Specification and Validation of Telecommunication Services in ACP$^\tau$

Yuan Zhaorui

Abstract

This paper is concerned with the formal specification and validation of telecommunication services. Firstly, we provide a method to model the telecommunication system. Based on this result, we give the formal specification of telecommunication services and the detection of feature interactions, using the Algebra of Communication Process with Abstraction (ACP$^\tau$). Specification examples are given, first basic, about the telecommunication system and plain old telephone services, then with supplementary services, such as *call waiting*, *call forwarding busy*, *three way calling*. We show by examples how to apply our method to the detection of the feature interactions.

Keywords Telecommunication Service, Feature Interactions, ACP$^\tau$, Intelligent Network

1 Introduction

The short history but quick development of telecommunication network witnesses how they are driven and stimulated into huge changes by market competition and continuously increasing customer requirements to provide new services as quickly as possible. A series of new techniques, such as intelligent network (IN) is developed to fulfill this task. However, due to the development of software and hardware technology, the telecommunication system has evolved to a large, real-time, distributed software system. Introducing a new telecommunication service into such a complicated system also introduces the risk of interfering the behavior of existing software component or services, which may bring costly and catastrophic software faults. In order to ensure the reliability of the telecommunication system, specification and validation work is required before introducing a new telecommunication service into the network. Moreover, if a new service has negative influence on the network, strategies must be introduced to the network to solve the problem.

In the behavior analysis of telecommunication system, formal techniques have a place because they allow logical modeling, prototyping and validation of features at the design stage [20]. When engineers start to give some formal specifications of a telecommunication system, they are undoubtedly in need of an efficient model to describe the call processing procedure in the system, which can provide enough information for the specification and analysis of the system.

This paper describes a method, based on Algebra of Communication Process with Abstraction (ACP$^\tau$), for specifying and analyzing the telecommunication system. Central to the method is the call processing model based on the basic call state model (BCSM) concept from the Telecommunication sector of International Telecommunication Union (ITU-T) Capability Set-2 (CS-2) recommendation of IN. BCSM is a state transition machine representing the behavior of switching processes. BCSM is able to supports the description of call processing procedure of plain old telephone services (POTS) and 38 IN services in CS-2 [1].

Our call and service processing model is build by the parallel composition of BCSMs and other processes representing other telecommunication system entities, such as users and telecommunication services. By analyzing the parallel execution and communication behaviors of these processes with the help of computer supported tools, we arrive at a stage where it is possible to analyze the execution traces of IN calls and detect the interactions between IN services.

For specification and analysis, we use ACP$^\tau$, which has been developed since 1982 at the center for Mathematics and Computer Science, Amsterdam [5]. ACP$^\tau$ is the formalism suitable for describing distributed system of processes communicating in a synchronous way. Process Specification Formalism (PSF) is a computer readable language to specify ACP$^\tau$ processes. Moreover, PSF formally supports the data types and modularization. PSF has a supporting toolset *PSF Toolkit* to analyze the behavior of a system specified in ACP$^\tau$.

This paper is structured as follows. Section 2 presents the model of IN call and service processing. Section 3 gives a brief introduction to ACP$^\tau$ and PSF. Section 4 presents formal descriptions of the telecommunication system and services. Section 5 presents examples on how to detect feature interaction based on formal specifications and formal tools. In the appendices, we give some specification examples. Appendix A presents a PSF specification of the data used in our module. Appendix B presents a detailed PSF description of telephone users and BCSMs. Appendix C presents a PSF description of several IN services. Appendix D and Appendix E present a description of IN services states and BCSM states respectively.

Acknowledgements

This report was refined in discussion with Kees Middelburg. I would like to thank him for his comprehensive reading and important comments. I would like also to thank Jan Bergstra for his remarks on the system modeling, and Vincent van Oostrom for his nice help.

2 Modeling of IN Call Processing

2.1 Introduction to the BCSM concept in IN

The basic aim of Intelligent Network (IN) is to provide a network structure to facilitate quick service deployment. This aim is achieved by constructing a computer network upon the transport network. The function of transport network is simplified to only switching and information transmission, while the service control functions are left for the upper level computer network. At certain points in the IN call processing, the switch can suspend the call and trigger IN services residing in the computer network, and apply for IN service's control over the call processing.

ITU-T proposes two state transition machines Originating Basic Call State Model (OBCSM) and Terminating Basic Call State Model (TBCSM) to represent the switching control on the calling party and the called party respectively. Figure 1 shows both state transition machines. On the left side is the state transition machine representing the behavior at the originating side, and on the right side is the state transition machine representing the behavior of the terminating side.

BCSMs provide a high-level model of switching activities required to establish and maintain communication paths for users. As such, BCSMs identify a set of basic call and connection activities in switching and show how these activities are joined together to process a basic call and connection [1]. In particular, it provides a framework for describing basic call and connection events that can lead to the invocation of or should be reported to telecommunication service processes, for describing those points in call and connection processing at which these events are detected, and for describing those points in call and connection processing when the transfer of control can occur.

Figure 1 shows the components that have been identified to describe a BCSM, including, Detection Points (DP), represented as small rectangles; Points in Call (PIC), represented as big rectangles; BCSM transitions and events. PICs identify switching activities associated with one or more basic call/connection states of interest to telecommunication services. DPs indicate states in basic call and connection processing at which transfer of control to telecommunication service logic can occur. Telecommunication services are permitted to interact with basic call and connection control capabilities at DPs. For example, PIC Authorize_Origination_Attempt in OBCSM is a state where the originating terminal rights are being checked; DP T_Busy is a state where the

Figure 1: Basic Call State Model

terminating party busy event is encountered in terminating call processing. It is possible for IN services to control the state transitions at DP.

BCSM transitions, represented as solid lines in Figure 1, are the transitions in Plain Old Telephone Services (POTS), which indicate the normal flow of basic call/connection processing from one PIC to another. In addition to BCSM transitions, there are extended transitions, represented as dotted lines in Figure 1, which are caused by IN services.

In Section 2.2 and Section 2.3, we show how to model IN call and service processing based on the BCSM concept. We start from POTS, the elementary service in the telecommunication system.

In the IN structure, telecommunication services provide additional functions over the basic phone service in the sense that new telecommunication services are introduced to the network without modifying the switching software, but by adding software for the service concerned in the computer network. IN services differ from each other by defining different communication behavior between the computer network and the communication network, thus causes different state transition procedure in the BCSM.

2.2 Modeling of POTS

POTS is elementary in that it only provides the voice connection between two terminal users. Supplementary services and multi-party calls are all built upon the two-party call provided by POTS. In this section, we illustrate how to model POTS.

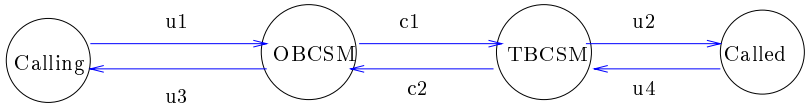

Figure 2: Modeling of a POTS call

Any POTS call is composed of 4 elements, *the calling party*, *the called party* and two switching

processes in charge of their behavior (these processes may be in the same physical entity, i.e. the same switch). Figure 2 is the modeling of a simple two party call. The entities calling and called are processes on behalf of terminal users, which are capable of sending user signals to the network, such as offhook, onhook, dialing; or receiving signals from the network, for example, *dial-tone*. These processes execute in parallel and communicate with each other to establish the voice connection between the calling party and the called party. Elements *u1* up to *u4* are communication channels between users and BCSMs. OBCSM and TBCSM synchronize with each other by communicating on Channel *c1* and *c2*.

OBCSM starts call processing when the calling party offhooks. It takes some internal actions to authorize the terminal rights to initiate the call, and to synchronize with the calling party again on the signal *dial-tone*. After the called party number is collected, OBCSM selects the route and initiates the terminal call processing on the TBCSM side. TBCSM is responsible for authorizing and ringing the called party. If the called party answers the call, the voice connection between the two parties is established and the BCSMs wait for a connection release event. BCSMs decide their state transitions based on the messages received from other entities. There are two kinds of communication between these parallel processes: communications between the users and the switching network, and communications between the switching processes, i.e. OBCSM and TBCSM.

There are several points in our model worth explanation:

1. In the modeling of POTS, we use the same process to describe the calling party and the called party,[1] which are distinguished by their communication behaviors with other processes. Actually, the deployment of multi-party services such as *three way calling* (TWC) already blur the distinction between the calling and the called. A user can be explicitly expected to be the calling or the called in a two-party call, but in multi-party calls, this situation is changed. For example, a *TWC* service subscriber can be the called party in an existing call, but his/her role changes to the calling party when the subscriber put the current call on hold and initiate a call to a new party using the functionality of the TWC service.

   If we insist on using different formal descriptions for the calling and the called party in this situation, the modeling of users will be difficult and confusing. A flexible but effective choice is to use the same process description to specify them. It is left for the communication behaviors to decide whether a user is a calling party or a called one in the current call processing context.

2. There is no direct communication between OBCSM and the called party, or between TBCSM and the calling party.

3. We use the same model for the inter-switch call and the intra-switch call. However in CS-2, the call processing model for inter-switch call is as described in Figure 3, which is derived directly from the physical implementation. In inter-switch calls, OBCSM can not communicate directly with TBCSM in another switch, so OBCSM$'$ and TBCSM$'$ are implemented respectively in the switch at the calling side and the one at the called side, in order to synchronize the OBCSM and the TBCSM. OBCSM$'$ and TBCSM$'$ are mirror processes of OBCSM and TBCSM in the sense that if a signal sent by OBCSM is received in TBCSM$'$ in certain state, the signal is received in TBCSM in the same state, and the signal is processed in the same way. If the signal causes state transitions in TBCSM and TBCSM$'$, the state transitions are the same. This is the same case for signals sent by TBCSM. Since the aim of our modeling is to analyze the service behavior and what we care about is the state transitions in OBCSM, TBCSM, as well as their communications with the environment and their influence on each other, we simplify OBCSM$'$, TBCSM$'$ and the communication channels between them to communication channels between OBCSM and TBCSM, and in this way, we use the same model for intra-switch calls and inter-switch calls.

[1] See Section 4.1 for the ACP$^\tau$ description of users.

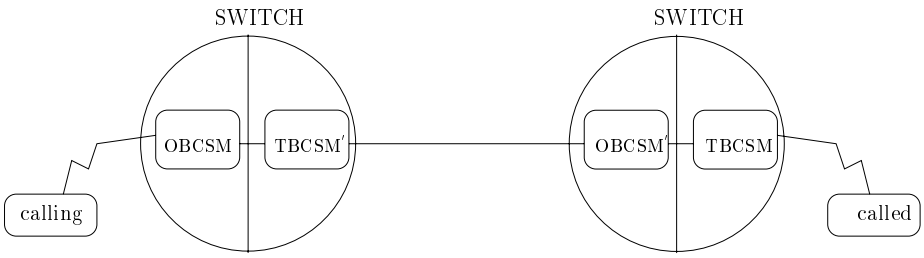

Figure 3: BCSMs in a two-party inter-switch call

In Table 1, we describe the set of user-network signals and the BCSM-BCSM signals.

We write U for the set of user-network signals, V for the set of BCSM-BCSM signals, *ADDR* for the set of terminal addresses, *ID* for the set of identifiers for the two party call. We have:

U = { *offhook*(*x*), onhook(x), flashhook(x), dial(x,y),
noanswer(x), userbusy(x), useridle(x), dial_tone(x),
alert_user(x,y), special_tone(x), stop_special_tone(x), back_ring(x,y),
CWtone(x), busy_tone(x,y), stop_back_ring(x,y), stop_alert_user(x,y),
stop_CWtone(x), stop_dial_tone(x)
| x, y ∈ *ADDR* }

V = { initiate-tb(id, y), calling-disconn(id), calling-abandon(id),
terminating-busy(id), call-acceptance(id), called-rejected(id),
called-noanswer(id), called-suspend(id), called-reanswer(id)
| id ∈ *ID*, y ∈ *ADDR* }

The modeling of a two party call is fundamental to IN service specifications and further research on feature interactions. Complicated IN calls, such as multi-party calls, conference calls can be decomposed into a number of two party calls. There are no direct communication between these two party calls, but these calls can be controlled together by one or more IN services. Similar decomposition work can be found in [17].

By decomposing complicated IN calls into a number of two party calls, we have a chance to look deep into how IN services interwork with each other, and how improper interworking results in feature interactions.[2]

2.3 Modeling of Telecommunication System with IN service

2.3.1 General Introduction

Supplementary telecommunication services, such as *call waiting* (CW), *call forwarding on busy* (CFB), *three way calling* (TWC), etc. provide additional functions to subscribers by modifying the basic call processing and connection control procedure in the basic switching element. The IN structure enables the service creation and deployment independent of the existing communication network. Services are defined by introducing additional software into the upper level computer network. The notion that service control and call control are provided separately in IN motivates us to describe a type of IN service with a parameterized process. The parameter value will be assigned when the service is activated. The interworking between the IN service and the communication network is modeled by the communications between the processes concerned. More specifically, when an IN call is initiated, at certain DP points, BCSMs can suspend call processing and send messages to IN services to report call processing events. After receiving the messages, IN services may send messages back to the BCSMs concerned, after which the BCSMs can decide the state transitions based on the messages received.

[2]See Section 5.

Signaling Set	Signaling Direction	Name	Comments
Between User and Network	User to Network	offhook	The user lifts the handset.
		onhook	The user hangs up.
		flashhook	–
		noanswer	Called party no answer
		userbusy	Called party busy
		useridle	Terminal is in the idle state
		dial	–
	Network to User	dial_tone	Dialing tone
		stop_dial_tone	To stop dialing tone
		special_tone	Alerting tone used by some supplementary services
		stop_special_tone	To stop special tone
		alert_user	Call alerting on the called party
		stop_alert_user	To stop call alerting on the called party
		busy_tone	Informing the calling party that the called is busy
		back_ring	Call alerting on the calling party
		stop_back_ring	To stop call alerting on the calling party
		CWtone	Call waiting tone
		stop_CWtone	To stop Call waiting tone
Between OBCSM and TBCSM	OBCSM to TBCSM	initiate_tb	To initiate TBCSM for call processing
		calling_disconn	To indicate calling party disconnection
		calling_abandon	To indicate calling party abandon
	TBCSM to OBCSM	terminating-busy	Called party busy detected
		call-acceptance	Call accepted by the called party
		called-rejected	Call refused by the network
		called-noanswer	Called party no answer detected
		called-suspend	Called party suspending detected
		called-reanswer	Called party re-answering detected

Table 1: the Set of User-Network signals and BCSM-BCSM Signals

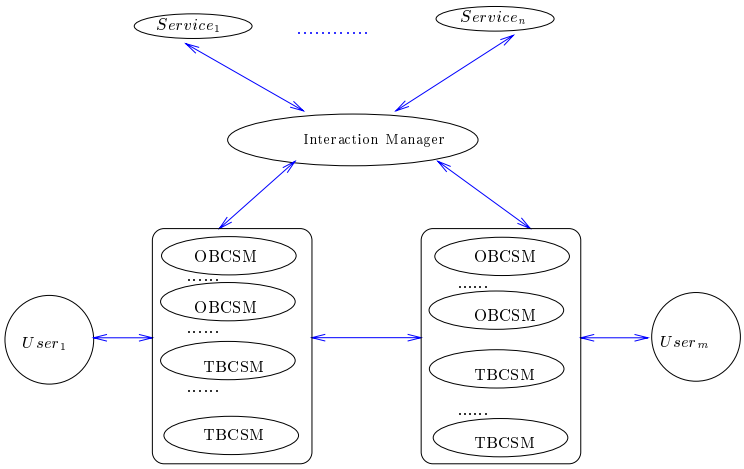

Figure 4: Modeling of IN call processing

Figure 4 is the model of a telecommunication system with m telephone subscribers and n IN service instances. Since the introduction of IN services is independent of the communication network underneath, which does not change the existing communication protocols between users and BCSMs, as well as between different BCSMs, the entities BCSMs, users and the communication between them are modeled in the same way as POTS. At any specific point of call processing, one user can only communicate with one BCSM. However, due to the existence of multi-party calls and IN services, during the course of call processing, one user may communicate with other BCSMs; the exact number of which depends on the number of calls in the system (possibly IN-calls or POTS).

The interaction manager (IM) is a managing process over communications between a number of IN service processes and BCSMs. When it comes to the dynamic solution to the feature interaction problem, [3] the IM plays an important role. The detailed function of the IM is beyond the scope of this paper. In the case of services validation and feature interaction detection, we simply assume that, for messages from IN services to BCSMs, IM acts as a buffer; for messages from BCSMs to IN services, IM acts as a broadcaster. For simplicity, in Section 4 and Section 5, we omit the process IM.

Another assumption in our model is that, if a BCSM detects a call processing event at a DP, and the reporting of the event is required by more than one telecommunication service, the DP event will be reported to all those services at the same time. In feature interaction detection, this is a useful assumption when we need to analyze how two service instances compete for one signal.

Another issue is how to model the communications between IN services and BCSMs. Such communications are based on Intelligent Network Application Protocol (INAP), the protocol defined in Q.1228 for the communication between different IN entities [1]. Basically, each INAP message is composed of an *operation_name* and an *operation_parameter*.

As a network protocol, the INAP defined in Q.1228 is very complicated. A large amount of messages, as well as some data in each message, are *call control irrelevant* (for example, *Activate-ServiceFiltering*). They are defined for providing database operation, network control and other management functions such as data security, service filtering, etc.

In our case, since we emphasize on the IN service's influence on the call processing model (represented as the BCSMs) for the purpose of service validation, *call control irrelevant* messages/parameters are ignored for they have no influence on the BCSM state transitions. By the word *"ignore"*, we mean that the action of sending or receiving such messages will not occur in the IN service's specification.

Now we can concentrate on the sub-set of INAP messages which influence the BCSM state transitions, i.e. the INAP messages send from or consumed by BCSMs. Even though, the large amount of parameters of each INAP message make it impossible to carry out any specification and validation. If we have a thorough look at the parameters which are *call processing relevant*, they can be categorized as:

1. Parameters to identify the *calling party*, such as *callingPartyNumber*, *callingPartySubaddress*, etc.
2. Parameters to identify the *called party*, such as *calledPartyNumber*, *originalCalledPartyID*, etc.
3. Parameters to identify the call, such as *callid*.
4. Parameters for billing and charging, such as *chargeNumber*, etc.

We simplify the INAP message parameter set by assigning one parameter for each type in the categorization. Moreover, since billing and charging behavior is not of our concern now, the relevant parameters are ignored. Each BCSM also carries these three parameters, *callid*, *callingPartyID*, *calledPartyID* to identify it.

[3] See Section 5.1.2.

Currently, we have successfully applied our model in specifying and validating some typical IN services and detecting some benchmark feature interactions.[4] Actually, the feature interactions we have detected cover the majority interactions ever mentioned in papers available. Furthermore, most of the feature interaction detection method available now use these three parameters or even less.

Considering this, we believe our method *"safely"* decrease the data complexity we are dealing with. However, in case that the screening of some parameters may potentially introduce feature interaction that does not really exist, a solution would be carefully adding the necessary parameter into both the BCSMs and the INAP messages, but the basic state transitions in BCSMs remain unchanged. Considering this, it will be very easy to expand and enrich our model if necessary.

From now on, we write I for the set of INAP signals, listed in Table 2 are the descriptions of the INAP message set we use. Let's call it sub-INAP.

2.3.2 An Example on Service Modeling

At the IN distributed function plane, telecommunication services can be viewed as information flows between different network entities. For service specification and validation, we only care about those information flows that influence the switch call processing procedure, i.e. the INAP messages from BCSM to IN service, which report the call processing event, and the INAP messages from IN services to BCSMs, which control the state transitions in BCSM. Based on this notion, each IN service is modeled by a process with several states, and in each state, the IN service can send and receive some INAP messages (i.e. messages from the set *sub-INAP*). Such modeling of IN services is consistent with the idea of separating call control from service control in IN. Call control is performed in switching processes, represented as BCSMs.

The key point of modeling an IN service is to describe how they control the state transitions in BCSMs by communicating with them. To achieve this goal, we need to decide *which* BCSMs the IN service communicates with, as well as *how* they communicate, i.e. at which DP the communication happens, the INAP message name, and parameters.

Take the modeling of the *call waiting* service as an example. A CW subscriber y will hear a call waiting tone when an incoming call arrives at him and he is in connection with someone else. After that, he can choose to answer the new incoming call or to ignore the call waiting tone. More specifically, there are two two-party calls involved in any *CW* call scenario, the existing two-party call and the incoming one, with each behavior represented as a pair of OBCSM and TBCSM. The two calls interwork with each other, by sharing a call party that subscribes to the CW service. The CW service subscriber can be the calling party (Case 1) or the called party (Case 2) in the existing call.

Based on this analysis, we give the modeling of both cases in Figure 5 and Figure 6. User y is the CW service subscriber. CW_y is the process describing the behavior of the CW service subscribed by the user y. Since these two scenarios are similar when analyzing the behavior of the CW service, we use the model in Figure 5 as an example.

The PSF specification of the CW service is given in Appendix B. The CW service has four states:

1. *call waiting null*: denoted as *CW-null* in the PSF specification; upon receiving the INAP message to report a *called party busy* event (*r(inap-TBusy(req,callxz,x))*), the service transfers to the *call waiting service active* state.

2. *call waiting service active*: denoted as *CW-active* in the PSF specification; a state where a terminating busy event has been reported by the TBCSM and the CW service is activated.

[4] See Section 5.2, 5.3 and 5.4.

[5] A parameter marked with the character o is optional.

