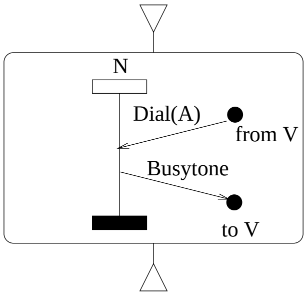

Figure 2: HMSC H_{AB}

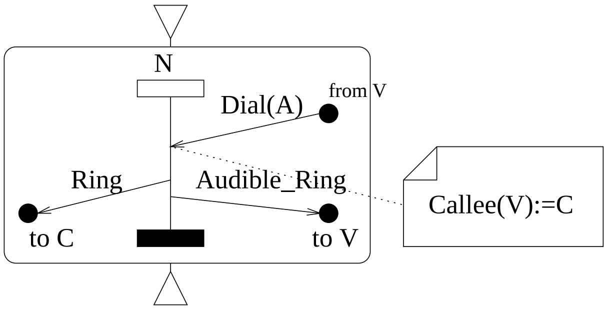

Figure 4: HMSC H_{CFB}: Call Forward on Busy

In our approach, features are classified into two categories: the first category of features (called type 1 feature hereafter) defines a set of new possible network behaviors that do not involve execution of user events. This kind of features is modeled by a couple $f = (\mathcal{R}_f, I_f)$, where:

- $\mathcal{R}_f = \{(g_h, P_h)\}$ is a set of guarded reactions. and I_f is a feature requirement. P_h is a HMSC containing no iteration, and no communications between the network and the user. Furthermore, all guards are exclusive.
- I_f is called the *requirement* of f, and is an invariant property that should be preserved during any execution.

Call Forward on Busy (CFB) is an instance of this category of features. A subscriber to the CFB can specify a number where any incoming call should be redirected when his line is engaged. This feature can be expressed by a tuple:

$$\begin{aligned} cfb = (\quad & \emptyset, \{(g_1 = whenA\_Busy \wedge \neg C\_Busy, H_{CFB}), \\ & (g_1 = whenA\_Busy \wedge C\_Busy, H_{AB})\}) \end{aligned}$$

Figure 4 provides the graphical representation of HMSC H_{CFB} for the CFB feature.

The second category of features (called type 2 feature hereafter) involves communications between the user and the network. This type of features is modeled by a tuple $f = (g_f, P_f, \mathcal{R}_f, I_f)$ where:

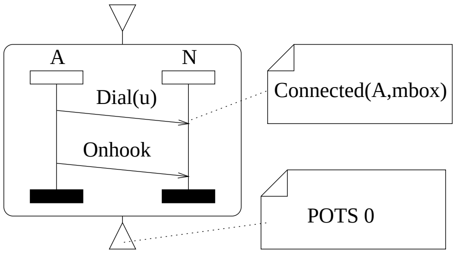

Figure 5: HMSC H_{ocs}: Outgoing Call Screening

- g_f is a guard, also called *triggering condition*, indicating when a feature can be triggered,
- P_f is the *feature protocol*, that replaces the normal protocol when the feature is triggered. The minimal event for P_f is called the triggering event. The feature is considered as triggered when this event has been executed.
- $\mathcal{R}_f$ is the set of reactions replacing the reactions defined in the normal protocol when f is triggered,
- I_f has the same meaning as previously.

Outgoing Call Screening (OCS) is an instance of the second category of features. This feature allows for a user to define a list of telephone numbers that should never be reached from his phone. This feature contains a behavioral part, which says that when a filtered number is composed, a message indicating that the call is screened must be delivered. From a user point of view, this feature can also be defined as: "A number from my screening list can never be reached using my phone". Let $Talking(u, u')$ be a boolean variable that is set to true when user u and user u' are connected and talking. Let $OCS\_list(u)$ be the screening list for user u. Then, the requirement for the OCS for a user A that must be preserved by any execution can be expressed as $\nexists u, Talking(A, u) \wedge u \in OCS\_list(A)$. OCS feature can be defined as a tuple

$$\begin{aligned} ocs = (\quad & g_{ocs} = u \in OCS\_list,\ H_{ocs},\ I_{ocs} = \nexists u, \\ & Talking(A, u) \wedge u \in OCS\_list(A)), \end{aligned}$$

where H_{ocs} is the HMSC defined in Figure 5.

The reason for differentiating the two categories of features is that in the first case, a subscriber defines a new set of reactions, which should not affect his normal exchanges with the network; while in the second case, a completely different set of possible scenarios is redefined by the triggered feature. These scenarios should overlap with the normal behavior. At the termination of a type 2 feature, the subscriber and the network will resume the normal behavior, and restart from the HMSC node that is specified in the end-node of the feature.