Signaling Direction	Name in Q.1228	Name in our Model	Parameters
From Services to BCSMs	AnalyseInformation	inap-ai	callid, calledPartynumber(o)[5]
	ColletInformation	inap-ci	callid
	Connect	inap-conn	callid, callingPartyNumber(o), calledPartyNumber(o)
	Continue	inap-cont	callid
	ContinueWithArg	inap-cwa	callid, callingPartyNumber(o), calledPartyNumber(o)
	DisconnectLeg	inap-dl	callid, calledPartyNumber
	InitialCallAttempt	inap-ica	callid, callingPartyNumber
	OriginateCall	inap-oc	callid, callingPartyNumber
	ReleaseCall	inap-rc	callid, callingPartyNumber/calledPartyNumber
	Reconnect	inap-rec	callid
	SelectRoute	inap-sr	callid, callingPartyNumber(o), calledPartyNumber(o)
	MergeCallSegment	inap-mgc	callid, callingPartyNumber/calledPartyNumber
	MoveCallSegment	inap-mvc	callid
	MoveLeg	inap-ml	callid, callingPartyNumber/calledPartyNumber
	SelectFacility	inap-sf	callid
	SplitLeg	inap-sl	callid, callingPartyNumber/calledPartyNumber
From BCSMs to Services	AnalysedInformation	inap-Analysed	req/ind, callid, callingPartyNumber
	OriginationAttempt	inap-OAttempt	req/ind, callid, callingPartyNumber
	OriginationAttempt-Authorized	inap-OAuthorized	req/ind, callid, callingPartyNumber
	CollectedInformation	inap-Collected	req/ind, callid, callingPartyNumber
	OTermSeized	inap-OSeized	req/ind, callid, callingPartyNumber
	OAnswer	inap-OAnswer	req/ind, callid, callingPartyNumber
	OReAnswer	inap-OReAnswer	req/ind, callid, callingPartyNumber
	OSuspended	inap-OSuspend	req/ind, callid, callingPartyNumber
	ODisconnect	inap-ODisc	req/ind, callid, callingPartyNumber
	OMidCall	inap-OMid	req/ind, callid, callingPartyNumber
	OCalledPartyBusy	inap-OBusy	req/ind, callid, callingPartyNumber
	ONoAnswer	inap-ONoAnswer	req/ind, callid, callingPartyNumber
	OAbandon	inap-OAbandon	req/ind, callid, callingPartyNumber
	RouteSelectFailure	inap-RouteFail	req/ind, callid, callingPartyNumber
	–	inap-ONull	req/ind, callid, callingPartyNumber
	TerminationAttempt	inap-TAttempt	req/ind, callid, calledPartyNumber
	TerminationAttempt-Authorized	inap-TAuthorized	req/ind, callid, calledPartyNumber
	FacilitySelectAvailable	inap-FacilAvail	req/ind, callid, calledPartyNumber
	TMidCall	inap-TMid	req/ind, callid, calledPartyNumber
	TNoAnswer	inap-TNoAnswer	req/ind, callid, calledPartyNumber
	TBusy	inap-TBusy	req/ind, callid, calledPartyNumber
	TAbandon	inap-TAbandon	req/ind, callid, calledPartyNumber
	TDisconnect	inap-TDisc	req/ind, callid, calledPartyNumber
	CallAccepted	inap-CAccept	req/ind, callid, calledPartyNumber
	TAnswer	inap-TAnswer	req/ind, callid, calledPartyNumber
	TReAnswer	inap-TReAnswer	req/ind, callid, calledPartyNumber
	TSuspended	inap-TSuspend	req/ind, callid, calledPartyNumber
	–	inap-TNull	req/ind, callid, calledPartyNumber

Table 2: The Refined INAP Message Set in Our Model

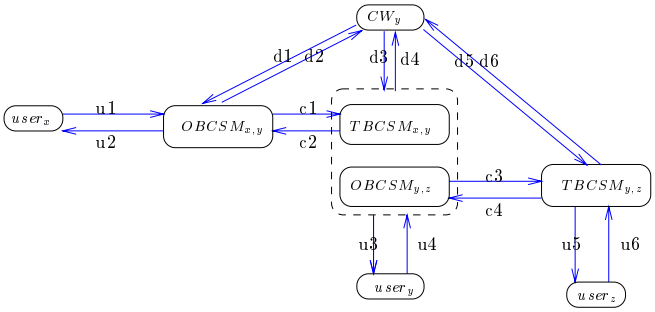

Figure 5: Modeling of a CW Call: Case 1

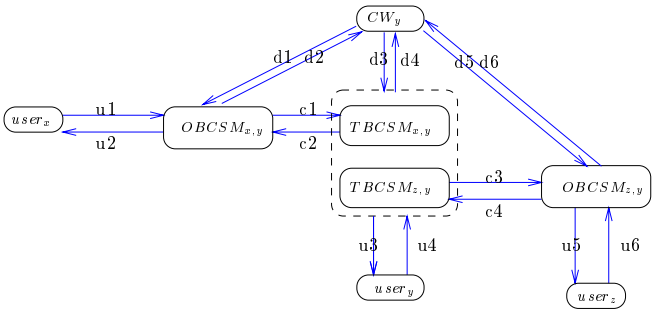

Figure 6: Modeling of a CW Call: Case 2

3. *call waiting*: denoted as *CW-cw* in the PSF specification; a state where a call waiting tone has been send to the CW subscriber. The CW service can deal with the following call events in this state: *flashhook* from the CW subscriber; *onhook* from any call party.

4. *call on hold*: denoted as *CW-ch* in the PSF specification; a state where a call is put on hold by the CW service instance. The CW service can deal with the following call event in this state: *flashhook* from the CW subscriber; *onhook* from any party.

The *CW* service communicates with $OBCSM_{x,y}$, $TBCSM_{x,y}$, $OBCSM_{y,z}$ and $TBCSM_{y,z}$. The INAP messages send by these BCSMs are sent as an *indication*[6] or as a *request*[7], depending on the service requirement. If a BCSM sends an INAP message as an *indication*, it will continue call processing without waiting for the response messages from IN services to decide the subsequent BCSM transitions. If a BCSM sends an INAP message as an *request*, it will suspend call processing and wait for the response messages from IN services.

For example, if in the *call on hold* state, the user z onhooks when he/she is on hold, an INAP message *inap-TDisc(req,callxz,z)* or *inap-TSuspend(req,callxz,z)* will be sent to the CW service by $TBCSM_{y,z}$.[8]

3 Introduction to ACP$^\tau$ and PSF

This section serves as a brief and basic introduction to ACP$^\tau$ and PSF, which is the base of tools to help writing and analyzing specifications in ACP$^\tau$. For more thorough discovery of ACP$^\tau$ and

[6] The message form is *message-name(ind, parameter1, parameter2)* ; *ind* is the constant.
[7] The message form is *message-name(req, parameter1, parameter2)* ; *req* is the constant.
[8] The exact message depends on the exact phase of call processing.

PSF, we refer to [6, 9, 10].

3.1 ACP$^\tau$

ACP$^\tau$ (Algebra of Communicating Processes with abstraction) is a process algebra approach for the study of concurrent processes, developed by J.Bergstra and J.W.Klop in the early 1980s. ACP$^\tau$ is suitable for both specification and verification of communicating processes.

Processes in ACP$^\tau$ are considered as capable of performing *atomic actions* from the alphabet $A = \{ a, b, c, \cdots \}$. An atomic action is pointwise, i.e. the execution of an atomic action takes no time. Atomic actions are considered not be be divisible into the smaller parts for further investigations. It depends on the need of the system abstraction that which actions one wants to see as atomic.

In addition to ordinary atomic actions, there are two constants to express special process behavior:

δ indicates that no activity is possible, i.e. a *deadlock* occurs.

τ represents an internal action that is of no interest to the environment.

A partial communication function $\gamma : A \times A \to A$ is defined in ACP$^\tau$ to describe how actions in A can synchronize in order to obtain interactions between processes.

ACP$^\tau$ provides operators to describe the combination of processes or atomic actions, which are listed below. We write $V = \{ x, y, z, \cdots \}$ for the set of variables with each of them standing for a process, $\mathrm{A}_{\delta\tau}$ for $A \cup \{\delta, \tau\}$.

$x + y$ represents the process, first making a choice between processes x and y, and then proceeding with the chosen course of action.

$x \cdot y$ represents the process who first performs actions in x, and if x terminates successfully, $x \cdot y$ proceeds as y. Deadlocks or infinite internal loops in x may cause $x \cdot y$ can not perform actions in y.

$x \| y$ represents the process who first chooses whether to do an action in x or in y, or the communication between x and y, i.e., $x \mid y$, and proceeds as the parallel composition of the remainders of x and y. In other words, the actions of x and y are interleaved. [5]

$x \lfloor\!\lfloor y$ is the process $x \| y$, but under the assumption that x starts first.

$x | y$ represents the process who starts with the communication between x and y, and then acts as the parallel composition of the remainder of x and y.

$\partial_H(x)$ represents the process x without the possibility of performing actions from H. $\partial_H(x)$ maps the atomic actions in H to deadlocks. In most cases, actions in set H are message receive and send actions of a process. By mapping them into deadlocks, communications is forced.

$\tau_I(x)$ represents the process x, with all actions in I considered as not important any more. $\tau_I(x)$ maps the atomic actions in I to internal actions.

Table 3 summarize the signature of ACP$^\tau$. We have the convention that $\cdot$ binds stronger than

constants	a	for any atomic action $a \in A$
	δ	deadlock
	τ	silent action
unary operators	∂_H	encapsulation, for any $H \subseteq A$
	τ_I	abstraction, for any $I \subseteq A$
binary operators	$+$	alternative composition (sum)
	$\cdot$	sequential composition (product)
	$\|$	parallel composition (merge)
	$\lfloor\!\lfloor$	left merge
	$\mid$	communication merge (bar)

Table 3: ACP$^\tau$ Signatures

$\|$ and $\lfloor\!\lfloor$, which in turn bind stronger than $+$. For simplity, we write ab for $a{\cdot}b$.

3.2 Axioms

ACP$^\tau$ axioms reflect the basic properties of the constants and operators in the signature. Table 4 lists those axioms which are used later for the telecommunication service specification and validation, including axioms on *communication*, *abstraction*, etc. For the same reason, axioms on ACP$^\tau$ operator *projection* are not listed here. For the whole set of axioms, see [5]. We have $a, b, c \in A_{\delta\tau}$, $H, I \subseteq A$ and x, y, z are arbitrary processes.

$x + y = y + x$	A1	$x\tau = x$	B1
$x + (y + z) = (x + y) + x$	A2	$x(\tau(y + z) + y) = x(y + z)$	B2
$x + x = x$	A3		
$(x + y)z = xz + yz$	A4		
$(xy)z = z(yz)$	A5		
$x + \delta = x$	A6	$a\|b = \gamma(a, b), if \gamma(a, b) \downarrow$	CF1
$\delta x = x$	A7	$a\|b = \delta, otherwise$	CF2
$x\|y = x\lfloor\!\lfloor y + y\lfloor\!\lfloor x + x\|y$	CM1	$(ax)\|b = (a\|b)x$	CM5
$a\lfloor\!\lfloor x = ax$	CM2	$a\|(bx) = (a\|b)x$	CM6
$ax\lfloor\!\lfloor y = a(x\|y)$	CM3	$(ax)\|(by) = (a\|b)(x\|y)$	CM7
$(x + y)\lfloor\!\lfloor z = x\lfloor\!\lfloor z + y\lfloor\!\lfloor z$	CM4	$(x + y)\|z = x\|z + y\|z$	CM8
		$x\|(y + z) = x\|y + x\|z$	CM9
$\partial_H(a) = a, if, a \notin H$	D1	$\tau_I(a) = a, if, a \notin I$	TI1
$\partial_H(a) = \delta, if, a \in H$	D2	$\tau_I(a) = \tau, if, a \in I$	TI2
$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	TI3
$\partial_H(xy) = \partial_H(x)\partial_H(y)$	D4	$\tau_I(xy) = \tau_I(x)\tau_I(y)$	TI4

Table 4: ACP$^\tau$ Axioms

Axioms A1-5 are the basic axioms for process algebra. Note that one distributive law, $z(x + y) = zx + zy$, does not hold in ACP$^\tau$. Otherwise, for example, we will have $ab = a(b + \delta) = ab + a\delta$. This mean that a process with deadlock is possibly equal to one without, and that conflicts with the intention to model also deadlock behavior of a process [5].

The axiom CF1 expresses that the operator $|$ is an extension of γ. Since δ is not in A, γ is not defined if one of its arguments is δ, and so the axiom CF2 implies that $a|\delta = \delta|a = \delta, \forall a \in A$.

The axiom B2 corresponds exactly to the situation in Figure 7. B1 is the special case of B2 when x and y are absent.

Axioms CM1-4 presents the behavior of the merge and left-merge as described before.

Axioms CM5-9 define the extension of the communication function to arbitrary processes, using the inductive definition of basic terms.

Axioms D1-4 formally describe the properties of the encapsulation operator. For any set I of atomic actions, the abstraction operator satisfies TI1-4. We always have that $\delta \notin I$ (since $\delta \notin A$ and $I \subseteq A$), so δ can never be renamed into τ, and we have $\tau_I(\delta) = \delta$.

3.3 A Simple Example

We try to give more concrete ideas about ACP$^\tau$, by illustrating how to formalize a two-bit buffer.

Figure 8 is the figurative description of a two-bit buffer B^{13}, which is composed of two one-bit buffers, B^{12} and B^{23}. We write B^{ij} for the buffer with input data from Channel i ($r_i(d)$),

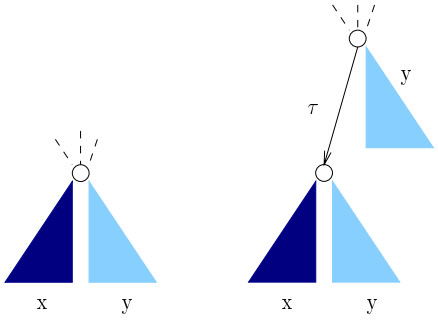

Figure 7: On B2 Law

and output data to Channel j ($s_j(d)$); $D = \{0, 1\}$ for the data set and $d \in D$. Let's define $r_2(d) \mid s_2(d) = c_2(d)$, and $c_2(d)$ is the communication of data d on Channel 2.

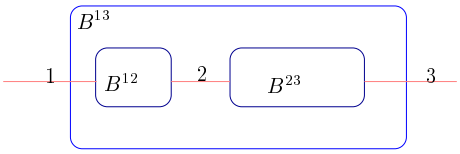

Figure 8: A Two-bit Buffer

B^{12} and B^{23} are infinite processes; each of them reads one bit from the environment, and sends it subsequently. The ACP$^\tau$ descriptions of their behaviors are:

$$B^{12} = \Sigma_{d \in D} r_1(d) s_2(d) B^{12}$$
$$B^{23} = \Sigma_{d \in D} r_2(d) s_3(d) B^{23}$$

If we map the message send and receive actions on Channel 2 to deadlocks by the encapsulation operator, the communication and message send and receive actions on Channel 2 will always happen simultaneously, which leads to a communication action on *Channel 2*. By doing so, we formalize buffer B as a transparent box, with *Channel 1*, *Channel 2* and *Channel 3* observable. Then we can formalize the two-bit buffer as

$$B_1 = \partial_{\{c2(d), s2(d) \mid d \in D\}}(B^{12} \| B^{23})$$

If we are not interested in the internal communications between B^{12} and B^{23}, then we can abstract the communication behavior on *Channel 2* to internal actions (τ) as the following:

$$B_2 = \tau_{\{c2(d) \mid d \in D\}}(B_1)$$

B_2 describes the same buffer as B_1, but now the communication between B^{12} and B^{23} is not observable, i.e. B_2 is viewed as a black box. B_2 is equal to B^{13} in the following specification:

$$B^{13} = \Sigma_{d \in D} r_1(d) \cdot B_d$$
$$B_d = s_j(d) \cdot B_2^{13} + \Sigma_{e \in D} r_1(e) \cdot s_3(d) \cdot B_e$$

The only observable behavior of B_2 for any observer from the environment is, two bits are send to the buffer, and they are buffered and sent out to the *Channel 3*. The procedure goes infinitely.

Generally speaking, a typical specification of a system of communicating processes $P_1, \ldots, P_n$ in ACP$^\tau$ is in the form of

$\partial_H(P_1 \parallel P_2 \parallel \cdots \parallel P_n)$, where n is the number of processes in the system.

By the parallel composition of communicating processes, it is possible to describe and analyze a distributed system. Prefixing the encapsulation operator says that the system $P_1, \ldots, P_n$ is to be perceived as a separate unit w.r.t. the communication actions mentioned in H; no communications between actions in H with an environment are expected or intended [5].

3.4 PSF

The process specification formalism(PSF) and its supporting toolkit *PSF Toolkit* aim at the construction of an integrated environment of computer tools for studying concurrent systems in the setting of ACP$^\tau$ [9].

3.4.1 From ACP$^\tau$ to PSF Specifications

PSF is a formal description technique for the specification of concurrent systems. It is based on process algebra and algebraic specifications. PSF supports modularization.

ACP$^\tau$ specifications and their relative parts in PSF are very similar. Operators such as $+$, $\parallel$, $\cdot$ remain unchanged in PSF. The operator $|$ is defined on atomic communicating actions in PSF. The ACP$^\tau$ operators $\partial_H(x)$ and $\tau_I(x)$ are *encaps(H,x)* and *hide(I,x)* in PSF respectively. In this sense, ACP$^\tau$ terms can be slightly changed and easily rewritten to PSF specifications.

Additionally, PSF specifications have a modular style. Any PSF specification consists of a sequence of modules each one of which is either a *data module* or a *process module*. The data modules are used to define the properties of the data types and the process modules define the behavior of the processes. A module consists of a number of sections which are listed below.

DATA MODULE	PROCESS MODULE
parameter section	parameter section
export section	export section
import section	import section
sort section	sort section
function section	atom section
variable section	process section
equation section	set section
	communication section
	variable section
	(process) definition section

Modularization is a key feature of PSF specification, and it is supported by the *export section*, *import section* and *parameter section*, which facilitates data re-usability. For example, all definitions that are listed in the *export section* of one module are visible outside the module. All *sorts*, *functions*, *atoms*, *processes* and *sets* declared but not presented in the export section are local to the module in which they are defined. By importing module A in module B, all exported objects in A become visible to B. A PSF module is reusable if it is parameterized. Parameters of a data module may only be *sorts* and *functions*, whereas parameters of the process module may be *atoms*, *processes* and *sets* as well.

Two other key features of PSF specifications are *data specification* and *process specification*.

Data specification is composed of the *sort section*, *function section* and *equation section*. *Data specification* can occur in the *data module* or the *process module*.

In the *sort section*, we define the types of data introduced. The functions are declared within the *function section* along with their *input type* (the type of the arguments) and the *output type*.

Functions without arguments are are called *constants*. In the *equation section*, relationships between data elements are defined. Equations may includes variables.

In the *process specification*, process behavior is defined. First, PSF supports the description of atomic actions, which may include variables or not. Atomic actions are combined to processes by the operators of ACP^{τ} such as *sequential composition*, *parallel composition*, *encapsulation* and *abstraction*, which are all supported in PSF.

3.4.2 PSF Toolkit

The *PSF Toolkit* is the computer tool-set for the testing and validation of PSF specifications. Basically, it provides a *compiler* to compile a PSF module to *Tool Interface Language* (TIL), a low level language easily accessible for some computer tools. The compiled code in TIL can be executed in the *PSF simulator*.

The *PSF simulator* is an interactive program that communicates with the user by several windows, to show the possible execution traces of a process. The user interface is based on the X Window system. By simulating a process the user gets better insight into its behavior. Especially *deadlocks*, *livelocks*, etc., are often detected through the use of process simulation.

The *PSF simulator* provides a series of windows for user interactions.

1. *The PSF window* displays the PSF version of the TIL specification that is currently being simulated.

2. *The choose window* offers the menu of possible next actions to the user. The user can choose which action to perform if there are more than one action. A *random once* choice is available if the user wants the system to make a random choice.

3. *The function window* contains a number of buttons that let the user issue commands to the simulator. There are:

   - *Random simulation*
     If the *random button* in the *function window* is chosen, the simulator will be set in the random mode, which means that the simulator does not show the *choose window* any longer, but chooses a random execution path until a process terminates or the *random button* is selected again.
   - *Tracing*
     - *Default tracing button*
       A default tracing mode for the PSF simulator. In this tracing mode, actions which are governed by the *encapsulation* and the *abstraction* operator are not visible.
     - *Trace button*
       After selecting this button, PSF simulator will provide a menu for the user to choose which module, or which item in a specific module to trace. This tracing mode provides the flexibility if the user wants more information about the system, not limited to the *encapsulation* and the *abstraction* operator.
   - *Breakpoints*
     If the user selects a breakpoint button, he/she can choose from a menu to set some atomic actions or processes as breakpoints. It is often used in the *random simulation mode*, if the user wants to get control over the process execution.
   - *Control buttons*
     A window contains the general control over the simulator. Control functions include displaying the internal status (e.g. process identifier) of the simulated terms in the *message window*, by selecting *process status* button, *saving* the content of the trace window to a file, *loading* a new TIL specification to replace the current one for simulation, and, *quitting* the simulator.

4. *The tracing window*

   All tracing information is written to this window.

5. *The message window*

   The PSF simulator sends message to the user via this window. Messages include the (un)successful termination of the process or the resetting of the simulation. The *process status* is also written to this window if requested by the user.

Besides all these tools, the *PSF Toolkit* also provides a proof assistant tool for interactively manipulating process expression and the proof of two processes' equivalence, and a term-rewriting tool.

4 From Modeling to Formal Specification and Validation

In the previous sections, we explain in detail how to model the telecommunication system, first basic, then with IN services. In this section, we go further by applying ACP$^\tau$ in our model, in order to analyze and validate the service behavior. Each system object such as the BCSM, the user and the IN service are formalized as individual ACP$^\tau$ processes. Operators of ACP$^\tau$, such as *parallel composition*, *encapsulation*, *abstraction*, are used to describe the interworking between processes. PSF provides tools for validating the specification in ACP$^\tau$, which we will show by examples.

4.1 Specification of Telecommunication System in ACP^τ

In this section, we give the ACP$^\tau$ description of the telecommunication system as modeled in Figure 4. First, we show how to formalize individual entities such as users, BCSMs and services, and then how to combine them into a system.

1. Equation 1 is the specification of a telephone user in ACP^τ, with subscript x to identify the user. The user process has two possible states, *idle* and *busy*. In each state, the user is able to send or receive some user-network signals. The model is intuitively consistent with what we know about the telephone in the real world. It is either in the state of the handset being picked up (*busy*), or the handset already being hooked on (*idle*).

   We write $ADDR$ for the set of terminal addresses, $s_{u_{i_x}}(sig)$ for the message send action (from $user_x$ to the BCSM) via channel u_{i_x}, $r_{u_{i_x-1}}(sig)$ for the message receive action (from the BCSM to $user_x$) via channel u_{i_x-1}, $i_x \in 2\mathbb{N} \wedge \forall x, y \in ADDR \bullet i_x = i_y \Leftrightarrow x = y$.

$$
\begin{aligned}
user_x &= idle_x + busy_x \qquad (1)\\
idle_x &= s_{u_{i_x}}(offhook(x)).busy_x\\
&+ s_{u_{i_x}}(useridle(x)).idle_x\\
&+ s_{u_{i_x}}(noanswer(x)).idle_x\\
&+ \Sigma_{y\in ADDR} r_{u_{i_x-1}}(alert\_user(y,x)).idle_x\\
&+ r_{u_{i_x-1}}(special\_tone(x)).idle_x\\
busy_x &= s_{u_{i_x}}(onhook(x)).idle_x\\
&+ s_{u_{i_x}}(flashhook(x)).busy_x\\
&+ r_{u_{i_x-1}}(dial\_tone(x)).\Sigma_{y\in ADDR} s_{u_{i_x}}(dial(x,y)).busy_x\\
&+ r_{u_{i_x-1}}(stop\_dial\_tone(x)).busy_x\\
&+ \Sigma_{y\in ADDR} r_{u_{i_x-1}}(back\_ring(x,y)).busy_x\\
&+ \Sigma_{y\in ADDR} r_{u_{i_x-1}}(stop\_back\_ring(x,y)).busy_x\\
&+ \Sigma_{y\in ADDR} r_{u_{i_x-1}}(busy\_tone(x,y)).busy_x\\
&+ \Sigma_{y\in ADDR} r_{u_{i_x-1}}(CW tone(y,x)).(s_{u_{i_x}}(flashhook(x)) + s_{u_{i_x}}(noanswer(x))).busy_x\\
&+ \Sigma_{y\in ADDR} r_{u_{i_x-1}}(stop\_CW tone(y,x)).busy_x\\
&+ \Sigma_{y\in ADDR} r_{u_{i_x-1}}(stop\_alert\_user(y,x)).busy_x\\
&+ s_{u_{i_x}}(userbusy(x)).busy_x
\end{aligned}
$$

2. BCSM is the basic state machine to model the network behavior. In the formalization of BCSM, on the one hand, we should include necessary information to facilitate the service specification and validation; on the other hand, useless details should be excluded in order not to complicate the problem.

   Appendix A gives the PSF specifications of OBCSM and TBCSM. Each PIC or each DP in Figure 1 is mapped to a process state. Each PIC can

   (a) Communicate with the *user* process;

   (b) Communicate with the TBCSM of the same call (identified by the variable *callid*);

   (c) Perform some internal actions, which are formalized as atomic actions in ACP$^\tau$. Moreover, such actions are abstracted to silent actions for we are not interested in the internal actions of the BCSM in the case of service validation.[9]

   Each DP can

   (a) Suspend call processing and send an INAP message of type *request* to IN services,[10] waiting for the responding messages from the IN services, and decide the subsequent state transitions;

   (b) Send an INAP message of type *indication*,[11] to notify IN services of some call processing event, and transfer to the subsequent state;

   (c) Transfer to the subsequent state without any communications.

   Again, since our work aims at the validation of telecommunication services, we introduce the following assumption to avoid unnecessary details in the BCSM concept originated from CS-2:

[9] See the sets *swto-skip* and *swtt-skip* in the PSF description of BCSMs in Appendix B.
[10] In PSF, such message is described as *message-name(*req, *parameter1, paramter2, ...)*.
[11] In PSF, such message is described as *message-name(*ind,*parameter1, paramter2, ...)*.

(a) The switches are fast enough to handle all telephone calls at once; there is no network busy conditions; the user is cooperative and he/she always finishes to dial the digits in time, and so on. Such assumptions ensures that actions such as *origination-denied, collect-timeout* as well as other actions in the set *swto-lock* of OBCSM description and *swtt-lock* of TBCSM description will never occur.[12] In ACP$^\tau$, we map such actions to deadlock.

(b) In CS-2, there is no communication defined between IN services and BCSMs at any PIC. Thus, it is not convenient to formalize services such as *Call Completion to Busy Subscriber* (CCBS) and *Automatic Recall* (AR). We enhance our BCSM model by enabling a BCSM to communicate with an IN service at *O_Null* and *T_Null*. The IN service can send INAP messages such as *InitialCallAttempt, OriginateCall* to OBCSM.[13]

We also introduce two other messages: *inap_ONull* sent by OBCSM at *ONull* PIC and *inap_TNull* sent by TBCSM at *TNull* PIC respectively. The two messages do not belong to the INAP message set in CS-2, but the introduction of them facilitates the description of CCBS and AR, which deal with the event that terminals become idle.

3. General principles of formalizing telecommunication services are given in Section 4.3.

4. Individual processes are combined into the system described in Equation 2 by parallel composition and encapsulation.

$$\begin{aligned} sys \quad = \partial_H(\quad & service_1 \parallel \cdots \parallel service_n \parallel \\ & user_1 \parallel \cdots \parallel user_m \parallel \\ & OBCSM_1 \parallel \cdots \parallel TBCSM_1 \parallel \\ & \cdots\cdots \\ & OBCSM_s \parallel \cdots \parallel TBCSM_s) \\ & n, m, s \in \mathbb{N} \end{aligned} \tag{2}$$

H is the set of signal receive and send actions on each channel in the system; s is the number of two party calls in the system. The encapsulation operator leads message send and receive actions to the communication between the processes concerned (if possible, i.e. the processes are free of deadlock caused by communications). The only observable actions of the system *sys* are the communication actions on each channel.

4.2 Specification of POTS

Equation 3 is the ACP^τ description of a POTS call scenario, based on the call processing model in Figure 2, with $user_x$ as the calling party, $user_y$ as the called party. Listed below are the symbols we use in Section 4 and Section 5:

$s_k(sig)$:	sending a signal via channel k, $k \in K$
$r_k(sig)$:	receiving a signal via channel k, $k \in K$
$user_x$:	the ACP^τ description of a telephone subscriber x. Similarly, we have $user_y$, $user_z$, etc.
$OBCSM_{x,y}$:	the ACP^τ description of the OBCSM of a two party call from x to y. Similarly, we have $OBCSM_{y,z}$, $OBCSM_{x,z}$, etc.
$TBCSM_{x,y}$:	the ACP^τ description of the TBCSM of a two party call from x to y. Similarly, we have $TBCSM_{y,z}$, $TBCSM_{x,z}$, etc.
$POTS_{x,y}$:	the ACP^τ description of a POTS call from x to y.

[12] See Appendix B for the PSF specifications of BCSMs.
[13] See Q.1228 for the description of INAP messages.

From now on, we write $u_1, \cdots, u_m$, $c_1, \cdots, c_n$, $d_1, \cdots, d_s$, $m, n, s \in \mathbb{N}$ for the communication channels between the user and the BCSM, the OBCSM and the TBCSM, the BCSM and the service respectively. Generally we have

$$K = \{u_1, \cdots, u_m, c_1, \cdots, c_n, d_1, \cdots, d_s\}.$$

In the case POTS, since no communication between the BCSM and the service is involved, we have

$$K = \{u_1, \cdots, u_m, c_1, \cdots, c_n\}.$$

$$\begin{aligned} POTS_{x,y} &= \partial_{H_{POT}}(user_x \parallel user_y \parallel OBCSM_{x,y} \parallel TBCSM_{x,y}) \\ H_{POT} &= \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot \cdot 4]\} \\ &+ \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot \cdot 2]\} \end{aligned} \tag{3}$$

The only observable actions of $POTS_{x,y}$ from the environment is the communication actions on each channel in the system.

Specifications in Equation 3 can be rewritten in PSF and simulated in the PSF Toolkit, which subsequently gives the following nice results:

1. *Simulation of POTS*

   With the help of *PSF Toolkit*, we have exhaustively simulated each branch in the service description of POTS at the user level, as described in Figure 9 [25]. This result can be used to validate POTS; moreover it confirms our belief in the effectiveness of our specification and modeling.

   Figure 10 gives an example, expressed in the form of sequence charts: the calling party off-hooks, dials the called party number; the call is active after the called party off-hooks. After a period of conversation, the calling party on-hooks first, then the called party will hear a disconnecting tone, and onhook subsequently. We presented the result in the form of sequence charts. For simplicity, we only give PICs in describing the traces of BCSMs, omitting DPs except for T_Disconnect.

2. *Monitoring the system states*

   PSF provides functions such as tracing, breakpoints setting to aid the process analysis. We can choose to monitor the communication actions, state transitions, or any other actions of any process in the POTS system that we are interested in.

4.3 Specification of Telecommunication Services

Equation 4 is the ACP^τ description of the CW call scenario modeled in Figure 5. $CWcall_{y,z,x}$ and CW_y is the ACP^τ description of the system and the CW service respectively.

$$\begin{aligned} CWcall_{y,z,x} &= \partial_{H_{CW}}(\quad user_x \parallel user_y \parallel user_z \parallel \\ &\qquad OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\ &\qquad OBCSM_{y,z} \parallel TBCSM_{y,z} \parallel \\ &\qquad CW_y) \\ H_{CW} &= \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot \cdot 6]\} \\ &+ \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot \cdot 4]\} \\ &+ \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot \cdot 6]\} \end{aligned} \tag{4}$$

Variables:
Busy A: true between an Off-hook A event and the next On-hook A event; between a Start Ring A B event and the next Stop Ringing A B event, if no Off-hook A intervenes; or between a Start Ringing A B event and the next On-hook A.

Ring A B: true between a Start AudibleRinging A B event immediately following a Dail A B event and the next Stop Ringing A B event.

AudibleRinging A B : true between a Start AudibleRinging A B event immediately following a Dial A B event and the next Stop AudibleRinging A B event.

All of the POTS event sequences start and end with Busy A= False (Idle A= True).

The values of Busy B in the diagram have been given in the preceding rules, but for illustration we show the value changes, in nodes 4, 9, 10 and 14.

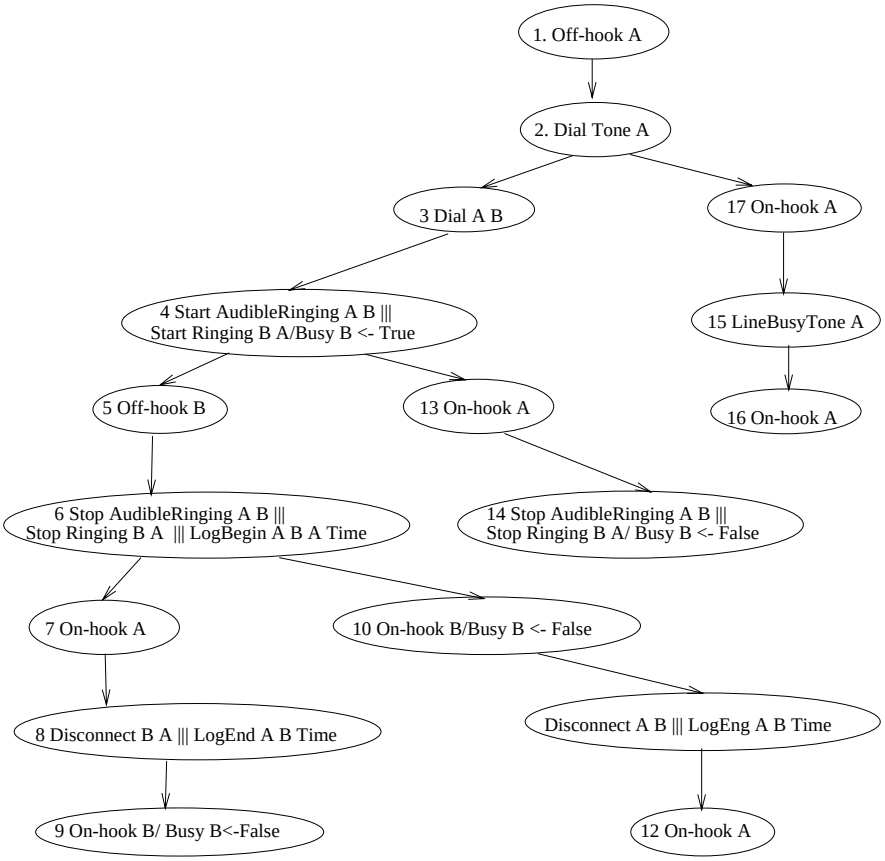

Figure 9: POTS description at the User Level

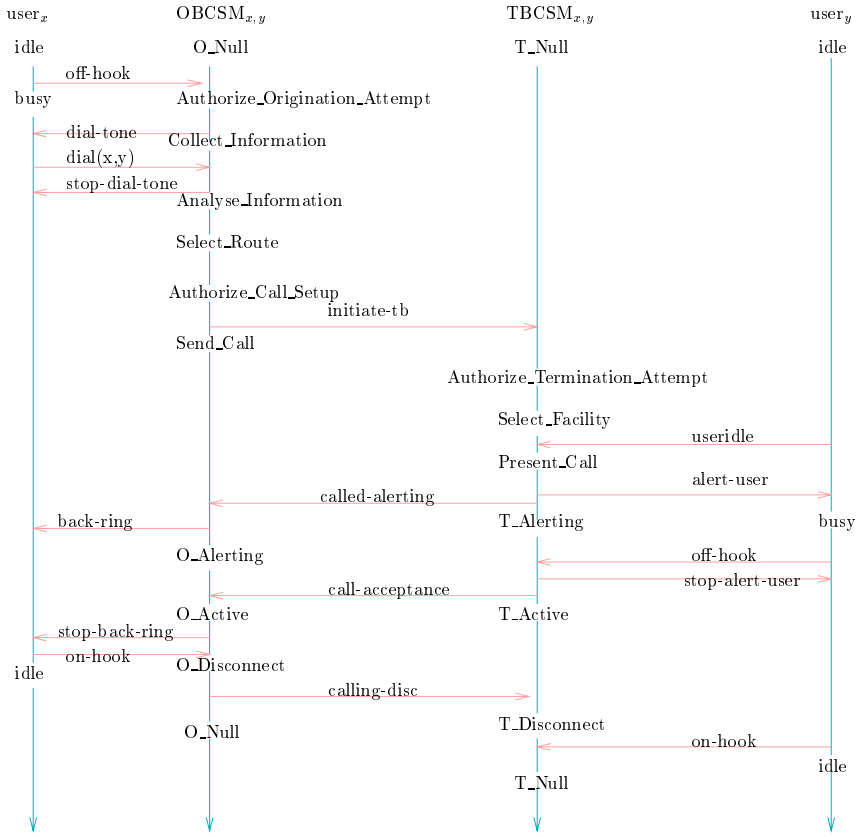

Figure 10: Sequence Charts of a POTS call

The encapsulation operation leads each message send and receive action in CW_y and $TBCSM_{x,y}$ to a communication action between the processes concerned or deadlock.

The CW service is activated when the *called party busy* event is encountered in the BCSM, and an INAP message *TBusy* is send to the CW service, asking for the service control over the subsequent call processing. Figure 11 describes a trace of a CW call. When $user_x$ initiates a call to a busy subscriber $user_y$, $TBCSM_{x,y}$ will detect a called party busy event, and transfer to *T_Busy* DP. At *T_Busy* DP, instead of sending *terminating-busy* to $OBCSM_{x,y}$, $TBCSM_{x,y}$ sends a message *TBusy* to CW_y, to establish the control relationship between CW_y and the basic call processing. Upon receiving the message, CW_y sends a message *MoveCallSegment*, requesting $TBCSM_{x,y}$ to send the call waiting tone to $user_y$. CW_y controls the subsequent call processing until the end of the service. In Figure 11, the control relationship is represented by a series of communications between CW_y and DPs such as *T_Busy*, *T_Mid_Call* in $TBCSM_{x,y}$ and *O_Suspend*, *O_Mid_Call* in $OBCSM_{y,z}$. At these DPs the BCSMs concerned take state transitions controlled by CW_y, instead of basic transitions described in Figure 1. In this example, we can see how the *call waiting* service is activated by the communication behavior between the IN service and BCSMs.

Theoretically speaking, it is possible to specify and verify all the services that are supported by IN CS-2 in this way (since our BCSM model is based on the one from CS-2). Actually, besides the *CW* service, PSF specifications of the *CFB*, *TWC*, etc., are also available, as described in Appendix C. A more encouraging result is, by enhancing some communication functions of BCSM (described in Section 4.1), we also formalized some services not supported by CS-2, such as *Automatic Recall (AR)* and *Call Completion to Busy Subscriber (CCBS)*. Similarly, it is possible to simulate all these service specifications on their user level behavior in the *PSF Toolkit*, and thus validate them.

5 Application on Feature Interactions

6 Introduction to the feature interaction problem

Telecommunication services provide additional function to telephone users. For example, the *call forwarding* service and the *call waiting* service modify the basic telephone service (i.e. POTS) that is offered to end users. Generally speaking, a telecommunication service includes one or more features, such as rerouting, number translation, etc. In this sense, features can be considered as *"elements"* of telecommunication services. However, the term *feature* has a flexible meaning. For example, *call forwarding* can be a feature of Freephone service, or an individual service offered by a service provider, depending on the application context. For this reason, we will not distinguish *feature* and *service* in this paper. Instead, they both can be considered as supplementary functions to POTS.

One aim of the development of the telecommunication network is to provide communication services quickly and conveniently. The introduction of new telecommunication services into the existing network can cause the interference between different services or different instances of the same service. Feature interactions occur if one or more services can not function well due to the interference of other services. More exactly, feature interactions are understood to be all interactions that interfere with the desired operation of the feature and that occur between a feature and its environment, including other features or other instances of the same feature [14].

Feature interactions can also happen when more than two services interfere with each other. In this case, the interactions can be bilateral (e.g. interactions between s1-s2, s2-s3 and s3-s1), multilateral interaction (e.g. interactions between s1-s2-s3) or roundabout (e.g. interactions between s1-s2 causes the interaction between s2-s3).

In the remainder part of Section 5.1, we first analyze the main causes of feature interactions, and explain three main methods in the feature interaction research. Finally, we briefly describe our research method.

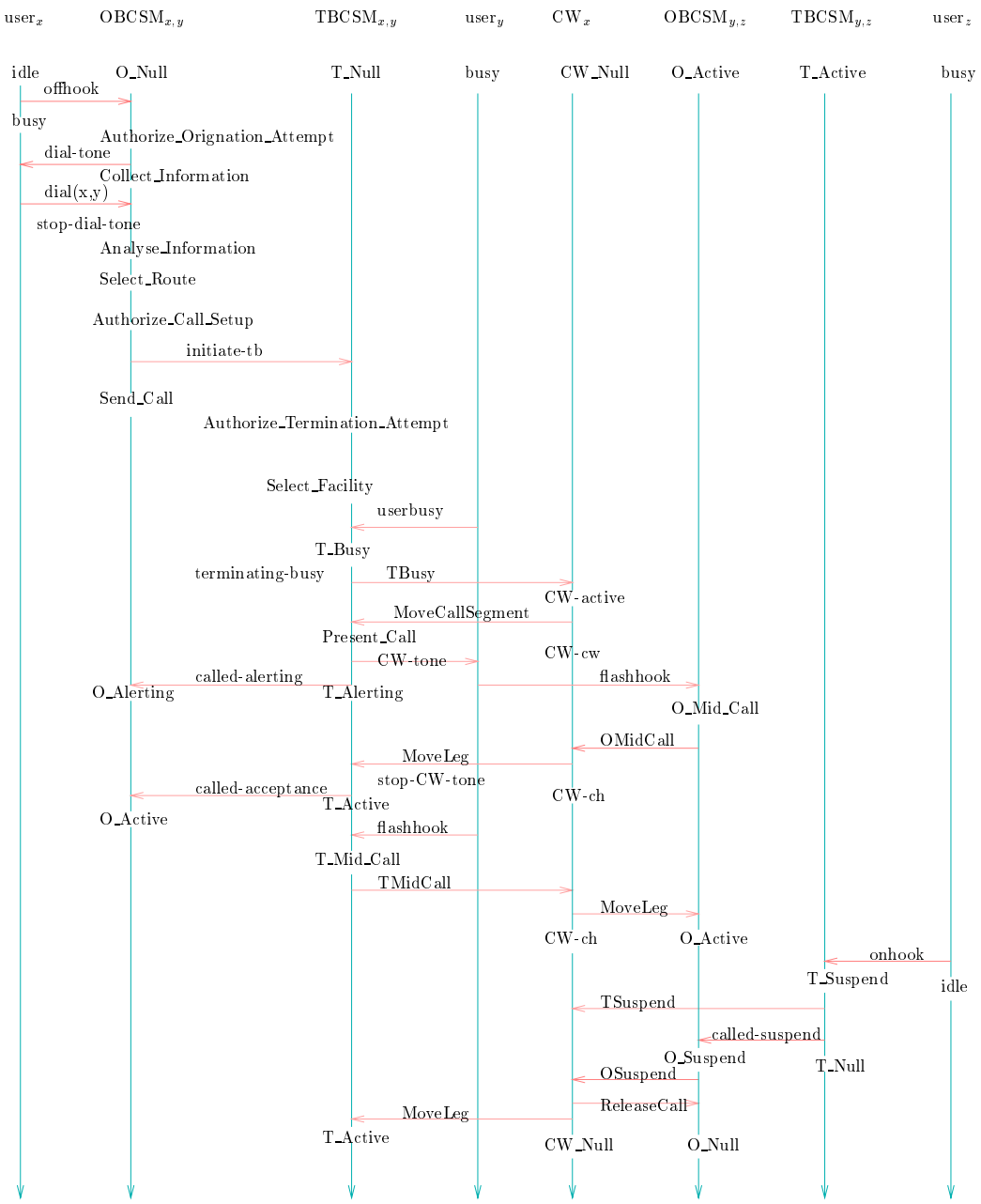

Figure 11: Sequence Charts of a CW Call

7 Categorization of the Causes of Feature Interactions

A lot of phenomena in a telecommunication network can result in feature interactions. Cameron et al in [14] categorize the many different causes of feature interactions into three main types.[14]

1. *Violation of feature assumption*

   Most of the features make assumptions about the telecommunication system, such as data availability, call processing model, billing method, etc. If the introduction of a new feature violates an assumption, feature interactions may occur.

   (a) Assumption about naming

   Historically, a telephone number uniquely identified a telephone line (one-number-one-line assumption). However the introduction of rerouting features (e.g. *call forwarding*), services including call distribution features such as *800-services* maps one called number to several user lines. Moreover, features like *Multiple Directory Number Line with Distinctive Ringing* permit more than one number to be assigned to a line, each with its own distinctive ring. The one-number-one-line assumption is again destroyed by mapping multiple called number to one physical line.

   (b) Assumption about data availability

   Some features requires certain data to be available from the network. For example, *Calling Number Delivery (CND)* requires *calling party number*. However, such assumption can be invalidated by other features or even the network capability. In the case of *CND*, if the calling party subscribe to the service *Calling Number Delivery Restriction (CNDR)* which makes the *calling number* private to the calling party, it is impossible for the device at the called side to show the calling number even with *CND* service. Actually, the *CND* can not function well if the call originates from an area lacking the equipment to transmit the *calling party number*. Note that in connection with this feature interaction that in some countries (e.g. Netherlands), when users subscribe to the CND service, they will be notified that *CND can not delivery the calling number to the called party in any call with the calling party subscribing to CNDR or when the telecommunication network is unable to transmit the calling party number*. Actually, the notification can be considered the supplementary description to the original CND service in the sense that it provides CND subscribers information on how CND interacts with CNDR, and the subscribers have to accept the result.

   (c) Assumption about administrative domain

   An administrative domain is a telephone network administrated by a single organization. A feature not supported in one administrative domain may not be usable from another administrative domain. Billing is a typical example. A common perception about billing is that a caller will not be billed for a call until the call is completed. When there are multiple administrative domains, the definition of a *complete call* can be confusing. Regional companies, inter-exchange carriers, even service providers of hotels, may all participate in connecting a simple call. If different administrative domains have different billing policy and techniques, feature interactions may occur when a call spans several administrative domains. Take *calling from hotel rooms* as an example. Many hotels contract with independent vendors to collect access charge information for calls originated from phones in their premises. Without being able to access the status of call connections, some billing applications developed by these vendors use a fixed amount of time to determine if a call is complete or not. Thus one can be billed for incomplete calls ringing a long time, or not billed for very short duration calls.

[14] Similar recitation can be found in Magill et al [23].

(d) Assumption about call control

Call control refers to the ability of a user to manipulate (e.g. accept, refuse, terminate or change the status of) a call. Interactions arise when the call control of one feature prevents other features from exercising their control. For example, when the calling party calls a busy user, the CW service generates a *call waiting* tone whereas *Answer Call* connects the calling party to an answering service. Suppose A subscribes to both CW and the *Answer Call*, if a second call comes in when A is already in conversation with someone else, should A receive a call waiting tone or should the second call be directed to the answering service?