Execution of a set of user views
Due to a lack a space, we do not provide a complete operational semantics. For each user view, a transition system is defined using the HMSC simulation framework defined in [7]. When the triggering condition of a feature becomes true, then the behaviour of the feature replaces the initial behaviour of the system. The global behaviour defined by a set of user views is obtained by overlapping the different behaviours as perceived by the users: external event for an user A are unified with internal events of another user. We will note $unifiable(e_1, e_2)$ when two events can be unified (they have the same label modulo a variable instantiation). The global transition system obtained from a model has an initial state G_0. An execution of a model is a sequence w of events leading to a state G (we will write $G_0 \xrightarrow{w} G$).

Let us consider a simple execution. Users A and B are two telephone users that have not subscribed to any feature. Each user view is described by two HMCS: the POTS specification of Figure 3 as the normal protocol, and the busy reaction H_{AB}. The following table describes an execution in which A tries to reach B. We will write $A\ !m\ to\ N$ when instance A sends a message m to instance N, and $A\ ?m\ from\ N$ when instance A receives a message m from instance N.

Action	**Consequences**
A !Offhook to N	there is no unifiable event in B's specification, but now A is busy, variable A_Busy is set to $true$.
N ?Offhook from A	
N !Dialtone to A	
A ?Dialtone from N	
A !Dial(B) to N	there is an unifiable event v!Dial(B) to N in B's specification, so A and B's local states evolve. Furthermore, variable B_Busy is set to $true$
N ?Dial(B) from A	A and B's local states evolve

4 Feature Interaction Detection

A commonly used definition for feature interaction is given in [8]: "A feature interaction occurs when the behavior of one feature is affected by the behavior of another feature or another instance of the same feature".

Feature interactions can be classified into different categories depending on their causes, the number of involved instances, etc. [3, 8, 10]. This paper mainly focuses on two categories of interactions. The first class of interactions is due to triggering conflicts (two or more features can be triggered at the same time). The second class of feature interactions is due to violation of feature requirements.

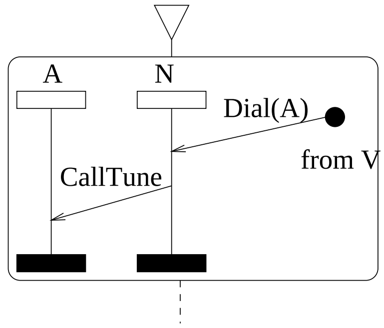

Figure 6: HMSC H_{cw}: Call Waiting

Detecting Triggering Conflicts
Within the framework, the triggering conflicts between two features is characterized by a *non-deterministic* situation during the simulation of the whole system. Searching for triggering conflicts consists in searching for the presence of states of the system from which two features can be triggered by the same action. A similar approach was proposed for a LOTOS based synchronous framework in [4].

Let $\mathcal{F}$ be a set of features. A pair of feature $(f_1, f_2) \in \mathcal{F}^2$, with guards g_1 and g_2 and triggering events e_1 and e_2 has a triggering conflict if and only if: $\exists w$ such that $G_0 \xrightarrow{w} G$ and $V(G) \models g_1 \wedge V(G) \models g_2$ and $G \xrightarrow{a} G' \wedge unifiable(a, e_1) \wedge unifiable(a, e_2)$.

Note that $(g_1 \wedge g_2 \neq false) \bigwedge unifiable(e_1, e_2)$ is not a sufficient condition for detecting that (f_1, f_2) is a feature interaction, as the specification may never reach a state in which $g_1 = true \wedge g_2 = true$. Let us consider the Call Waiting feature (CW) and Call Forward on Busy (CFB). CW feature allows for a user to receive a warning tone form the network when his line is engaged, and another user is calling him. The CW subscriber can then switch from one user to the other. The interaction between CW and the CFB is a well known case of interaction [3]. Now, let us see how the simulation framework can help detecting this interaction. We assume that a user A has subscribed to CW and CFB (to a third number C). The network view of A can therefore be described as a tuple $B = (N_A, \mathcal{F}_A)$, where N_A is the classical POTS normal behavior, and where $\mathcal{F}_A = \{CW_A = (Talking(A, x), H_{CW}, \emptyset, true); CF_{A>C}(\{(A\_Busy \quad \wedge$

$\neg C\_Busy, H_{CFB})\}, true)\}$. Figure 4 provides the graphical representation of HMSC H_{CFB} for the CFB feature. Figure 6 describes the protocol part H_{CW} of the CW feature. Notice that this feature is very large and only the signaling part is described. Let us consider the following execution:

	Action	**Assignments**
1	A !Offhook to N	A_Busy:=true
2	N ?Offhook from A	
3	N !Dialtone to A	
4	A ?Dialtone from N	
5	A !Dial(B)	B_Busy:=true
6	N ?Dial(B)	
7	N !Ring(B)	
6	N !Ringing	
8	A ?Ringing	
9	B ?Ringing	
10	B !Offhook	
11	N ?Offhook	Talking(A,B):=true

Now, if a user D dials A's number, CFB and CW features of A can both be triggered. Therefore, a feature interaction has been detected. Studying the intersection of guards on each pair of features for each user view can help detecting non-determinism. However, this is not sufficient, as guards may be formulas on distinct variables that turn to be equivalent during execution. Consider the guards A_busy of CFB and $Talking(A, B)$ of CW. From the specification, it is obvious that $Talking(A, B) \Longrightarrow A\_Busy$, but a syntactical check without this knowledge would have failed to detect the interaction between CFB and CW.