(e) Assumption about signaling protocol

Some signals are used to indicate the called party's status, such as *busy tone*, *back ring*. New features, such as *call waiting* can make a line appear to be free when it is busy. Interactions will occur when one feature needs to know the status of the other party but cannot tell exactly what the status is from the signal received. For example, *call completion to busy subscriber (CCBS)* is triggered if the called party is busy. However a line with *call waiting (CW)* appears to be idle to a caller, although it is actually busy. Suppose a CCBS subscriber A calls a CW subscriber B when B is talking with C. In this case, will CCBS work for A?

2. *Limitations on network support*

Network components have their own limited capabilities in communicating with other network components or processing calls, which may bring interactions to seemingly independent features.

(a) Limited customer premised equipment (CPE) signaling

Standard telephone supports limited (actually, a few of) CPE signals, such as *flashhook*, #, $*$, etc. Interactions will occur if several features simultaneously act on the same CPE signaling but they have different interpretation of the signal.

(b) Limited functionalities for communications among network components

The network has limited support of communication between different network components, such as the signaling format, bandwidth, communicating frequency, etc. If features compete for limited network support, interactions may occur.

3. *Intrinsic problems in distributed system*

As any large, distributed, real-time system, telecommunication network has intrinsic problems which may causes feature interactions.

(a) resource contention

It is a typical problem for any distributed system. If two features compete for the same resource (data resource or hardware resource), the activation of one feature can make the other unavailable.

(b) Personalized instantiation

Many features allow each subscriber to assign personalized values (e.g. *call forwarding number*) to some feature parameters before using the features. In many cases a particular set of assignments can make two otherwise independent features interfere with each other.

(c) Timing and race conditions

Timing (i.e. at what time a particular event occurs, or for how long an event lasts) is always critical in distributed systems. For example, how soon a customer dials the second digit after a dialed *"0"* could change an intended *0+-service* call into an *operator-service* call. When more than one features are active, the interpretation of certain

signals may depend upon which feature is active when the signals arrive at the network component. Sometime, such communication delays can cause non-deterministic system behavior or infinite loops, thus feature interactions occur.

(d) Distributed support of features

The present network architecture allows different services to be supported by different network components (e.g. switches). Thus no single component is aware of all the services activated in a call and can coordinate the execution of them. Consequently, activation of one service in one network component can lead to interactions with services in another network component.

The last two classes of problems (limitations on network support and intrinsic problems in distributed systems) relate to the real implementation of services, covering network and distributed systems. The *violation of feature assumptions* represents more fundamental problems and are mostly implementation independent. These interactions are therefore arguably the most important of the three classes.

8 To Handle Feature Interaction with Various Techniques

Feature interactions bring unexpected and undesired system behavior to the service subscribers and users, and decrease the system performance. Research work must be done to bring the problem under control. Current research on the feature interaction problem includes three directions [13]:

1. Avoidance

   Avoidance technique can be used during service creation, in order to help generate a service which are intrinsically less prone to interactions.

   To make the service requirement analysis and service definition as precise as possible is of great use, since a lot of feature interactions are the result of un-precise service description in the beginning. Object-oriented software development techniques can be used for this purpose. Moreover, a telecommunication system can choose to provide service development tools. For example, in IN, the service is created in service creation environment (SCE), with testing and simulation tools. Moreover, IN also provides service independent building blocks (SIBs), and IN services can be composed of SIBs. This object-orient style service description is much more precise than the one provided in an old way of modifying the switching software.

   Enhancement of the network capability, such as, careful enrichment of the existing network protocol may eliminate some feature interactions. To use more sophisticated customer premises equipment (CPE) like ISDN (Integrated Switch Data Network) terminals is helpful too.

   For those interactions that can not be avoided, detection and resolution is necessary.

2. Detection

   Feature interaction detection can be done statically and dynamically.

   (a) Static Detection

   To detect feature interactions statically is to find the possible incompatibility between two or more telecommunication service before they are introduced the communication network. From the engineering point of view, it is favorable to build a method to detect the interactions at an early phase of service design. Leaving the problem to be found in a later phase only means to add more complexity into the network software, which may cause the system to be out of control. Moreover, the trend in the rapid and independent service deployment demands a quick and reliable computer supported detection method,

instead of slow manual analysis. Considering these points, the off-line spotting of feature interaction becomes very important.

Current research on the static detection of feature interactions focus on using formal techniques to specify and validate the telecommunication services. It includes two main aspects:

The first task is *service specification*. Two specification styles are available. One is to derive the formal specification of a service directly from its description in natural or other informal language. Each service is expressed as an independent finite state machine (FSM), and formal methods can be used to analyze the behavior and property of the "FSM". Specifications in [16, 18] are of this style. The advantage of this specification style is, it is quick and straightforward to formalize service in this way. Once the user requirement is analyzed and services are described in natural language, formal specification can be given based on the specifier's understanding of the service. However this method has some shortcomings. First it is due to the vagueness of natural language. This method relies heavily on specifier's understanding ability, which may vary from person to person according to his personal and technical background. Natural language is well-known for its vagueness and unclarity. How can we assure the service behavior we specified is really the behavior of what the user will experience in the network? There is huge difference between the service described in natural language and its implementation in the real network. Secondly, it is difficult to model the interworking between services, which is very important for feature interaction detection.

There is another specification style. For the target services we want to specify (often it is the case they are services sharing a very general property, for example, *voice services*, *non-mobile services*, etc.), we can build a unified call processing model for them all, and services can be specified as processes interworking with the call model to control its behavior. By properly specifying the interworking between the call model and the telecommunication services, we can build a system where the service behavior can be analyzed. The call model can be built on the existing ones. F.J.Lin in [17] use the basic all model (BCM) defined in Bellcore Advanced Intelligent Network (AIN) Release 0.2 for some AIN services.

This specification style in essence reflects the idea of *separating service control from call control*, which is introduced by the IN structure. The call model can be used to describe the network behavior on call and service processing, while services are described separately how they influence the functions of the call model.

In order to detect feature interactions, the interworking between different service instances and the *unwanted behaviors* must be formalized. Furthermore, feature interactions can be detected based on the deduction function of the formal methods.

In this specification style, besides the general principle described above, there are some other key points worth consideration,

i. Call processing models (e.g. BCM from AIN, BCSM from IN CS-2) include some states and transitions for network congestion, routing failure, etc. In the feature interaction research, we only care about the *unwanted* behavior to users due to the interference of services *themselves*. What should be concentrate on here is the potential interactions that can be caused by insufficient service requirement analysis, wrong implementation etc., but not the problems when the *unhealthy* network is not able to support a *healthy* service.
   When formalizing the call model, some methods must be used to avoid irrelevant details. Meanwhile, this work must be done carefully not to loss important information and make the call model ineffective.

ii. How to combine the call model and services and build them into a system is also a problem. It is largely dependent on the formal method used in the specification.

In most cases, the interworking is modeled by communication between different entities.

iii. In feature interactions detection, lack of formal definitions of what feature interaction is makes the problem very confused.

In [2], European Telecommunication Standards Institute (ETSI) proposes the term *spotting* to replace *static detection*, in order to emphasize on the distinguishment between the *static detection* and the *dynamic detection*. However, due to the historical reasons in the telecommunication world, *detection* is now a popular term which means the same as *static detection*. We follow this tradition, and always write *dynamic detection* explicitly.

Static detection, together with the static analysis of service behavior, forms the basis to solve the feature interaction problem. The detection result should be expressed in a form that can be used later for handling the feature interactions on-line. The result is also a reference if service redesign is necessary.

A lot of theoretical work is done in the feature interaction detection. Formal methods applied in this area are, state-transition based language (SDL), process algebra such as Communication Sequential Process (CSP), Algebra of Communicating Process (ACP) and others such as Promela, Object-Z.

In our point of view, the formal method itself is not the decisive point for the quality of detection result, but the method must be fit for the description of the call model.

Despite of all the theoretical work on this area, there are only few detection methods that has been put into effective implementation. From this point of view, it is advisable if the chosen formal method is supported by computer-aided tools.

(b) Dynamic detection

In the dynamic feature interaction detection, the telecommunication system operators should provide mechanism (e.g. execution tracing) to monitor the behavior of all activated service instances and to report the feature interactions before they manifest themselves.

S.Tsang and E.H.Magil in [19] provide a typical dynamic detection method. They introduce a feature and service manager into the system, which is responsible for checking the services' behavior after they are activated. More exactly, each service instance has a pre-defined state transition sequence called *state signatures* in [19]. When the executed state transitions are not consistent with the one in the state signatures, feature interactions are detected.

In order to detect feature interactions dynamically, the telecommunication network should provide functions such as service execution tracing to judge when and how feature interactions manifest themselves. However, most of the existing network do not provide such functions, so how to enhance the network structure for the purpose of dynamic feature interaction detection is of great interest.

3. Resolution

There are dynamic resolution and static resolution too.

Static resolution is the off-line redefinition and re-description of the service. Since a large amount of feature interactions are the result of un-precise service descriptions at the early phase of service creation, making service description as precise as possible will help to obtain a better service implementation.

Dynamic resolution is to employ additional functions (could be distributed) in the existing network, to negotiate between interacting services and help perplexed users to find a way out. Not all feature interactions can be solved in a unified way. Some features are not allowed to interwork, the firewall must be set between different features by assigning different priority

for different features. Some interworking between features is allowed, with mechanism to negotiate their behavior when interactions are possible to occur.

Adding a service to a telecommunication system in fact is to add or modify part of its software. For such a large distributed and wide-spread heterogeneous software system, it is hard to analyze and predicate what the exact influence when we do such modifications. To foreseen whether a group of services will cause undesired system behavior to their subscribers and users is even more harder.

Because of the complexity of the telecommunication system, it is advisable to use an integrated method to solve the feature interaction problem. Firstly, the method includes the use of avoidance technique at the phrase of service design (e.g. object-oriented service design). Secondly, before a new service is introduced to the telecommunication system, it must be validated to find whether there are feature interactions between the new service and those existing ones. This should be done with the help of formal methods and formal tools. Thirdly, if there are feature interactions, the service should be redesigned or some on-line feature interaction handling mechanism must be built to solve the feature interaction dynamically.

9 Our Methods on Feature Interaction Detection

Categorization of Cameron et al provides a better understanding of the feature interaction problem. However, it is not so effective if we want to develop a detection method with which *feature interactions of a certain type can be detected using a strategy specialized for them.*

Although telecommunication services have complicated behavior, they implement their functions via two basic ways: either by changing the data in a call (e.g. screening some data, forbidding some data to be written by other services), or by changing the switching and connection control procedure in the call. This notion is consistent with how IN services interwork with BCSMs via INAP: each INAP message consumed by a BCSM is to make some data (un)available to the BCSM, or to change the state transitions in the BCSM, or sometimes both.

From this point of view, feature interactions can be categorized as two basic types:

1. *Data invalidation*

   A telecommunication network carries a lot of data, and the execution of each service needs to operate (e.g. read, write) on certain data. If two services have an inconsistent way of operate on some data, for example, one service wants to obtain the data which are not supported by the other service, feature interactions occur. The interaction between *call number delivery (CND)* and *call number delivery restriction (CNDR)* belongs to this category.

2. *Control interaction*

   Control interactions arise when services active on one or more calls have different connection and switching control and thus interfere with each other.

   For example, any telecommunication system should be free of *livelock* and *non-determinism.* However, this can not hold true due to control interactions which can result in non-deterministic system behavior. Competing for CPE signals is a typical type of control interaction. In Section 5.2 and Section 5.3, we give some examples about the *livelock* and *non-determinism* caused by control interactions.

   Moreover, the execution of each service makes some properties hold true for the telecommunication system. For example, a *Terminal Call Screening* (TCS) subscriber will never receive a call from an unexpected party. However,when feature interactions occur, one or more properties can not hold due to the interference of other services. In this case, the call processing can be finished, but some properties established by one or more features are invalidated due

to control interactions. The interaction between *Originating Call Screening (OCS)* and *Abbreviate Dialing (ABD)* services, as well as other examples presented in Section 5.4 belong to this type.

The detection of feature interaction caused by *data invalidation* requires careful pre-analysis of the data operation at the early phase of service design. Meanwhile, since a large number of feature interactions are caused by incompatibility between different service control functions and it is of great importance to analyze them with formal method, in this paper we emphasize on such interactions. We formally define feature interactions which cause *non-determinism*, *livelock* in the telecommunication system, or *invalidate the property of other services*. Furthermore, we show by examples on how to detect these interactions.

In the Section 5.2, Section 5.3 and Section 5.4, we describe by examples, how to detect feature interactions by analyzing the behavior of parallel processes described in ACP^{τ}. The method is based on the call and service processing model in Section 2.3. IN services provide additional features by controlling the state transitions in BCSMs. When more than one IN services are interacting on a group of BCSMs, it is possible for the system to run into livelocks, non-determinism or property invalidation between different IN services, thus feature interactions occur. This paper describes how to detect feature interactions by detecting the non-determinism and livelocks in BCSM state transitions, as well as how to define property invalidation between services and thus detect feature interactions caused by property invalidation.

9.1 Detection of Non-determinacy Feature Interactions

9.1.1 Introduction to Non-determinacy Feature Interactions

When designing the communication system and deploying telecommunication services in the system, we must ensure the system behavior is deterministic and reliable to telephone users. For example, if an user picks up the handset, he will soon hear a dial-tone. If an user dials a number, he will either get connected to another subscriber or a voice mail box, or be notified by the network such a connection is not possible at that moment. In other words, what will happen in a telephone call, is decided by the state of end terminals, actions users have taken, and functions of the telecommunication services concerned. If at any time, it is left for the telecommunication network to decide what will happen to an user, the system is considered to be non-deterministic.

9.1.2 Definition of Non-determinacy Feature Interactions

Considering the call and service processing model we proposed, if we map the communication actions on channels between the service and the network, and on channels between different BCSMs to silent actions in ACP_{τ}, the telecommunication system can be viewed as showed in Figure 12, where, the communication network is completely a black box. The only observable actions of the telecommunication system are the communication actions between the network and users.

We write u_{2i} for communication channels from $user_i$ to the network, u_{2i-1} for channels from the network to $user_i$, $i \in [1 \cdot \cdot m]$.

Definition 1 (Relationship between Process States) *Let $S, S^{'} S_1, S_2$ be process states expressed in ACP_{τ}; $a, a_1, a_2, \cdots, a_n$ be atomic actions, $n \in \mathbb{N}$; $\sigma, \sigma^{'}$ be sequences of atomic actions, and $\sigma = a_1 \cdot a_2 \cdots a_n$, $\sigma^{'} = a_2 \cdots a_n$. Then*

1. $S \xrightarrow{a} S_1 \Leftrightarrow \exists S^{'} \bullet S = a \cdot S_1 + S^{'}$
2. $S \xrightarrow{\sigma} S_1 \Leftrightarrow \exists S^{'} \bullet S \xrightarrow{a} S' \wedge S' \xrightarrow{\sigma^{'}} S_1$
3. $S \not\xrightarrow{g} S_1 \Leftrightarrow \neg(S \xrightarrow{a} S_1)$

The intuitive idea of the relations between system states is as follows: [15]

[15] See *action relations* in [6] for more details

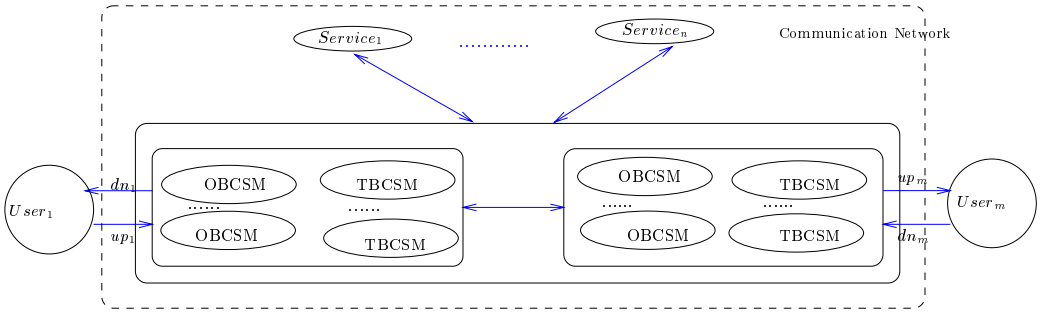

Figure 12: Abstraction of Telecommunication System

- $S \xrightarrow{a} S_1$ denotes that S can execute a and thereby evolve into S_1; i.e. in the state S, performing the action a can lead the process into the state S_1.
- $S \xrightarrow{\sigma} S_1$ denotes that S can evolve into S_1 via the sequence of actions σ.
- $S \stackrel{a}{\not\rightarrow} S_1$ denotes that the process S can not perform the action a and evolve into the state S_1.

Definition 2 (System Non-determinacy: Case 1) *A telecommunication system* S *has system non-determinacy if state transitions of the following form exist*

$S \xrightarrow{\sigma} S_1 \stackrel{c_{u_{2i-1}}(sig_1)}{\longrightarrow} S_2^{'} \wedge S \xrightarrow{\sigma} S_1 \stackrel{c_{u_{2i-1}}(sig_2)}{\longrightarrow} S_2^{''}$

where S_1, $S_2^{'}$ *and* $S_2^{''}$ *are states of the telecommunication system;* σ *is a non-empty sequence of communication events between users and the network;* $c_{u_{2i-1}}(sig_1)$ *and* $c_{u_{2i-1}}(sig_2)$ *are communication actions of* sig_1 *and* sig_2 *from the communication network to the* $user_i$ *on the channel* u_{2i-1}, $i \in [1 \cdots m]$, $sig_1 \neq sig_2$. [16]

Definition 3 (System Non-determinacy: Case 2) *A telecommunication system* S *has system non-determinacy if state transitions of the following form exist*

$S \xrightarrow{\sigma} S_1 \stackrel{c_{u_{2i-1}}(sig_1)}{\longrightarrow} S_2^{'} \wedge S \xrightarrow{\sigma} S_2^{''} \wedge S_2^{''} \stackrel{c_{u_{2i-1}}(sig_1)}{\not\longrightarrow} S_2^{'}$

where S_1, $S_2^{'}$ *and* $S_2^{''}$ *are states of the telecommunication system;* σ *is a non-empty sequence of communication events between users and the network;* $c_{u_{2i-1}}(sig_1)$ *is the communication action of* sig_1 *from the communication network to the* $user_i$ *on the channel* u_{2i-1}, $i \in [1 \cdots m]$.

The definition indicates that, in a telecommunication system, after certain communication actions between users and the network, if the network has more than one choice on how to react to a subscriber, the system is non-deterministic. The non-deterministic choice can be what signal to send to the subscriber (Case 1), or whether to send a signal to the subscriber (Case 2).

Definition 4 (Non-determinacy Feature Interaction) *For a set of services* $\{ f_1, \cdots, f_n \}$, *there is a non-determinacy feature interaction between* $f_1, \cdots, f_n$ *if the system* S *defined below has*

[16] Two communication signals are considered to be equal in our model when they have the same signal name and parameters.

system non-determinacy (either case 1 or case 2).

$$
\begin{aligned}
S \quad = \quad & \tau_C(\partial_H(\\
& f_1 \parallel \cdots \parallel f_n \parallel \\
& user_1 \parallel \cdots \parallel user_m \parallel \\
& OBCSM_1 \parallel \cdots \parallel TBCSM_1 \parallel \\
& \cdots\cdots \\
& OBCSM_s \parallel \cdots \parallel TBCSM_s)) \\
& n, m, s \in \mathbb{N}
\end{aligned}
$$

We write K for the set of all the channels in the system, $K^{'}$ for the set of channels between different BCSMs and between the service and the BCSM, H for the set of signals receiving and sending events on every channel in K, C for the set of communication actions on on every channel in $K^{'}$.

Formally, we have

$$
\begin{aligned}
H \quad &= \quad \{s_k(sig), r_k(sig) \mid k \in K, sig \in U \cup V \cup I\} \\
C \quad &= \quad \{c_{k'}(sig) \mid k \in K^{'}, sig \in V \cup I\}
\end{aligned}
$$

9.1.3 Examples of Non-determinacy Feature Interaction

Example 1 (Interaction between CW and CFB) *Suppose that a subscriber* y *has subscribed to the CFB service in addition to the CW service, An ambiguous situation will arise in the system if* x *initiates a call to* y *when* y *is in connection with* z*. When encountering the called party busy event, CW will send a call waiting tone to* y *to indicate an incoming call is waiting, but CFB will forward the call from* x *to* z1 *without informing* y*. The inconsistency is, whether to send* y *a call waiting tone or not.*

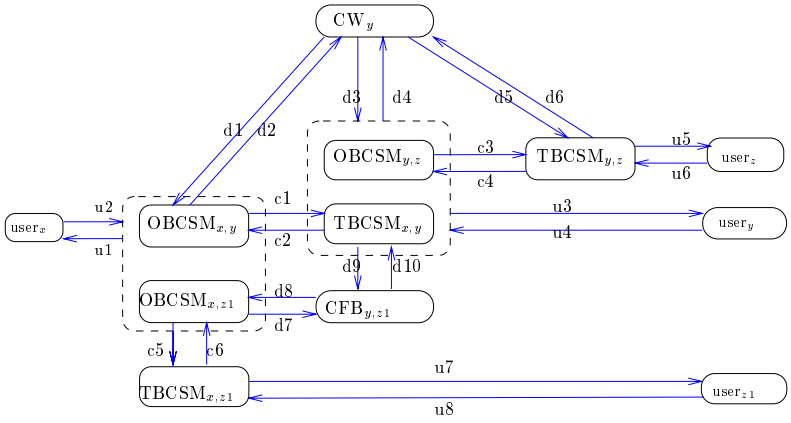

Figure 13: Parallel composition of CW and CFB

Figure 13 is the model of this call scenario. In the two party call between y and z, we suppose y is the calling party. The case of y being the called party is similar to this one.

There are three two-party calls involved in this call scenario, each represented as a pair of OBCSM and TBCSM. *User$_x$'s* behavior is influenced by the two BCSMs he/she is in communication with: $OBCSM_{x,y}$ and $OBCSM_{x,z1}$. Similarly, *User$_y$'s* behavior is influenced by $OBCSM_{y,z}$ and $TBCSM_{x,y}$.

$CFB_{y,z1}$ is the *CFB* service instance responsible for forwarding the incoming call to *z1* when y is busy; it communicates with $OBCSM_{x,z1}$ and $TBCSM_{x,y}$ to control their state transitions. CW_y is the instance of the *CW* service y subscribed to; it communicates with $OBCSM_{y,z}$, $TBCSM_{y,z}$, $OBCSM_{x,y}$ and $TBCSM_{x,y}$ to control their state transitions.

When the incoming call from x arrives at y and y is busy, an INAP message *T_Busy* will be reported by $TBCSM_{x,y}$ to $CFB_{y,z1}$ and CW_y simultaneously. However the two service instances have different interpretation for this INAP message, which will result in the feature interaction described at the beginning of this example. Below the feature interaction is described precisely.

The definition of this system is

$$\begin{aligned}
CW2CFB_{x,y,z,z1} &= \partial_{H_{CW2CFB}}(\\
&\quad user_x \parallel user_y \parallel user_z \parallel user_{z1} \parallel \\
&\quad OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\
&\quad OBCSM_{y,z} \parallel TBCSM_{y,z} \parallel \\
&\quad OBCSM_{x,z1} \parallel TBCSM_{x,z1} \parallel \\
&\quad CW_y \parallel CFB_{y,z1}) \\
H_{CW2CFB} &= \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot\cdot 8]\} \\
&+ \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot\cdot 6]\} \\
&+ \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot\cdot 10]\}
\end{aligned}$$

Suppose that

$$\begin{aligned}
S &= \tau_{C_{CW2CFB}}(CW2CFB_{x,y,z,z1}) \\
&\quad C_{CW2CFB} = \{c_{c_i}(sig), c_{d_i}(sig') \mid sig \in V, sig' \in I, i \in [1 \cdot\cdot 6]\} \\
\sigma &= c_{u4}(offhook(y)) \cdot c_{u3}(dial_tone(y)) \cdot c_{u3}(stop_dial_tone(y)) \cdot \\
&\quad c_{u4}(dial(y,z)) \cdot c_{u6}(useridle(z)) \cdot c_{u5}(alert_user(y,z)) \cdot \\
&\quad c_{u5}(stop_alert_user(y,z)) \cdot c_{u6}(offhook(z)) \cdot c_{u2}(offhook(x)) \cdot \\
&\quad c_{u1}(dial_tone(x)) \cdot c_{u1}(stop_dial_tone(x)) \cdot c_{u2}(dial(x,y)) \cdot c_{u4}(userbusy(y))
\end{aligned}$$

Then we have
$S \xrightarrow{\sigma} S_1'' \xrightarrow{c_{u3}(CWtone(y,x))} S_1''$ and
$S \xrightarrow{\sigma} S_2''$ and
$S_2'' \not\xrightarrow{c_{u3}(CWtone(y,x))} S_1''$

where

$$
\begin{aligned}
S_1^{'} &= \tau_{CCW2CFB}(\partial_{HCW2CFB}(\\
&\quad idle_{z1} \parallel busy_y \parallel busy_x \parallel busy_z \\
&\quad O\_Active_{y,z} \parallel T\_Active_{y,z} \parallel \\
&\quad Send\_Call_{x,y} \parallel T\_Busy_{x,y} \parallel \\
&\quad O\_Null_{x,z1} \parallel T\_Null_{x,z1} \parallel \\
&\quad CW\_Null_y \parallel CFB\_Null_{y,z1})) \\
S_1^{''} &= \tau_{CCW2CFB}(\partial_{HCW2CFB}(\\
&\quad idle_{z1} \parallel busy_y \parallel busy_x \parallel busy_z \\
&\quad O\_Active_{y,z} \parallel T\_Active_{y,z} \parallel \\
&\quad O\_Alerting_{x,y} \parallel T\_Alerting_{x,y} \parallel \\
&\quad O\_Null_{x,z1} \parallel T\_Null_{x,z1} \parallel \\
&\quad CW\_call\_wait_{x,y} \parallel CFB\_Null_{y,z1})) \\
S_2^{''} &= \tau_{CCW2CFB}(\partial_{HCW2CFB}(\\
&\quad busy_{z1} \parallel idle_y \parallel busy_x \parallel idle_z \\
&\quad O\_Active_{y,z} \parallel T\_Active_{y,z} \parallel \\
&\quad O\_Null_{x,y} \parallel T\_Null_{x,y} \parallel \\
&\quad Send\_Call_{x,z1} \parallel Present\_Call_{x,z1} \parallel \\
&\quad CW\_Null_{x,y} \parallel CFB\_forward_{y,x,z1}))
\end{aligned}
$$

$CW\_call\_wait_{x,y}$ and $CFB\_forward_{y,x,z1}$ are service execution states. See Appendix D for their meaning. $O\_Active_{y,z}$, $T\_Active_{y,z}$, $Send\_Call_{x,y}$, $Present\_Call_{x,z1}$, etc., are BCSM states. See Appendix E for their meaning.

Below is the general description of the system states:

- $S_1^{'}$: y is in connection with z; x has initiated a call to y; T_Busy is encountered in $TBCSM_{x,y}$.
- $S_1^{''}$: The CW service is activated; call waiting tone is being sent to y; back ring is being sent to x.
- $S_2^{''}$: The CFB service is activated, which redirected the call from x to *z1*; $OBCSM_{x,z1}$ and $TBCSM_{x,z1}$ are establishing the call from x to *z1*.

Example 2 (Interaction between CW and TWC) *Suppose a subscriber* y *has subscribed to the TWC service and the CW service, and* x *initiates a call to* y *when* y *is in connection with* z*;* y *will flash-hook after hearing a call waiting tone, thus an ambiguous situation arises in the system. According to CW, the flash-hook is understood to be the acceptance of a CW call, and* y *will be connected to* x*; but according to TWC, the flash-hook is interpreted as a signal to initiate a new call from* y *(supposed the new call is to* z1*), and* y *will hear a dial-tone. The ambiguity is, whether to send* y *a dial-tone or not?*

Figure 14 is the model of this call scenario. Note that for the convenience to analyze the feature interaction, we suppose the new call in the TWC context is a call to *z1*. In the two party call between y and z, we suppose y is the calling party. The case of y being the called party is similar to this one.

There are three two-party calls involved in this call scenario, each represented as a pair of OBCSM and TBCSM. $User_y$*'s* behavior is influenced by three BCSMs he/she is in communication with: $OBCSM_{y,z}$, $TBCSM_{x,y}$ and $OBCSM_{y,z1}$.

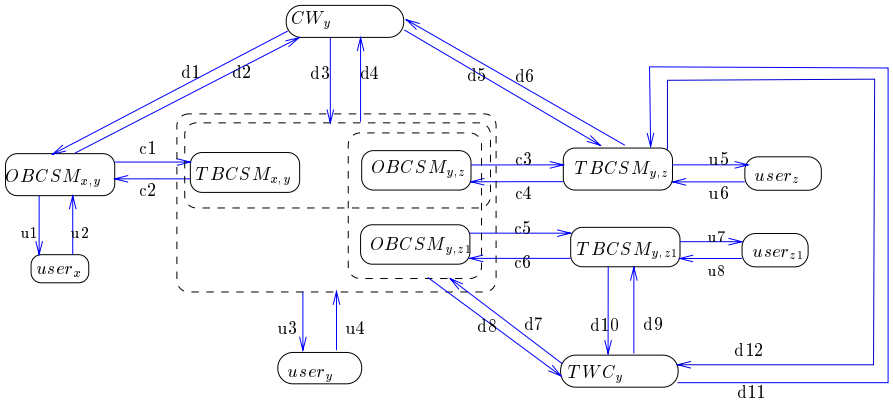

Figure 14: Parallel Composition of CW and TWC

TWC_y is the instance of the TWC service y subscribes to; it communicates with $OBCSM_{y,z}$, $TBCSM_{y,z}$, $OBCSM_{y,z1}$ and $TBCSM_{y,z1}$ to control their state transitions. CW_y is the instance of the CW service y subscribed to; it communicates with $OBCSM_{y,z}$, $TBCSM_{y,z}$, $OBCSM_{x,y}$ and $TBCSM_{x,y}$ to control their state transitions.

When the incoming call from x arrives at y and y flashhooks, and an INAP message *O_Mid_Call* will be reported by $OBCSM_{y,z}$ to TWC_y and CW_y simultaneously to report this call event. However the two service instances have different interpretation for this INAP message, which will result in the feature interaction described at the beginning of this example. Below the interaction is described precisely.

The definition of this system is

$$
\begin{aligned}
CW2TWC_{x,y,z,z1} &= \partial_{H_{CW2TWC}}(\\
&\quad user_x \parallel user_y \parallel user_z \parallel user_{z1} \parallel \\
&\quad OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\
&\quad OBCSM_{y,z} \parallel TBCSM_{y,z} \parallel \\
&\quad OBCSM_{y,z1} \parallel TBCSM_{y,z1} \parallel \\
&\quad CW_y \parallel TWC_y) \\
H_{CW2TWC} &= \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot \cdot 8]\} \\
&+ \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot \cdot 6]\} \\
&+ \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot \cdot 12]\}
\end{aligned}
$$

Suppose that

$$
\begin{aligned}
S &= \tau_{C_{CW2TWC}}(CW2TWC_{x,y,z,z1}) \\
&\quad C_{CW2TWC} = \{c_{c_i}(sig), c_{d_j}(sig') \mid sig \in V, sig' \in I, i \in [1 \cdot \cdot 6], j \in [1 \cdot \cdot 2]\} \\
\sigma &= c_{u4}(offhook(y)) \cdot c_{u3}(dial\_tone(y)) \cdot c_{u3}(stop\_dial\_tone(y)) \cdot \\
&\quad c_{u4}(dial(y,z)) \cdot c_{u6}(useridle(z)) \cdot c_{u5}(alert\_user(y,z)) \cdot \\
&\quad c_{u5}(stop\_alert\_user(y,z)) \cdot c_{u6}(offhook(z)) \cdot c_{u2}(offhook(x)) \cdot \\
&\quad c_{u1}(dial\_tone(x)) \cdot c_{u1}(stop\_dial\_tone(x)) \cdot c_{u2}(dial(x,y)) \cdot \\
&\quad c_{u4}(userbusy(y)) \cdot c_{u3}(CWtone(x,y)) \cdot c_{u4}(flashhook(y))
\end{aligned}
$$

Then we have

$S \xrightarrow{\sigma} S_1^{'} \xrightarrow{c_{u3}(dial\_tone(y))} S_1^{''}$ and
$S \xrightarrow{\sigma} S_2^{''}$, and
$S_2^{''} \not\xrightarrow{c_{u3}(dial\_tone(y))} S_1^{''}$
where

$$
\begin{aligned}
S_1^{'} &= \tau_{C_{CW2TWC}}(\partial_{H_{CW2TWC}}(\\
&\quad idle_{z1} \parallel busy_y \parallel busy_x \parallel busy_z \\
&\quad O\_Mid\_Call_{y,z} \parallel T\_Active_{y,z} \parallel \\
&\quad O\_Alerting_{x,y} \parallel T\_Alerting_{x,y} \parallel \\
&\quad O\_Null_{y,z1} \parallel T\_Null_{y,z1} \parallel)) \\
&\quad CW\_call\_wait_y \parallel TWC\_Null_y \\
S_1^{''} &= \tau_{C_{CW2TWC}}(\partial_{H_{CW2TWC}}(\\
&\quad idle_{z1} \parallel busy_y \parallel busy_x \parallel busy_z \\
&\quad O\_Mid\_Call_{y,z} \parallel T\_Active_{y,z} \parallel \\
&\quad O\_Alerting_{x,y} \parallel T\_Alerting_{x,y} \parallel \\
&\quad Collect\_Information_{y,z1} \parallel T\_Null_{y,z1} \parallel \\
&\quad CW\_call\_wait_{x,y} \parallel TWC\_M\_orig_y)) \\
S_2^{''} &= \tau_{C_{CW2TWC}}(\partial_{H_{CW2TWC}}(\\
&\quad busy_{z1} \parallel idle_y \parallel busy_x \parallel idle_z \\
&\quad O\_Mid\_Call_{y,z} \parallel T\_Active_{y,z} \parallel \\
&\quad O\_Active_{x,y} \parallel T\_Active_{x,y} \parallel \\
&\quad O\_Null_{y,z1} \parallel T\_Null_{y,z1} \parallel \\
&\quad CW\_call\_hold_{x,y} \parallel TWC\_Null_y))
\end{aligned}
$$

$CW\_call\_wait_{x,y}$, $CW\_call\_hold_{x,y}$, TWC_Null_y and $TWC\_M\_orig_y$ are service execution states. See Appendix D for their meaning. $O\_Null_{y,z1}$, $Collect\_Information_{x,y}$, etc., are BCSM states. See Appendix E for their meaning.

Below is the general description of the system states.

- $S_1^{'}$: The O_Mid_Call event is encountered in $OBCSM_{y,z}$, when y is in connection with z; the CW service is activated.
- $S_1^{''}$: The TWC service is activated; dial-tone is being sent to y; $OBCSM_{y,z1}$ and $TBCSM_{y,z1}$ are establishing the call from y to *z1*; the call from y to z is being on hold.
- $S_2^{''}$: The call from z to y is active; the call from y to z is being on hold.

Example 3 (Interaction between two instances of the CW service) *Suppose that both* y *and* z *have subscribed to the CW service (denoted by* CW_y *and* CW_z *respectively), and* y *has put* z *on hold to talk to* x*. While on hold,* z *flash-hooks to answer an incoming call from* z1*. After a period of conversation with* z1*,* z *flash-hooks again to return to the call with* y*, but finding that he is still being hold by* y*,* z *on-hooks to end the call. The on-hook action can be interpret by the system in an ambiguous way. According to the* CW_y*, the on-hook is simply interpreted as a disconnection from the called party on hold, thus* x *and* y *are placed in a normal two-way conversation, and* z *is idled. However according to* CW_z*, the on-hook is from a CW subscriber with a call on hold, thus* z *will be rung back [14]. The system has non-deterministic choices on whether to ring back* z *or not.*

Figure 15 is the model of this call scenario. In the two party call between y and z, we suppose y is the calling party. The case of y being the called party is similar to this one.

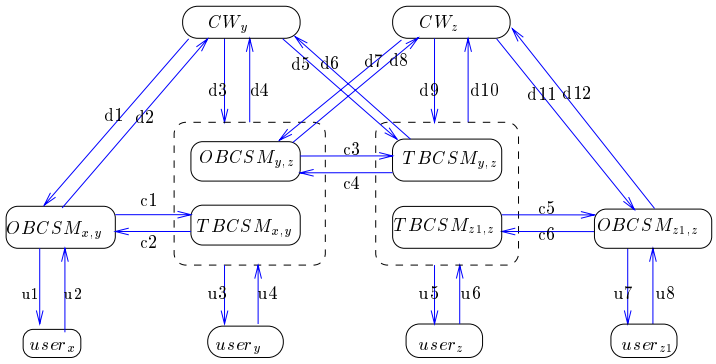

Figure 15: Parallel Composition of two instances of the CW service

There are three two-party calls involved in this call scenario, each represented as a pair of OBCSM and TBCSM. $User_y$'s behavior is influenced by two BCSMs he/she is in communication with: $OBCSM_{y,z}$, $TBCSM_{x,y}$. $User_z$'s behavior is influenced by two BCSMs he/she is in communication with: $TBCSM_{y,z}$, $TBCSM_{z1,z}$.

CW_y communicates with $OBCSM_{y,z}$, $TBCSM_{y,z}$, $OBCSM_{x,y}$ and $TBCSM_{x,y}$ to control their state transitions. CW_z communicates with $OBCSM_{y,z}$, $TBCSM_{y,z}$, $OBCSM_{z1,z}$ and $TBCSM_{z1,z}$ to control their state transitions.

When z on-hooks to end the call, an INAP message *O_TDisconnect* will be reported by $TBCSM_{y,z}$ to both CW_x and CW_y simultaneously. However the two service instances have different interpretation for this INAP message, which will result in the feature interaction described at the beginning of this example. Below the interaction is described precisely.

The definition of this system is

$$
\begin{aligned}
CW2CW_{x,y,z,z1} &= \partial_{H_{CW2CW}}(\\
&\quad user_x \parallel user_y \parallel user_z \parallel user_{z1} \parallel \\
&\quad OBCSM_{y,z} \parallel TBCSM_{y,z} \parallel \\
&\quad OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\
&\quad OBCSM_{z1,z} \parallel TBCSM_{z1,z} \parallel \\
&\quad CW_y \parallel CW_z) \\
H_{CW2CW} &= \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot\cdot 8]\} \\
&+ \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot\cdot 6]\} \\
&+ \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot\cdot 12]\}
\end{aligned}
$$

Suppose that

$$
\begin{aligned}
S &= \tau_{C_{CW2CW}}(CW2CW_{x,y,z,z1}) \\
&\quad C_{CW2CW} = \{c_{c_i}(sig), c_{d_j}(sig') \mid sig \in V, sig' \in I, i \in [1 \cdot\cdot 8], j \in [1 \cdot\cdot 4]\}
\end{aligned}
$$

Then we have
$S \xrightarrow{\sigma_0} S_1 \xrightarrow{\sigma_1} S_2 \xrightarrow{\sigma_2} S_3 \xrightarrow{\sigma_3} S_3 \xrightarrow{c_{u5}(alert\_user(x,y))} S_4'$, and
$S \xrightarrow{\sigma_0} S_1 \xrightarrow{\sigma_1} S_2 \xrightarrow{\sigma_2} S_3 \xrightarrow{\sigma_3} S_4''$, and $S_4'' \overset{c_{u5}(alert\_user(x,y))}{\not\longrightarrow} S_4'$

where

$$
\begin{aligned}
\sigma_0 &= c_{u4}(\mathit{offhook}(y)) \cdot c_{u3}(\mathit{dial\_tone}(y)) \cdot c_{u3}(\mathit{stop\_dial\_tone}(y)) \cdot \\
&\quad c_{u4}(\mathit{dial}(y,z)) \cdot c_{u6}(\mathit{useridle}(z)) \cdot c_{u5}(\mathit{alert\_user}(y,z)) \cdot \\
&\quad c_{u3}(\mathit{back\_ring}(y,z)) \cdot c_{u6}(\mathit{offhook}(z)) \cdot c_{u5}(\mathit{stop\_alert\_user}(y,z)) \cdot \\
&\quad c_{u3}(\mathit{stop\_back\_ring}(y,z)) \\
\sigma_1 &= c_{u2}(\mathit{offhook}(x)) \cdot c_{u1}(\mathit{dial\_tone}(x)) \cdot c_{u1}(\mathit{stop\_dial\_tone}(x)) \cdot \\
&\quad c_{u2}(\mathit{dial}(x,y)) \cdot c_{u4}(\mathit{userbusy}(y)) \cdot c_{u3}(\mathit{CW\,tone}(x,y)) \cdot \\
&\quad c_{u1}(\mathit{back\_ring}(x,y)) \cdot c_{u4}(\mathit{flashhook}(y)) \cdot c_{u3}(\mathit{stop\_CW\,tone}(x,y)) \cdot \\
&\quad c_{u1}(\mathit{stop\_back\_ring}(x,y)) \\
\sigma_2 &= c_{u8}(\mathit{offhook}(z1)) \cdot c_{u7}(\mathit{dial\_tone}(z1)) \cdot c_{u7}(\mathit{stop\_dial\_tone}(z1)) \cdot \\
&\quad c_{u8}(\mathit{dial}(z1,z)) \cdot c_{u6}(\mathit{userbusy}(z)) \cdot c_{u5}(\mathit{CW\,tone}(z1,z)) \cdot \\
&\quad c_{u1}(\mathit{back\_ring}(z1,z)) \cdot c_{u6}(\mathit{flashhook}(z)) \cdot c_{u5}(\mathit{stop\_CW\,tone}(z1,z)) \cdot \\
&\quad c_{u1}(\mathit{stop\_back\_ring}(z1,z)) \cdot c_{u6}(\mathit{flashhook}(z)) \\
\sigma_3 &= c_{u6}(\mathit{onhook}(z)) \\
S_1 &= \tau_{C_{CW2CW}}(\partial_{H_{CW2CW}}(\\
&\quad \mathit{idle}_x \parallel \mathit{busy}_y \parallel \mathit{busy}_z \parallel \mathit{idle}_{z1} \parallel \\
&\quad O\_\mathit{Active}_{y,z} \parallel T\_\mathit{Active}_{y,z} \parallel \\
&\quad O\_\mathit{Null}_{x,y} \parallel T\_\mathit{Null}_{x,y} \parallel \\
&\quad O\_\mathit{Null}_{z1,z} \parallel T\_\mathit{Null}_{z1,z} \parallel \\
&\quad \mathit{CW\_Null}_y \parallel \mathit{CW\_Null}_z)) \\
S_2 &= \tau_{C_{CW2CW}}(\partial_{H_{CW2CW}}(\\
&\quad \mathit{busy}_x \parallel \mathit{busy}_y \parallel \mathit{busy}_z \parallel \mathit{idle}_{z1} \parallel \\
&\quad O\_\mathit{Mid\_Call}_{y,z} \parallel T\_\mathit{Active}_{y,z} \parallel \\
&\quad O\_\mathit{Active}_{x,y} \parallel T\_\mathit{Active}_{x,y} \parallel \\
&\quad O\_\mathit{Null}_{z1,z} \parallel T\_\mathit{Null}_{z1,z} \parallel \\
&\quad \mathit{CW\_call\_hold}_{y,x,z} \parallel \mathit{CW\_Null}_z)) \\
S_3 &= \tau_{C_{CW2CW}}(\partial_{H_{CW2CW}}(\\
&\quad \mathit{busy}_x \parallel \mathit{busy}_y \parallel \mathit{busy}_z \parallel \mathit{busy}_{z1} \parallel \\
&\quad O\_\mathit{Mid\_Call}_{y,z} \parallel T\_\mathit{Active}_{y,z} \parallel \\
&\quad O\_\mathit{Active}_{x,y} \parallel T\_\mathit{Active}_{x,y} \parallel \\
&\quad O\_\mathit{Active}_{z1,z} \parallel T\_\mathit{Mid\_Call}_{z1,z} \parallel \\
&\quad \mathit{CW\_call\_hold}_{y,x,z} \parallel \mathit{CW\_call\_hold}_{z,y,z1})) \\
S_4^{'} &= \tau_{C_{CW2CW}}(\partial_{H_{CW2CW}}(\\
&\quad \mathit{busy}_x \parallel \mathit{busy}_y \parallel \mathit{idle}_z \parallel \mathit{busy}_{z1} \parallel \\
&\quad O\_\mathit{Null}_{y,z} \parallel T\_\mathit{Null}_{y,z} \parallel \\
&\quad O\_\mathit{Active}_{x,y} \parallel T\_\mathit{Active}_{x,y} \parallel \\
&\quad O\_\mathit{Active}_{z1,z} \parallel T\_\mathit{Mid\_Call}_{z1,z} \parallel \\
&\quad \mathit{CW\_call\_hold}_{y,x,z} \parallel \mathit{CW\_Null}_z)) \\
S_4^{''} &= \tau_{C_{CW2CW}}(\partial_{H_{CW2CW}}(\\
&\quad \mathit{busy}_x \parallel \mathit{busy}_y \parallel \mathit{idle}_z \parallel \mathit{busy}_{z1} \parallel \\
&\quad O\_\mathit{Null}_{y,z} \parallel T\_\mathit{Null}_{y,z} \parallel \\
&\quad O\_\mathit{Active}_{x,y} \parallel T\_\mathit{Active}_{x,y} \parallel \\
&\quad O\_\mathit{Active}_{z1,z} \parallel T\_\mathit{Mid\_Call}_{z1,z} \parallel \\
&\quad \mathit{CW\_Null}_y \parallel \mathit{CW\_call\_hold}_{z,y,z1}))
\end{aligned}
$$

$O\_Mid\_Call_{y,z}$, $T\_Active_{x,y}$, etc., are BCSM states. See Appendix E for their meaning. Because of the complexity of this example, we have given more states here. Below is the general description of the system states:

S_1: y is in connection with z.