Detecting Feature Requirement Violation

This class of interactions happens when the requirements of the features under consideration are not compatible. A feature requirement can be defined as an invariant property, which should always hold when the feature is activated. The requirement I of a feature is said to be *violated* if there is a reachable state in which the property defined for the feature is false: $\exists w$ such that $G_0 \xrightarrow{w} G$ and $V(G) \not\models I$

Due to the asynchronous communications, which may cause unbounded buffering of messages, an exhaustive search of the specification is not possible, as the underlying transition system may be infinite. The method proposed is to run an interactive simulation, and try to reach a state where an invariant is violated. Let us now consider the well known interaction between CFB and OCS. Let us assume that a user B has redirected his calls to C, A has subscribed to the OCS feature and has put C in his screening list. Then, states such that A and C are connected and talking should never be reached. Consider the following execution:

	action	**Assignments**
1	B !dial(D)	Busy_D := true; Busy_B := true
2	A !Dial(B) to N	Busy_A := true; CFB triggered
3	N ?Dial(B) from A	
4	N !Ring(C) to C	Busy_C := true
5	N !Ringing to A	
6	C ?Ringing from N	
7	C !offHook to N	
8	N ?offHook from C	Talking(A,C):=true

During the simulation, a state where the OCS requirement is violated have been reached . Therefore, the requirements of CFU and OCS are inconsistent.

5 Conclusion

This paper proposes a formal framework for telephone networks and features modeling. This framework is based on a scenario representation of network entities that communicate asynchronously. The model is modular and allows for incremental specification of features without redesigning existing features and models. As communication buffers may not be bounded in an asynchronous model, the underlying transition system may be infinite. Therefore an exhaustive search for feature interactions can not be used. However, with the given formal operational semantics, systems under study can be simulated and undesirable behaviors detected.

Our approach allows, so far, for the detection of two classes of interactions: triggering conflicts, and requirements violation. It is user oriented, and allows for detecting interactions between features from the user point of view. This approach should be extended to include network requirements in order to detect interactions where a feature prevents the network from behaving correctly. A classical example of interaction between the network and a user feature is the interaction between OCS and emergency numbers. The network should always process emergency numbers, while one of them could appear in a screening list.

We have considered only two important classes of feature interactions, among the future extensions of this work we will also consider other classes of interactions, such as conflicting resources (data) manipulations class of interactions.

REFERENCES

[1] J.Bergstra, W.Bouma, Models for Feature Descriptions and Interactions, in Feature Interactions in Telecommunications and Distributed Systems IV, P.Dini et al. (Eds.) IOS Press 1997, pp 31- 45.

[2] J. Blom, Formalisation of Requirements with Emphasize on Feature Interaction Detection. in Fea-

ture Interactions in Telecommunications Networks IV, P.Dini et al. (Eds.) IOS Press 1997, pp 61.

[3] E.J. Cameron, N.D. Griffeth, Y.J. Lin, M.E. Nilson, W.K. Schnure, and H. Velthuijsen,A Feature Interaction Benchmark for Intelligent Network and Beyond , pp. 1-23, Amesterdam, IOS Press, May 1994.

[4] M.Faci and L.Logrippo, Specifying Features and Analyzing their Interactions in a LOTOS Environment, Second International Workshop on Feature Interactions in Telecommunication Software Systems, IOS Press, 1994, pp. 136 - 151.

[5] D.O. Keck and P. J. Kuehn, The Feature and Service Interaction problem in Telecommunication systems: A survey, IEEE Transactions on Software Engineering, Vol. 24, No. 10, October 1998.

[6] N. Griffeth, R. Blumenthal, Jean-Charles Gregoire, Tadashi Ohta, Feature Interaction Detection Contest of the 5th International Workshop on Feature Interactions.

[7] L.Helouet, A Simulation framework for Message Sequence Charts, Proceedings of the 9th SDL Forum, June 1999.

[8] K.Kimbler, H. Velthuijsen, Feature Interaction Benchmark Presented in the Panel Session Benchmarking, FIW '95, Kyoto, Japan, 11-13 October, 1995.

[9] Graubmann.P, Rudolph.E, Grabowski.J. Tutorial on message sequence charts (msc'96). FORTE/PSTV'96, october 1996.

[10] A. Sefidcon and F. Khendek, A Pragmatic Approach for Feature Interaction Detection in Intelligent Networks, in Proceedings of the IEEE International Conference on Computer Communications and Networks (IEEE ICCCN'99), Boston, Mass., October 11-13, 1999.

[11] ITU, Norm Z.120, Message Sequence Charts