S_2: The CW service y subscribed to is activated;
y is in voice connection with z, and z is being put on hold.

S_3: The CW service z subscribed to is activated;
z has put *z1* is on hold.

$S_4^{'}$: call from y to z is cleared by the CW service z subscribed to;
z is being alerted that *z1* is still on hold;
y is in the voice connection with x.

$S_4^{''}$: call from y to z is cleared by the CW service y subscribed to;
y is in the voice connection with x.

Example 4 (Interaction between CCBS and CW) *Suppose that* y *and* x *have subscribed to the CW service (denoted by* CW_y*) and the CCBS (denoted by* $CCBS_x$*) service respectively, The user* y *is in conversation with the user* z *when an incoming call from* x *arrives at* y*. A called party busy event is encountered on* y *and the system has inconsistent way to deal with this event: according to the* CW_y*, a CW tone should be sent to* y *and the* back ring *should be sent to* x*, but according to the* $CCBS_x$*,* x *will hear a* busy tone*. The inconsistency is whether to ring back* x *or send* busy tone *to the user* x.

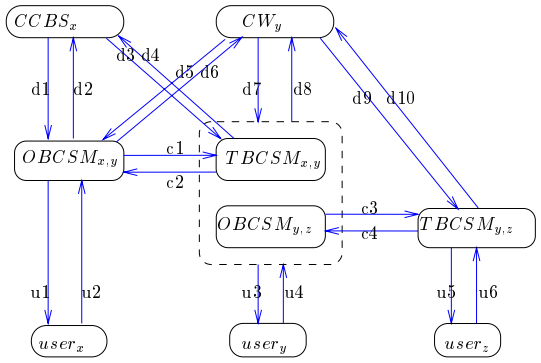

Figure 16: Parallel Composition of two instances of the CW service

Figure 16 is the parallel processing model of such a call scenario. In the two party call between y and z, we suppose y is the calling party. The case of y being the called party is similar to this one.

There are two two-party calls involved in this call scenario, each represented as a pair of OBCSM and TBCSM. $User_x$*'s* behavior is influenced by $OBCSM_{x,y}$; $user_y$*'s* behavior is influenced by two BCSMs he/she is in communication with: $OBCSM_{y,z}$, $TBCSM_{x,y}$; $user_z$*'s* behavior is influenced by $TBCSM_{y,z}$.

CW_y communicates with $OBCSM_{y,z}$, $TBCSM_{y,z}$, $OBCSM_{x,y}$ and $TBCSM_{x,y}$ to control their state transitions. $CCBS_x$ communicates with $OBCSM_{x,y}$, $TBCSM_{x,y}$, $OBCSM_{y,z}$.

When an incoming call from x arrives at y and y is busy, an INAP message *T_Busy* will be reported by $TBCSM_{x,y}$ to both $CCBS_x$ and CW_y simultaneously. However the two services have different interpretation for this INAP message, which will result in the feature interaction described at the beginning of this example. Below the interaction is described precisely.

The definition of this system is

$$\begin{aligned}
CCBS2CW_{x,y,z} &= \partial_{H_{CCBS2CW}}(\\
&\quad user_x \parallel user_y \parallel user_z \parallel \\
&\quad OBCSM_{y,z} \parallel TBCSM_{y,z} \parallel \\
&\quad OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\
&\quad CW_y \parallel CCBS_x) \\
H_{CCBS2CW} &= \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot\cdot 6]\} \\
&+ \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot\cdot 4]\} \\
&+ \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot\cdot 10]\}
\end{aligned}$$

Suppose that

$$\begin{aligned}
S &= \tau_{C_{CCBS2CW}}(CCBS2CW_{x,y,z,z1}) \\
&\quad C_{CCBS2CW} = \{c_{c_i}(sig), c_{d_j}(sig^{'}) \mid sig \in V, sig^{'} \in I, i \in [1 \cdot\cdot 4], j \in [1 \cdot\cdot 10]\}
\end{aligned}$$

Then we have
$S \xrightarrow{\sigma_0} S_1 \xrightarrow{c_{u1}(back\_ring(x,y))} S_1^{'} \wedge$
$S \xrightarrow{\sigma_0} S_1 \xrightarrow{c_{u1}(busy\_tone(x,y))} S_1^{''}$ where

$$\begin{aligned}
\sigma_0 &= c_{u4}(offhook(y)) \cdot c_{u3}(dial_tone(y)) \cdot c_{u3}(stop_dial_tone(y)) \cdot \\
&\quad c_{u4}(dial(y,z)) \cdot c_{u6}(useridle(z)) \cdot c_{u5}(alert_user(y,z)) \cdot \\
&\quad c_{u3}(back_ring(y,z)) \cdot c_{u6}(offhook(z)) \cdot c_{u5}(stop_alert_user(y,z)) \cdot \\
&\quad c_{u3}(stop_back_ring(y,z)) \cdot c_{u2}(offhook(x)) \cdot c_{u1}(dial_tone(x)) \cdot \\
&\quad c_{u1}(stop_dial_tone(x)) c_{u4}(dial(x,y)) \cdot c_{u4}(userbusy(y)) \cdot c_{u3}(CWtone(x,y)) \\
S_1 &= \tau_{C_{CCBS2CW}}(\partial_{H_{CW2CW}}(\\
&\quad busy_x \parallel busy_y \parallel busy_z \parallel \\
&\quad O_Active_{y,z} \parallel T_Active_{y,z} \parallel \\
&\quad Send_Call_{x,y} \parallel T_Busy_{x,y} \parallel \\
&\quad CW_call_wait_{x,y} \parallel CCBS_active_{x,y})) \\
S_1^{'} &= \tau_{C_{CCBS2CW}}(\partial_{H_{CW2CW}}(\\
&\quad busy_x \parallel busy_y \parallel busy_z \parallel \\
&\quad O_Active_{y,z} \parallel T_Active_{y,z} \parallel \\
&\quad O_Alerting_{x,y} \parallel T_Alerting_{x,y} \parallel \\
&\quad CW_call_wait_{x,y} \parallel CCBS_active_{x,y})) \\
S_1^{''} &= \tau_{C_{CW2CW}}(\partial_{H_{CW2CW}}(\\
&\quad busy_x \parallel busy_y \parallel busy_z \parallel \\
&\quad O_Active_{y,z} \parallel T_Active_{y,z} \parallel \\
&\quad O_Called_Party_Busy_{x,y} \parallel T_Busy_{x,y} \parallel \\
&\quad CW_call_wait_{x,y} \parallel CCBS_active_{x,y}))
\end{aligned}$$

Below is the general description of the system states:

S_1: The user y is in connection with z, and a call from x arrives at y;
both the CW service and the CCBS service are activated.
$S_1^{'}$: The user y is in voice connection with z, x is waiting for the response from y.
$S_1^{''}$: The *busy_tone* is being sent to x.

In the examples given above, PSF plays an important role in analyzing the system states and the communications between processes for the detection of non-determinacy feature interactions.

9.2 Detection of Livelock Feature Interactions

9.2.1 Introduction to Livelock Feature Interactions

Any system must be livelock free, otherwise some critical task can never be finished. Livelocks in a system composed of parallel processes are caused by internal loops in some processes, or unbounded communication loops between parallel processes.

In designing the telecommunication system, we must ensure it is free of livelock of both kinds. Firstly, any individual process, such as the switching process, the user process, etc., must be livelock free. And, of course, when introducing a supplementary service to the system, the service itself should not bring livelock into the system. Therefor we suppose a telecommunication system is livelock free, if the system only provides POTS service, or there is only one instance of the supplementary service in the system.

What will happen if several services or several instances of the same service are executed in parallel? Is the system still free of unbounded internal communication loops, in which the user will not realize he/she is doing something that can never end? The answer is no, because supplementary services change the call processing procedure; their parallel execution may cause unbounded communications between processes, which finally results in one or more calls that can not terminate normally. Such behavior is called livelock feature interaction.

Next, we give the formal definition of the livelock feature interactions, and show by examples how to detect such interactions.

9.2.2 Definition of Livelock Feature Interaction

Section 2.3 presents a telecommunication system model consisting of a number of communicating processes; each of the processes such as the BCSM, the user, the supplementary service should be livelock free. In this system, livelocks occur due to the unbounded communication loop between processes (as described in in Example 5 and Example 6). However, there is another type of communication loops in the telecommunication system, which does not lead to livelock. For example, any user in the telecommunication system is able to initiate as many POTS calls as he wants. This statement can be described by unbounded communication loops on the communication channels between the network and the users: the calling party off-hooks, dials the number of and gets connected with the called party; after a period of conversation, the calling and the called on-hooks to terminate the current call; and then the calling party lifts the handset, dials the number of the called party again $\cdots$. Unbounded communication loops occur in the telecommunication system, which are obviously not livelocks: the users know what they are doing, and each of the telephone calls can terminate normally.

The difference between the two kinds of communication loops makes the definition of livelock more complicated. In order to distinguish them, we first consider those communications in which users play an active role (by the word *active*, we mean the user is able to decide whether to send the signal or not.). Of course, in the communications between internal processes in the communication network, such as communications between different BCSMs, or between the BCSM and the service, users play passive roles for these communications are invisible to them. Communications between users and communication network can be categorized as:

1. The user plays an active role in the communications of signals such as *offhook*, *onhook*, *flashhook*; he/she is able to decide whether the communication will happen or not.

2. The user plays a passive role in the communication of signals which report the terminal user's state, such as *useridle*, *userbusy*, for communication of these signals are actually decided upon the user's state, and they are sent on the request of the communication network.

3. The user plays a passive role in the communications of signals from the network to users, for in such communications, users only accept some information from the network.

We use set T to describe the communications in which the user plays a passive role. The element in T is either a communication action between the user and the network, or a sequence of such communication actions. The character of elements in T is that they can not change the call processing procedure and the system behavior. If the system follows a unbounded loop, which is composed of elements in T only, we declare there is livelock in the telecommunication system. Notably, if *onhook* follows a *busy_tone* on the same user, we put the two actions into T together and consider the user to be passive in the communication of both signals. This because a sensible telephone user will always *onhook* after hearing the *busy_tone*. In the case of feature interaction research, we assume the users are sensible in order to emphasize on the service behavior itself. Similarly, we put an *offhook* following a *special_tone* on the same user into T.

We write $ADDR$ for the set of terminal addresses, K for the set of channel identifiers in the telecommunication system, then we have

$$\begin{aligned} T \quad = \{ \quad & c_k(special\_tone(x)), c_k(busy\_tone(x,y)), \\ & c_k(dial\_tone(x)), c_k(stop\_dial\_tone(x), \\ & c_k(back\_ring(x,y)), c_k(stop\_back\_ring(x,y)), \\ & c_k(alerting\_tone(x,y)), c_k(stop\_alerting\_tone(x,y)), \\ & c_k(CW\_tone(x,y)), c_k(stop\_CW\_tone(x,y)), \\ & c_k(stop\_special\_tone(x)), \\ & c_k(busy\_tone(x,y)) \cdot c_k(onhook(x)), \\ & c_k(special\_tone(x)) \cdot c_k(offhook(x)), \\ & c_k(userbusy(x)), c_k(useridle(x)) \\ & \mid x, y \in ADDR, k \in K\} \\ + \quad & \{c_k(sig) \mid sig \in V \cup I, k \in K\}^{17} \end{aligned}$$

Definition 5 (System Livelock) *A telecommunication system* S *has* **system livelock** *if the following state transitions exist*

$S \xrightarrow{\sigma_0} S_1 \xrightarrow{\sigma} S_1$

where S_1 *is a state of the telecommunication system,* $\sigma = P_1 \cdots P_j, j \in \mathbb{N}$, $P_1, \cdots, P_j \in \mathrm{T}$; σ *is the sequence of communication actions between users and the network.*

The definition describes a situation in the telecommunication system, in which it is possible for the telecommunication system to take unbounded communications invisible to the terminal users. The result of the unbounded communications is one or more calls can not terminate without the interference of users.

Definition 6 (Livelock Feature Interaction) *For a set of services* $\{ f_1, \cdots, f_n \}$, $f_1, \cdots, f_n$ *have* **livelock feature interactions** *if the telecommunication system* S *defined below has system livelock and* $\mathrm{S}_i, (1 \leq i \leq n)$ *defined below is free of system livelock, where*

[17] I is the set of signals between different BCSMs; I is the set of INAP messages.

$$
\begin{aligned}
S &= \tau_C(\partial_H(\\
&\quad f_1 \parallel \cdots \parallel f_n \parallel \\
&\quad user_1 \parallel \cdots \parallel user_m \parallel \\
&\quad OBCSM_1 \parallel \cdots \parallel TBCSM_1 \parallel \\
&\quad \cdots\cdots \\
&\quad OBCSM_s \parallel \cdots \parallel TBCSM_s)) \\
S_i &= \tau_C(\partial_H(\\
&\quad f_i \parallel user_1 \parallel \cdots \parallel user_m \parallel \\
&\quad OBCSM_1 \parallel \cdots \parallel TBCSM_1 \parallel \\
&\quad \cdots\cdots \\
&\quad OBCSM_s \parallel \cdots \parallel TBCSM_s)) \\
&\quad s \in \mathbb{N}^{18}
\end{aligned}
$$

S_i is the ACP_τ description of the telecommunication system which provides the service f_i only.

9.2.3 Examples of Livelock Feature Interaction

Example 5 (Interaction between three instances of the CFB service) *A Call Forwarding on Busy (CFB) service subscriber can forward an incoming call to a new destination when busy. Suppose users* x, y *and* z *have all subscribed to the CFB service, and the forwarding destinations are* y, z *and* x *respectively (denoted as* $CFB_{x,y}$, $CFB_{y,z}$ *and* $CFB_{z,x}$ *respectively). If* x *is initiating a call to* y *when* y *and* z *are busy, the call from* x *to* y *will be forward to* z*, then back to* x*, then to* y*; and so on. Thus an infinite loop occurs.*

There are three two-party calls involved in this call scenario. Figure 17 is the model. $CFB_{x,y}$ communicates with $OBCSM_{x,y}$, $TBCSM_{x,x}$ to control their state transitions. Similarly, $CFB_{y,z}$ communicates with $OBCSM_{x,z}$, $TBCSM_{x,y}$, and $CFB_{z,x}$ communicates with $OBCSM_{x,x}$, $TBCSM_{x,z}$ to control their state transitions.

The definition of this system is

$$
\begin{aligned}
CFB2CFB_{x,y,z} &= && (5)\\
&\quad \partial_{H_{CFB2CFB}}(&& idle_x \parallel busy_y \parallel busy_z \parallel \\
&&& OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\
&&& OBCSM_{x,z} \parallel TBCSM_{x,z} \parallel \\
&&& OBCSM_{x,x} \parallel TBCSM_{x,x} \parallel \\
&&& CFB_{x,y} \parallel CFB_{y,z} \parallel CFB_{z,x}) && (6)\\
H_{CFB2CFB} &= && \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot\cdot 6]\} \\
&+ && \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot\cdot 6]\} \\
&+ && \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot\cdot 12]\}
\end{aligned}
$$

Suppose that

$$
\begin{aligned}
S &= \tau_{C_{CFB2CFB}}(CFB2CFB_{x,y,z}) \\
&\quad C_{CFB2CFB} = \{c_{c_i}(sig), c_{d_i}(sig') \mid sig \in V, sig' \in I, i \in [1 \cdot\cdot 6]\} \\
\sigma_0 &= c_{u2}(offhook(x)) \cdot c_{u1}(dial\_tone(x)) \cdot c_{u1}(stop\_dial\_tone(x)) \cdot c_{u1}(dial(x,y)) \\
\sigma &= c_{u3}(userbusy(y)) \cdot c_{u5}(userbusy(z)) \cdot c_{u2}(userbusy(x))
\end{aligned}
$$

[18] See Definition 4 for the meaning of τ_C and ∂_H.

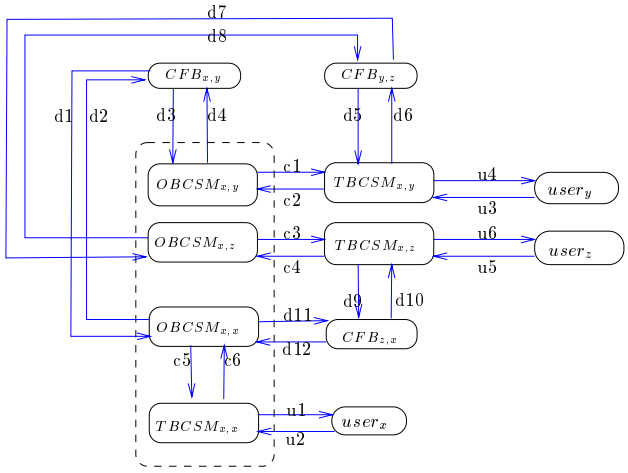

Figure 17: Modeling of the Parallel Execution of CFB services

Then we have $S \xrightarrow{\sigma_0} S_1 \xrightarrow{\sigma} S_1$, where

$$
\begin{aligned}
S_1 \quad = \quad & \tau_{C_{CFB2CFB}}(\partial_{H_{CFB2CFB}}(\\
& busy_x \parallel busy_y \parallel busy_z \parallel \\
& Send\_Call_{x,y} \parallel Select\_Facility_{x,y} \parallel \\
& O\_Null_{x,z} \parallel T\_Null_{x,z} \parallel \\
& O\_Null_{x,x} \parallel T\_Null_{x,x} \parallel \\
& CFB\_Null_{x,y} \parallel CFB\_Null_{y,z} \parallel CFB\_Null_{z,x}))
\end{aligned}
$$

S_1 describes a state where x, y and z are busy; the system is processing the call from x to y and the CFB services are not activated yet.

The feature interaction between three instances of the CFB service is detected as livelock in the system defined by Equation 7. After x dials y, because of the livelock, x gets no response from the system, and he will be kept on waiting without realizing something is wrong in the system. We find the livelock when simulating the system in PSF Toolkit, and the process traces are described in Figure 18. To save space, the DP points in BCSM's state transitions are omitted except for T_Busy; the states of user x, y and z start from *busy*.

Example 6 (Interaction between CCBS and AR) *When a Call Completion to Busy Subscriber (CCBS) service subscriber* x *dials to* y *that turns out to be busy, CCBS will be activated. CCBS automatically redials* y *until* y *is idle and the call from* x *to* y *is completed. On the other hand, Automatic ReCall (AR) automatically initiates a call from* y *to the latest calling party for the subscriber* y*, whether* y *answers the call or not. Suppose a CCBS subscriber* x *dials to a busy user* y*, who has subscribed to AR. CCBS and AR are activated when the called party busy event is detected. When* y *becomes idle, CCBS will send a special ring to* x*, to establish the call from* x *to* y*; at the same time, AR will send a special ring to* y*, to establish the call from* y *to* x*. What will happen to the system if the two services get synchronized? Both* x *and* y *will hear a special alerting tone at the same time. Suppose that they are very patient users, who will always lift the handset after the alerting tone. User* x *and* y *may lift the handset at the same time, so neither of CCBS and AR can establish the connection between* x *and* y *because both of the terminal users are busy. Now* x *and* y *will hear a busy tone and onhook. Since none of the CCBS and the AR fulfills their task, they will try again when* y *becomes* idle*. Suppose that the synchronization between the*

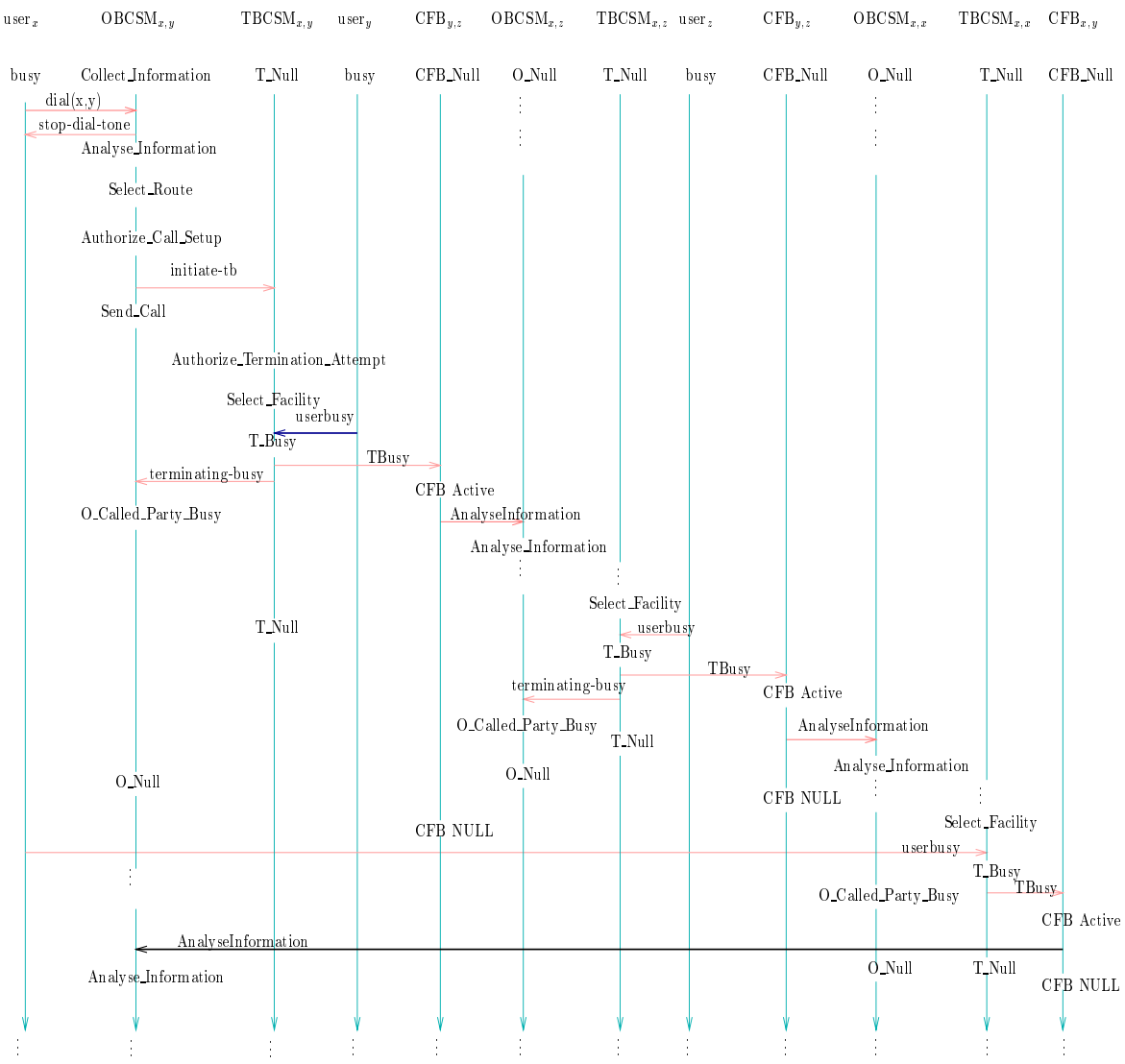

Figure 18: Livelocks in the Parallel Execution of CFB services

CCBS and the AR keeps occurring, the communication between x *and* y *will never be established, and the service and call processing will never end.*

Figure 19 is the parallel processing model of such a call scenario.

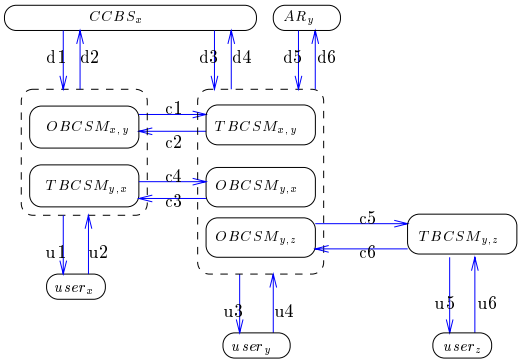

Figure 19: Modeling of the Parallel Execution of AR and CCBS

The formal description of this system is

$$
\begin{aligned}
AR2CCBS_{x,y,z} \quad &= \\
&\partial_{H_{AR2CCBS}}(\quad idle_x \parallel idle_y \parallel idle_z \parallel \\
&\qquad OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\
&\qquad OBCSM_{y,z} \parallel TBCSM_{y,z} \parallel \\
&\qquad OBCSM_{y,x} \parallel TBCSM_{y,x} \parallel \\
&\qquad AR_y \parallel CCBS_x) \\
H_{AR2CCBS} \quad &= \quad \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot\cdot 6]\} \\
&+ \quad \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot\cdot 6]\} \\
&+ \quad \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot\cdot 6]\}
\end{aligned}
$$

Suppose that

$$
\begin{aligned}
S &= \tau_{C_{AR2CCBS}}(AR2CCBS_{x,y,z}) \\
&\quad C_{AR2CCBS} = \{c_{c_i}(sig), c_{d_i}(sig^{'}) \mid sig \in V, sig^{'} \in I, i \in [1 \cdot\cdot 6]\} \\
\sigma_0 &= c_{u4}(offhook(y)) \cdot c_{u3}(dial\_tone(y)) \cdot c_{u3}(stop\_dial\_tone(y)) \cdot \\
&\quad c_{u4}(dial(y,z)) \cdot c_{u6}(useridle(z)) \cdot c_{u6}(offhook(z)) \cdot \\
&\quad c_{u2}(offhook(x)) \cdot c_{u1}(dial\_tone(x)) \cdot c_{u1}(stop\_dial\_tone(x)) \cdot \\
&\quad c_{u2}(dial(x,y)) \cdot c_{u4}(userbusy(y)) \cdot c_{u1}(busy\_tone(x,y)) \cdot \\
&\quad c_{u2}(onhook(x)) \cdot c_{u4}(onhook(y)) \cdot c_{u6}(onhook(z)) \\
\sigma &= c_{u1}(special\_tone(x)) \cdot c_{u2}(offhook(x)) \cdot c_{u1}(stop\_special\_tone(x)) \cdot \\
&\quad c_{u3}(special\_tone(y)) \cdot c_{u4}(offhook(y)) \cdot c_{u3}(stop\_special\_tone(y)) \cdot \\
&\quad c_{u2}(userbusy(x)) \cdot c_{u4}(userbusy(y)) \cdot c_{u1}(busy\_tone(x,y)) \cdot \\
&\quad c_{u2}(onhook(x)) \cdot c_{u3}(busy\_tone(y,x)) \cdot c_{u4}(onhook(y))
\end{aligned}
$$

Then we have $S \xrightarrow{\sigma_0} S_1 \xrightarrow{\sigma} S_1$, where

$$
\begin{aligned}
S_1 \quad = \quad & \tau_{C_{AR2CCBS}}(\partial_{H_{AR2CCBS}}(\\
& idle_x \parallel idle_y \parallel idle_z \parallel \\
& O\_Null_{x,y} \parallel T\_Null_{x,y} \parallel \\
& O\_Null_{y,z} \parallel T\_Null_{y,z} \parallel \\
& O\_Null_{y,x} \parallel T\_Null_{y,x} \parallel \\
& AR\_active_y \parallel CCBS\_active_x))
\end{aligned}
$$

S_1 describes a state where x, y and z are busy; AR and CCBS have been activated.

The synchronization between AR and CCBS leads to livelocks in the system, which results in very annoying behaviors to the terminal users x and y, who keep on lifting the handset after being alerted, but only to hear a busy tone from the network. Figure 20 describes the execution traces of the parallel processes in the system. To save space, we omit all Detection Points except those communicating with AR or CCBS.

In this example, in order to facilitate analyzing the feature interaction, we make assumption that AR and CCBS get synchronized internally. This assumption is arguable in a real telecommunication system. However, if we consider livelock in a broader sense: not only **infinitive** internal communications, but also **finite** internal communication loops unexpected and thus deteriorate the service quality, this example make sense for even AR and CCBS get synchronized two or three times (and this is possible), the system behavior is already unsatisfactory to end users.

9.3 Detection of Property Invalidation Feature Interactions

9.3.1 Introduction to Property Invalidation Feature Interaction

Each telecommunication service has certain properties. For example, an *originating call screen* subscriber can never originate a call to a called party on the screening list; a *terminating call screen* subscriber can never accept a call from a calling party in the screening list. Sometimes, when several services are active in the same call, one service's property is invalidated by a subsequent feature, thus feature interaction occurs. Here below, we formally define this type of feature interactions (named as *property invalidation feature interactions* in our paper) based on ACP$^\tau$ and show by examples how to detect such interactions.

9.3.2 Definition of Property Invalidation Feature Interaction

Below are the terms used in the formal definition of property invalidation feature interactions.

Let us write S, $S^{'}$ and $S^{''}$ for the processes described in certain form of ACP$^\tau$; a for the atomic action; σ and $\sigma^{'}$ for sequences of atomic actions (possibly empty). Σ is a set of sequences of atomic actions. We define the relationship between a process and an atomic action as follows:

Definition 7 (Relationship between the Process and the Atomic Action)

1. S *can result in* a.
   $S \models a \Leftrightarrow \exists \sigma, S \xrightarrow{\sigma} S^{'} \xrightarrow{a} S^{''}$
2. S *can not result in* a.
   $S \models \neg a \Leftrightarrow \neg \exists \sigma, S \xrightarrow{\sigma} S^{'} \xrightarrow{a} S^{''}$
3. S *can result in* a, *without* $\sigma^{'}$ *taking place first.*
   $S, \neg\sigma^{'} \models a \Leftrightarrow \exists \sigma, \sigma^{'} \not\subset \sigma \wedge S \xrightarrow{\sigma} S^{'} \xrightarrow{a} S^{''}$

4. S *can not result in* a*, without* $\sigma^{'}$ *taking place first.*

$S, \neg\sigma^{'} \models \neg a \Leftrightarrow \neg\exists\sigma, \sigma^{'} \not\subset \sigma \wedge S \xrightarrow{\sigma} S^{'} \xrightarrow{a} S^{''}$

Additionally, we have

$$\begin{aligned}
\sigma^{'} \subset \sigma \quad &\Leftrightarrow \quad \forall t \in \sigma^{'} \bullet t \in \sigma \wedge \\
&\qquad \forall t_1, t_2 \in \sigma^{'}, \exists \sigma_1, \sigma_2, \sigma_3, \sigma_1^{'}, \sigma_2^{'}, \sigma_3^{'} \bullet \\
&\qquad \sigma^{'} = \sigma_1^{'} \cdot t_1 \cdot \sigma_2^{'} \cdot t_2 \cdot \sigma_3^{'} \wedge \sigma = \sigma_1 \cdot t_1 \cdot \sigma_2 \cdot t_2 \cdot \sigma_3 \\
\sigma^{'} \not\subset \sigma \quad &\Leftrightarrow \quad \neg(\sigma^{'} \subset \sigma)
\end{aligned}$$

For simplicity, we define
$S, \neg\sigma^{'} \wedge \neg\sigma^{''} \models a \Leftrightarrow S, \neg\sigma^{'} \models a \wedge S, \neg\sigma^{''} \models a$
$S, \neg\Sigma \models a \Leftrightarrow \forall\sigma \in \Sigma, S, \neg\sigma \models a$
Similarly, we have
$S, \neg\sigma^{'} \wedge \neg\sigma^{''} \models \neg a$
$S, \neg\Sigma \models \neg a$

Definition 8 (Property Invalidation Feature Interaction) *For a set of services* $\{ f_1, \cdots, f_n \}$*,* $f_1, \cdots, f_n$ *have* **property invalidation feature interaction** *if one of the following situation occurs in the telecommunication system* S *defined below:*

$S_i \models a \wedge$ $S \models \neg a$	$S_i \models \neg a \wedge$ $S \models a$
$S_i, \neg\sigma \models a \wedge$ $S, \neg\sigma \models \neg a$	$S_i, \neg\sigma \models \neg a \wedge$ $S, \neg\sigma \models a$
$S_i, \neg\Sigma \models a \wedge$ $S, \neg\Sigma \models \neg a$	$S_i, \neg\Sigma \models \neg a \wedge$ $S, \neg\Sigma \models a$

where

$$\begin{aligned}
S \quad = \quad & \tau_C(\partial_H(\\
& f_1 \parallel \cdots \parallel f_n \parallel \\
& user_1 \parallel \cdots \parallel user_m \parallel \\
& OBCSM_1 \parallel \cdots \parallel TBCSM_1 \parallel \\
& \cdots\cdots \\
& OBCSM_s \parallel \cdots \parallel TBCSM_s)) \\
S_i \quad = \quad & \tau_C(\partial_H(\\
& f_i \parallel user_1 \parallel \cdots \parallel user_m \parallel \\
& OBCSM_1 \parallel \cdots \parallel TBCSM_1 \parallel \\
& \cdots\cdots \\
& OBCSM_s \parallel \cdots \parallel TBCSM_s))^{19} \\
& s \in \mathbb{N}
\end{aligned}$$

S_i is the ACP^{τ} description of a telecommunication system, which provides the service f_i only. We suppose that each single service in the set $\{ f_1, \cdots, f_n \}$ is free of property invalidation behavior, i.e. a service will never violate its own property while being executed.

[19] See Definition 4 for the meaning of τ_C and ∂_H.

9.3.3 Examples of Property Invalidation Feature Interaction

Example 7 (Interaction between OCS and ABD services) *An OCS subscriber* x *has* y *on the screening list; and* x *has also subscribed to the* abbreviated dialing *(ABD) service. If* x *dials a number* s*, the ABD service will translate* s *to the terminal number* y*, and the network will try to establish a call from* x *to* y*. Suppose that* x *picks up the hand-set and dials* s *now, the OCS service will be activated and* x *passes the authorization for* s *is not on the screening list; and then, ABD is activated, which translates* s *to* y*. Hence the call from* x *to* y *will be established if* y *is idle. In this case, the ABD service invalidates the property of the OCS service that* x *can not initiate a call to* y*.*

Figure 21 is the model of this call scenario.
The definition of the system is

$$\begin{aligned}
OCS2ABD_{x,s,y} &= \\
&\partial_{H_{OCS2ABD}}(\quad idle_x \parallel busy_y \parallel idle_z \parallel \\
&\qquad OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\
&\qquad OCS_{x,y} \parallel ABD_{x,s,y}) \\
H_{OCS2ABD} &= \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot\cdot 4]\} \\
&+ \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot\cdot 2]\} \\
&+ \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot\cdot 2]\}
\end{aligned}$$

Suppose that

$$\begin{aligned}
S &= \tau_{C_{OCS2ABD}}(OCS2ABD_{x,s,y}) \\
&\quad C_{OCS2ABD} = \{c_{c_i}(sig), c_{d_i}(sig^{'}) \mid sig \in V, sig^{'} \in I, i \in [1 \cdot\cdot 2]\} \\
\sigma &= c_{u2}(offhook(x)) \cdot c_{u1}(dial_tone(x)) \cdot c_{u1}(stop_dial_tone(x)) \cdot \\
&\quad c_{u2}(dial(x,s)) \cdot c_{u4}(useridle(y))
\end{aligned}$$

Then we have $S \xrightarrow{\sigma} S^{'} \xrightarrow{c_{u1}(alert\_user(x,y))} S^{''}$, where

$$\begin{aligned}
S^{'} &= \tau_{C_{OCS2ABD}}(\partial_{H_{OCS2CFB}}(\\
&\quad busy_x \parallel busy_y \parallel \\
&\quad Send_Call_{x,y} \parallel Present_Call_{x,y} \parallel \\
&\quad OCS_Null_{x,z} \parallel ABD_Null_{x,s,y})) \\
S^{''} &= \tau_{C_{OCS2ABD}}(\partial_{H_{OCS2CFB}}(\\
&\quad busy_x \parallel busy_y \parallel \\
&\quad O_Alerting_{x,y} \parallel T_Alerting_{x,y} \parallel \\
&\quad OCS_Null_{x,y} \parallel ABD_Null_{x,s,y}))
\end{aligned}$$

$S^{'}$ is the state where the OCS has finished call authorization and screening on the x party and the ABD service has translated s to y. $S^{''}$ is the state when the call from x to y has been established.

We have
$S \models c_{u1}(alert\_user(x,z))$
However according to the definition of OCS, we have
$S_i \models \neg c_{u1}(alert\_user(x,z))$, where

$$\begin{aligned}
S_i &= \tau_{C_{OCS2CFB}}(\partial_{H_{OCS2CFB}}(\\
&\quad OCS_{x,y} \parallel idle_x \parallel idle_y \parallel \\
&\quad O_Null_{x,y} \parallel T_Null_{x,y}))
\end{aligned}$$

An intuitive explanation of the feature interaction is that, OCS has been executed and deactivated after x picks up the hand-set and dials s. After the ABD translates s to y and the network tries to establish the call from x to y, OCS is unable to authorize the call.

Example 7 (and also Example 8) illustrates that the order to activate services is very important. If ABD can be invoked before OCS such that OCS is still able to verify the called party number translated by ABD, this interaction can be avoided. Actually, the interaction is caused by the un-precise definition of ABD and OCS which have not described explicitly the execution order of ABD and OCS. In this sense, Example 7 (and also Example 8) tells how important it is to make the service description as precise as possible, even taking into account factors such as relationship between telecommunication services.

Example 8 (Interaction between OCS and CFB services) *Suppose that an OCS subscriber* x *has* z *on his screening list (denoted as $OCS_{x,z}$); a CFB subscriber* y *forwards the incoming call to* z *when busy (denoted as $CFB_{y,z}$), and x is initiating a call to y when y is busy. According to the CFB service, the call from x to y will be forward to z. if* z *is idle,* x *will hear a back ring, thus the feature interaction occurs for the property of OCS that* x *can not initiate a call to* z *is invalidated by the CFB service.*

Figure 22 is the model of this call scenario.
The definition of this system is

$$
\begin{array}{rcl}
OCS2CFB_{x,y,z} & = & \\
 & \partial_{H_{OCS2CFB}}(& idle_x \parallel busy_y \parallel idle_z \parallel \\
 & & OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\
 & & OBCSM_{x,z} \parallel TBCSM_{x,z} \parallel \\
 & & OCS_{x,z} \parallel CFB_{y,z}) \\
H_{CFB2CFB} & = & \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot\cdot 6]\} \\
 & + & \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot\cdot 6]\} \\
 & + & \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot\cdot 4]\}
\end{array}
$$

Suppose that

$$
\begin{array}{rcl}
S & = & \tau_{C_{OCS2CFB}}(OCS2CFB_{x,y,z}) \\
 & & C_{OCS2CFB} = \{c_{c_i}(sig), c_{d_j}(sig^{'}) \mid sig \in V, sig^{'} \in I, i \in [1 \cdot\cdot 6], j \in [1 \cdot\cdot 4]\} \\
\sigma & = & c_{u2}(\mathit{offhook}(x)) \cdot c_{u1}(\mathit{dial\_tone}(x)) \cdot c_{u1}(\mathit{stop\_dial\_tone}(x)) \cdot \\
 & & c_{u2}(\mathit{dial}(x,y)) \cdot c_{u4}(\mathit{userbusy}(y)) \cdot c_{u6}(\mathit{useridle}(z))
\end{array}
$$

Then we have $S \stackrel{\sigma}{\rightarrow} S^{'} \stackrel{c_{u5}(alert\_user(x,z))}{\longrightarrow} S^{''}$, where

$$
\begin{array}{rcl}
S^{'} & = & \tau_{C_{OCS2CFB}}(\partial_{H_{OCS2CFB}}(\\
 & & busy_x \parallel busy_y \parallel idle_z \parallel \\
 & & O\_Null_{x,y} \parallel T\_Null_{x,y} \parallel \\
 & & Send\_Call_{x,z} \parallel Present\_Call_{x,z} \parallel \\
 & & OCS\_Null_{x,z} \parallel CFB\_Null_{y,z})) \\
S^{''} & = & \tau_{C_{OCS2CFB}}(\partial_{H_{OCS2CFB}}(\\
 & & busy_x \parallel busy_y \parallel idle_z \parallel \\
 & & O\_Null_{x,y} \parallel T\_Null_{x,y} \parallel \\
 & & O\_Alerting_{x,z} \parallel T\_Alerting_{x,z} \parallel \\
 & & OCS\_Null_{x,y} \parallel CFB\_Null_{y,z}))
\end{array}
$$

$S^{'}$ is the state where the OCS has finished call authorization and screening on the x party; the call from x to y has been cleared because of the called party busy event encountered; and the CFB service has finished forwarding the call to z. $S^{''}$ is the state when the call from x to z has been established.

We have

$S \models c_{u1}(alert\_user(x,z))$

However, according to the definition of OCS, we have

$S_i \models \neg c_{u1}(alert\_user(x,z))$, where

$$\begin{aligned} S_i \quad = \quad & \tau_{C_{OCS2CFB}}(\partial_{H_{OCS2CFB}}(\\ & OCS_{x,y} \parallel busy_x \parallel busy_y \parallel idle_z \parallel \\ & O\_Null_{x,y} \parallel T\_Null_{x,y} \parallel \\ & O\_Null_{x,z} \parallel T\_Null_{x,z})) \end{aligned}$$

An intuitive explanation of the feature interaction is that, OCS has been executed and deactivated after x picks up the hand-set and dials y. When y forwards the call to z, OCS is unable to authorize and check the call.

Example 9 (Interaction between TCS and AR) *The* terminating call screen *(TCS) service blocks all the incoming call from the calling parties in the screening list. Suppose that* y *has subscribed to both the TCS and AR services,*[20] *and* x *is in the screening list (denoted as* $TCS_{y,x}$ *and* AR_y *respectively. Now x initiates a call to* y*. The call is refused by the TCS service* y *subscribing to. However, AR is also activated when the network detects there is an incoming call for* y*. The execution of the AR service causes the network to initiate a call from* y *to* x*, and establish the voice connection between* y *and* x*. The paradox is that, such voice connection is established without* y *party's obvious intention and action to initiate a call to* x *and the property of the TCS is nullified.*

Figure 23 is the model of this call scenario.

The definition of this system is

$$\begin{aligned} TCS2AR_{x,y} \quad = \quad & \\ & \partial_{H_{TCS2AR}}(\quad idle_x \parallel busy_y \parallel \\ & \qquad OBCSM_{x,y} \parallel TBCSM_{x,y} \parallel \\ & \qquad OBCSM_{y,x} \parallel TBCSM_{y,x} \parallel \\ & \qquad TCS_{y,x} \parallel AR_y) \\ H_{TCS2AR} \quad = \quad & \{s_{u_i}(sig), r_{u_i}(sig) \mid sig \in U, i \in [1 \cdot \cdot 4]\} \\ + \quad & \{s_{c_i}(sig), r_{c_i}(sig) \mid sig \in V, i \in [1 \cdot \cdot 4]\} \\ + \quad & \{s_{d_i}(sig), r_{d_i}(sig) \mid sig \in I, i \in [1 \cdot \cdot 2]\} \end{aligned}$$

Suppose that

$$\begin{aligned} S \quad = \quad & \tau_{C_{TCS2AR}}(TCS2AR_{x,y}) \\ & C_{TCS2AR} = \{c_{c_i}(sig), c_{d_j}(sig^{'}) \mid sig \in V, sig^{'} \in I, i \in [1 \cdot \cdot 4], j \in [1 \cdot \cdot 2]\} \\ \sigma \quad = \quad & c_{u2}(offhook(x)) \cdot c_{u1}(dial\_tone(x)) \cdot c_{u1}(stop\_dial\_tone(x)) \cdot c_{u2}(dial(x,y)) \cdot \\ & c_{u2}(onhook(x)) \cdot c_{u3}(special\_tone(y)) \cdot c_{u4}(offhook(y)) \end{aligned}$$

[20] See Example 6 for the service description of AR

Then we have $S \xrightarrow{\sigma} S' \xrightarrow{c_{u3}(alert\_user(y,x))} S''$, where

$$\begin{aligned}
S' &= \tau_{C_{TCS2AR}}(\partial_{H_{TCS2AR}}(\\
&\quad idle_x \parallel busy_y \parallel \\
&\quad O_Null_{x,y} \parallel T_Null_{x,y} \parallel \\
&\quad Send_Call_{y,x} \parallel Present_Call_{y,x} \parallel \\
&\quad TCS_Null_{y,x} \parallel AR_Null_y)) \\
S'' &= \tau_{C_{TCS2AR}}(\partial_{H_{TCS2AR}}(\\
&\quad idle_x \parallel busy_y \parallel \\
&\quad O_Null_{x,y} \parallel T_Null_{x,y} \parallel \\
&\quad O_Alerting_{y,x} \parallel T_Alerting_{y,x} \parallel \\
&\quad OCS_Null_{x,y} \parallel CFB_Null_{y,z}))
\end{aligned}$$

S' is the state where the TCS has finished call authorization and screening on the x party and refused the call from x to y; and the AR service has finished initiating the call from y to x. S'' is the state when the call from y to x has been established.

We have

$S, \neg\Sigma_y \models c_{u3}(alert\_user(y,x))$

However, according to the definition of TCS, we have

$S_i, \neg\Sigma_y \models \neg c_{u3}(alert\_user(y,x))$, where

$$\begin{aligned}
S_i &= \tau_{C_{OCS2CFB}}(\partial_{H_{OCS2CFB}}(\\
&\quad TCS_{x,y} \parallel busy_x \parallel busy_y \parallel idle_z \parallel \\
&\quad O_Null_{x,y} \parallel T_Null_{x,y} \parallel \\
&\quad O_Alerting_{x,z} \parallel T_Alerting_{x,z})) \\
\Sigma_y &= \{dial(y,z) \mid z \in ADDR\}
\end{aligned}$$

An intuitive explanation of the feature interaction is that, TCS has been executed and deactivated after the network detects there is an incoming call to y. When the AR service initiates a call from y to x, TCS is unable to authorize and check the call.

9.4 Conclusion on Feature Interaction Detection

The basic principle in the feature interaction detection is to find inconsistent behavior in the telecommunication system when several communication services or several instances of the same service are executed in parallel. In [22], P.Combes describes the method in a more formal way as the following:

Let $f_1, f_2, \cdots$ resp. N be the formal specification of features, resp. the basic telecommunication network. Let $N \oplus f_i$ denote the specification of the network obtained by adding service feature f_i to the basic network N. Let $P_1, P_2, \cdots$ be formulae expressing feature requirements in a suitable property language and let $N \models P$ denote that the network specification N satisfies P.

There is interaction between features $f_1, \cdots, f_n$ if

$N \oplus f_i \models P_i, 1 \leq i \leq n$, but

$N \oplus f_1 \oplus \cdots \oplus f_n \not\models P_1 \wedge \cdots \wedge P_n$

This method depends heavily on careful and, if possible, complete formalization of expectations.

In our modeling, the network behavior is represented as the parallel communicating processes such as users and BCSMs. Each service is represented as independent communicating process too. The interworking between network and features ($\oplus$) are modeled by the parallel combination operator in ACP$^\tau$ ($\parallel$). We expect the telecommunication system to be free from non-deterministic

behavior, livelock and property invalidation behavior when different services or different instances of the same service are executed in parallel. Otherwise we declare that feature interactions occur. The principle is stated more precisely as the following:

1. Non-determinacy feature interactions occur if

   $N \parallel f_i \models$ *deterministic behavior*, $1 \leq i \leq n$, but

   $N \parallel f_1 \parallel \cdots \parallel f_n \models$ *non-deterministic behavior*

2. Livelock feature interactions occur if

   $N \parallel f_i \models$ *livelock free*, $1 \leq i \leq n$, but

   $N \parallel f_1 \parallel \cdots \parallel f_n \models$ *livelock*

3. Property invalidation feature interactions occur if

   $N \parallel f_i \models A, 1 \leq i \leq n$, but

   $N \parallel f_1 \parallel \cdots \parallel f_n \models \neg A$

   A is the formal specification of certain properties in the telecommunication system.

10 Conclusion

This paper has presented a system model suitable for specifying and validating the Intelligent Network services. Different network entities such as users, IN services and switching processes are represented by different processes. Process interworking is described by communications between their parallel composition.

The central idea of our modeling is based on the basic call state model from ITU-T Q.12x4 recommendations. Switching behavior is modeled by an independent process from IN services, based on the concept of BCSM in Q.1224. IN services provides supplementary functions to the switching processes; they can receive and send INAP messages to control the state transition procedure in the BCSM process. System properties are described by their communication behaviors. By doing so, we make our model consistent with the key idea of IN, to separate call control from service control.

In order to validate the system, we formalize the IN system in ACP$^\tau$ and implement the formal specifications in PSF, which leads to the following results,

1. Simulation of IN services in *PSF Toolkit*. By checking whether the simulation result is consistent with the service definition at the user level, it possible to find potential errors in the original service specification and correct them. In this way, we make our formal specification of IN services reliable, which we believe better than those not supported by simulation tools. In Appendix C, specification examples of services such as TWC, CW, CFB, CCBS, AR are given.

2. Tracing of the parallel processes in the system as showed in Figure 10, Figure 11 and Figure 18, etc.

3. Detection of feature interactions. Interactions can be detected by analyzing the execution traces of parallel processes. In Section 5.2, Section 5.3 and Section 5.4 we give examples on the detection of feature interactions, which can be found with the help of PSF Toolkit.

By building a structured call and service processing model based on BCSM, we believe this paper serves a better understanding of the call processing mechanism and the feature interaction problem in the telecommunication system.

References

[1] Q.122x. *ITU-T CS-2 recommendation.*

[2] ETSR NA. *Intelligent Network(IN); Service and Service Feature Interaction Service Creation, Service Management and Service Execution Aspects, ETSI draft ETR 137, July 1995.*

[3] ETSR NA. *Intelligent Network(IN); Interaction between INAP and ISDN signaling protocols; Part 1: Switching Signaling Requirements for IN CS1 Service Support in N-ISDN environment, ETSI draft ETR 322, September 1995.*

[4] ETSR NA. *Intelligent Network(IN); Service Life Cycle Reference Model for Services Supported by an IN, ETSI draft ETR 322, December 1996.*

[5] J.A.Bergstra and J.W.Klop. *An Introduction to Process Algebra. Applications of Process Algebra. Cambridge Tracts in Theoretical Computer Science 17.*

[6] J.C.M.Baeten and W.P.Weijland. *Process Algebra. Cambridge Tracts in Theoretical Computer Science 18.*

[7] J.A.Bergstra, C.A.Middelburg and Y.S.Usenko. *Discrete Time Process Algebra and the Semantics of SDL. Report SEN-R9809, Centrum voor Wiskunde en Informatica (CWI), June 1998, the Netherlands.*

[8] R.J.van Glabbeek *Bounded Nondeterminism and the Approximation Induction Principle in Process Algebra. Report CS-R8634, Centrum voor Wiskunde en Informatica (CWI), 1986, the Netherlands.*

[9] G.J.Veltink. *Tools for PSF. ILLC Dissertation Series 1995-9.*

[10] Sjouke Mauw. *PSF - A Process Specification Formalism. Universiteit van Amsterdam, 1991.*

[11] J.F.Groote. *Specification and Verification of Real Time Systems in ACP. Tenth International IFIP WG6.1 Symposium on Protocol Specification, Testing and Verification, pages 259-270, Ottawa, June 1990, Canada.*

[12] F.S.Dworak, T.F.Bowen, G.E.Herman, N.Griffeth and Y.J.Lin. *In Proceeding of the Seventh International Conference on Software Engineering for Telecommunication Switching Systems. July 1989, Bournemouth, UK.*

[13] *Introduction. Feature interactions in Telecommunications Systems, IOS Press 1994.*

[14] 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. Feature Interactions in Telecommunications Systems, IOS Press 1994.*

[15] K.H.Braithwaite and J.M.Atlee. *Towards Automated Detection of Feature Interactions. Feature Interactions in Telecommunications Systems, IOS Press 1994.*

[16] T.Ohta and Y.Harada. *Classification, Detection and Resolution of Service Interactions in Telecommunication Services. Feature Interactions in Telecommunications Systems, IOS Press 1994.*

[17] F.J.Lin and Y.J.Lin. *A Building Block Approach to Detecting and Resolving Feature Interactions. Feature Interactions in Telecommunications Systems, IOS Press 1994.*

[18] J.Blom, B.Josson and L.Kempe. *Using Temporal Logic for Modular Specification of Telephone Services. Feature Interactions in Telecommunications Systems, IOS Press 1994.*

[19] S.Tsang and E.H.Magill. *Detecting Feature Interaction in the Intelligent Network. Feature Interactions in Telecommunications Systems, IOS Press 1994.*

[20] Mohammed Faci, Liugi Logrippo and Bernard Stepien. *Structural Models for specifying telephone systems. Computer Networks and ISDN systems 29(1997) 501-508.*

[21] k.H.Braithwaite and J.M.Atlee *Towards Automated Detection of Feature Interactions. Feature Interactions in Telecommunications Systems, IOS Press 1994.*

[22] P.Combes and S.Pickin *Formalization of a User View of Network and Services for Feature Interaction Detection. Feature Interactions in Telecommunications Systems, IOS Press 1994.*

[23] E.Magill, S.Tsang and B.Kelly *Feature Interactions in Multimedia Systems: Some Potential Areas for Feature Interactions in Multimedia Services. http://www.comms.eee.strath.ac.uk/ fi/fimna.html, Apirl 1996.*

[24] E.Magill, S.Tsang and B.Kelly *The Feature Interaction Problem in Networked Multimedia Services: Past, Present and Future. http://www.comms.eee.strath.ac.uk/ fi/fimna.html, October 1996.*

[25] Nancy.Griffeth *http://www-db.research.bell-labs.com/user/nancyg/Appendix.ps*

Appendix A: PSF Data Modules

---

data module *Networkdatas*
begin
exports
begin
sorts
ADDRESS, *ID*
functions
adr0 : → *ADDRESS*
adr1 : → *ADDRESS*
adr2 : → *ADDRESS*
adr3 : → *ADDRESS*
adr4 : → *ADDRESS*
eq : *ADDRESS* # *ADDRESS* → *BOOLEAN*
map : *ADDRESS* → *NATURAL*
0 : → *ID*
1 : → *ID*
2 : → *ID*
3 : → *ID*
4 : → *ID*
5 : → *ID*
eq : *ID* # *ID* → *BOOLEAN*
map : *ID* → *NATURAL*
end
imports
Naturals, *Booleans*
variables
sn1, *sn2* : → *ADDRESS*
id1, *id2* : → *ID*
equations
[1] $map(adr0) = zero$
[2] $map(adr1) = s(zero)$
[3] $map(adr2) = s(s(zero))$
[4] $map(adr3) = s(s(s(zero)))$
[5] $map(adr4) = s(s(s(s(zero))))$
[6] $eq(sn1, sn2) = eq(map(sn1), map(sn2))$
[7] $map(0) = zero$
[8] $map(1) = s(zero)$
[9] $map(2) = s(s(zero))$
[10] $map(3) = s(s(s(zero)))$
[11] $map(4) = s(s(s(s(zero))))$
[12] $map(5) = s(s(s(s(s(zero)))))$
[13] $eq(id1, id2) = eq(map(id1), map(id2))$
end *Networkdatas*

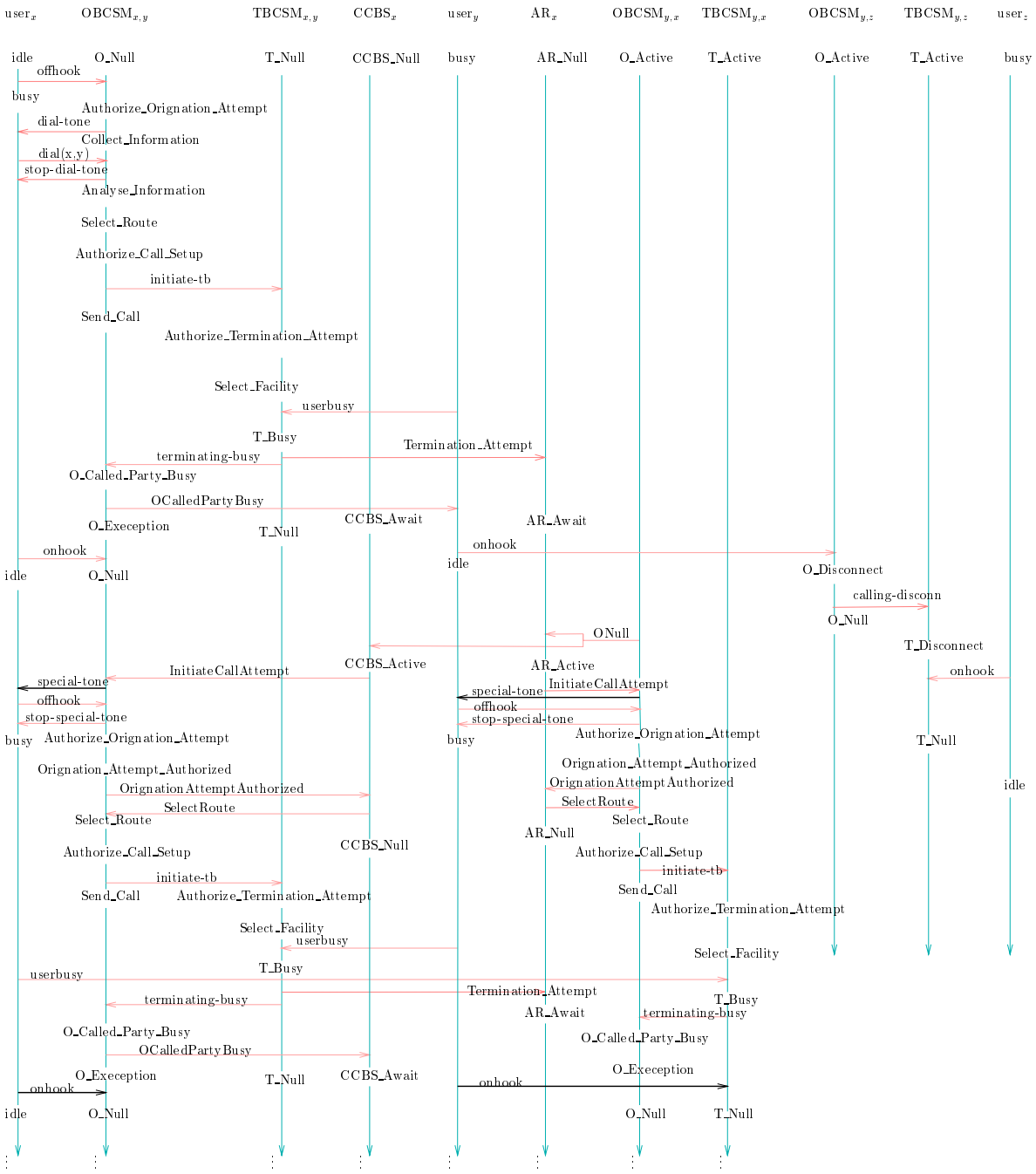

Figure 20: Live-locks in the Parallel Execution of AR and CCBS services

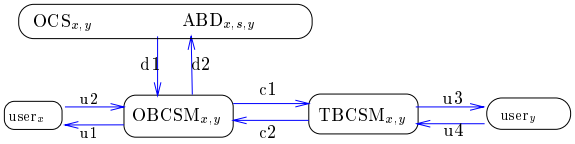

Figure 21: Modeling of the Parallel Execution of OCS and ABD

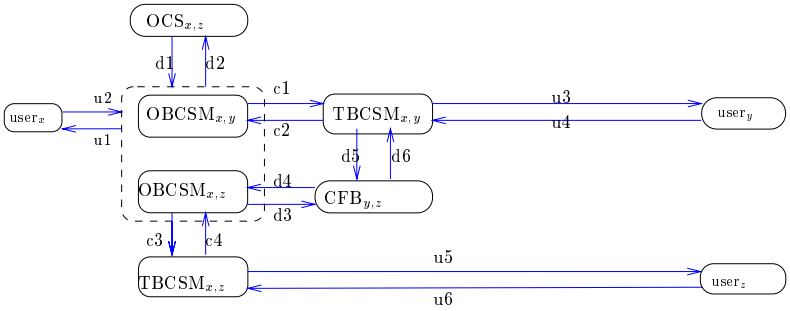

Figure 22: Modeling of the Parallel Execution of OCS and CFB

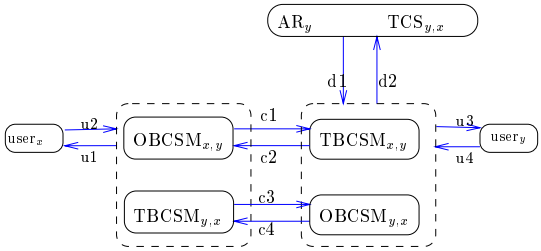

Figure 23: Modeling of the Parallel Execution of TCS and AR

data module *Intrasigs*
−−In this module,
−−we define the signals between OBCSM and TBCSM to synchronize
−−their behavior. The definition is based on
−−ITU−T Q.1224, Intra Local Exchange BCSM Indications
begin
exports
begin
sorts
INTRASIG, *BCSMCVSSIG*
functions
initiate-tb : *ID* # *ADDRESS* → *INTRASIG*−− From OB to TB, (1)
terminating-busy : *ID* → *INTRASIG*−− From TB to OB, (3),(4)
non-presentation : *ID* → *INTRASIG*−− From TB to OB, (5)
call-acceptance : *ID* → *INTRASIG*−− From TB to OB, (6),(11),(12)
called-alerting : *ID* → *INTRASIG*−− From TB to OB, (7)
called-rejected : *ID* → *INTRASIG*−− From TB to OB, (8)
called-noanswer : *ID* → *INTRASIG*−− From TB to OB, (9),(10)
called-suspend : *ID* → *INTRASIG*−− From TB to OB, (13)
called-reanswer : *ID* → *INTRASIG*−− From TB to OB, (14)
calling-disconn : *ID* → *INTRASIG*−− From OB to TB, (15),(16)
called-disconn : *ID* → *INTRASIG*−− From TB to OB, (17),(18)
calling-abandon : *ID* → *INTRASIG*−− From OB to TB, (19)
end
imports
Naturals, *Networkdatas*

end *Intrasigs*

data module *Inapsigs*
begin
exports
begin
sorts
INAPSIG, *MODE*
functions
inap-adi : $ID \rightarrow INAPSIG$
inap-at : $ID \rightarrow INAPSIG$
inap-ai : $ID \# ADDRESS \rightarrow INAPSIG$
inap-ai : $ID \rightarrow INAPSIG$
inap-ci : $ID \rightarrow INAPSIG$
inap-conn : $ID \# ADDRESS \# ADDRESS \rightarrow INAPSIG$
inap-conn : $ID \rightarrow INAPSIG$
inap-cont : $ID \rightarrow INAPSIG$
inap-cwa : $ID \rightarrow INAPSIG$
inap-dl : $ID \# ADDRESS \rightarrow INAPSIG$
inap-ica : $ID \# ADDRESS \rightarrow INAPSIG$
inap-oc : $ID \# ADDRESS \rightarrow INAPSIG$
inap-rc : $ID \# ADDRESS \rightarrow INAPSIG$
inap-rec : $ID \rightarrow INAPSIG$
inap-sr : $ID \# ADDRESS \# ADDRESS \rightarrow INAPSIG$
inap-sr : $ID \rightarrow INAPSIG$
inap-ccsa : $ID \rightarrow INAPSIG$
inap-mgc : $ID \# ADDRESS \rightarrow INAPSIG$
inap-mvc : $ID \rightarrow INAPSIG$
inap-ml : $ID \# ADDRESS \rightarrow INAPSIG$
inap-sf : $ID \rightarrow INAPSIG$
inap-sl : $ID \# ADDRESS \rightarrow INAPSIG$

−−The following is the definition of DP report messages −−
inap-Analysed : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-OAttempt : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-OAuthorized : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-Collected : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-OAnswer : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-OSuspend : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-ODisc : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-OMid : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-OBusy : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-ONoAnswer : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-OAbandon : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-RouteFail : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-OSuspend : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-ONull : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TAttempt : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TAuthorized : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-FacilAvail : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TMid : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TNoAnswer : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$

inap-TBusy : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TAbandon : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TDisc : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-CAccept : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TAnswer : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TReAnswer : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TSuspend : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
inap-TNull : $MODE \# ID \# ADDRESS \rightarrow INAPSIG$
req : $\rightarrow MODE$
ind : $\rightarrow MODE$

end
imports
Networkdatas

end *Inapsigs*

---

data module *Usersigs*
begin
exports
begin
sorts
USERSIG
functions
offhook : $ADDRESS \rightarrow USERSIG$
onhook : $ADDRESS \rightarrow USERSIG$
flashhook : $ADDRESS \rightarrow USERSIG$
noanswer : $ADDRESS \rightarrow USERSIG$
special-tone : $ADDRESS \rightarrow USERSIG$
userbusy : $ADDRESS \rightarrow USERSIG$
useridle : $ADDRESS \rightarrow USERSIG$
dial : $ADDRESS \# ADDRESS \rightarrow USERSIG$
dial-tone : $ADDRESS \rightarrow USERSIG$
busy-tone : $ADDRESS \rightarrow USERSIG$
back-ring : $ADDRESS \rightarrow USERSIG$
alert-user : $ADDRESS \rightarrow USERSIG$
CWtone : $ADDRESS \rightarrow USERSIG$
stop-back-ring : $ADDRESS \rightarrow USERSIG$
stop-dial-tone : $ADDRESS \rightarrow USERSIG$
stop-CWtone : $ADDRESS \rightarrow USERSIG$
stop-alert-user : $ADDRESS \rightarrow USERSIG$
stop-special-tone : $ADDRESS \rightarrow USERSIG$
end
imports
Networkdatas

end *Usersigs*

Appendix B: PSF Process Modules: User, OBCSM and TBCSM

---

process module *User*
−−This is the PSF description of the behavior of the phone user
begin
exports
begin
atoms
s : $USERSIG$
r : $USERSIG$
processes
$idle$: $ADDRESS$
$busy$: $ADDRESS$
end
imports
Usersigs, *Networkdatas*
variables
x : $\rightarrow ADDRESS$
definitions
$idle(x) = s(offhook(x))$
$\cdot\ busy(x)$
$+\ s(useridle(x))$
$\cdot\ idle(x)$
$+\ r(alert\text{-}user(x))$
$\cdot\ idle(x)$
$+\ r(special\text{-}tone(x))$
$\cdot\ idle(x)$
$+\ s(noanswer(x))$
$\cdot\ idle(x)$
$busy(x) = s(onhook(x))$
$\cdot\ idle(x)$
$+\ s(flashhook(x))$
$\cdot\ busy(x)$
$+\ s(userbusy(x))$
$\cdot\ busy(x)$
$+\ r(dial\text{-}tone(x))$
$\cdot$ **sum**(y **in** $ADDRESS$,
$s(dial(x,\ y))$
$\cdot\ busy(x)$
)
$+\ r(CWtone(x))$
$\cdot$ (
$s(flashhook(x))$
$+\ s(noanswer(x))$
)
$\cdot\ busy(x)$
$+\ r(busy\text{-}tone(x))$
$\cdot\ busy(x)$
$+\ r(back\text{-}ring(x))$
$\cdot\ busy(x)$
$+\ r(stop\text{-}back\text{-}ring(x))$

$\cdot$ $busy(x)$
$+ r(\textit{stop-dial-tone}(x))$
$\cdot$ $busy(x)$
$+ r(\textit{stop-alert-user}(x))$
$\cdot$ $busy(x)$
$+ r(\textit{stop-special-tone}(x))$
$\cdot$ $busy(x)$

end *User*

---

process module *OBCSM*
−−This is the PSF describtion of OBCSM in CS2 −−
−−
−−OBCSM needs to communicate with the calling party, TBCSM, and IN, with the
−−signals from signal set USERSIG, INTRASIG and INAPSIG, respectively.
−−
−−In this original OBCSM model, the communication directly between
−−the called party and the OBCSM is forbidden,
−−i.e. such communication must be compeleted via TBCSM.
begin
exports
begin
atoms
s : *INTRASIG*
s : *INAPSIG*
s : *USERSIG*
s : *BCSMCVSSIG*
r : *INTRASIG*
r : *INAPSIG*
r : *USERSIG*
origination-denied collect-timeout invalid-information author-route-failure
o-active-failure o-suspend-failure route-selected route-busy route-failure
oreconnect orig-auth-success addr-available unable-to-sel-route
call-setup-authorized term-seized-event
processes
obcsm : *ID # ADDRESS # ADDRESS*
onull : *ID # ADDRESS # ADDRESS*
aoa : *ID # ADDRESS # ADDRESS*
ci : *ID # ADDRESS # ADDRESS*
ai : *ID # ADDRESS # ADDRESS*
sr : *ID # ADDRESS # ADDRESS*
acs : *ID # ADDRESS # ADDRESS*
sc : *ID # ADDRESS # ADDRESS*
oal : *ID # ADDRESS # ADDRESS*
oac : *ID # ADDRESS # ADDRESS*
os : *ID # ADDRESS # ADDRESS*
oe : *ID # ADDRESS # ADDRESS*
dp-oat : *ID # ADDRESS # ADDRESS*
dp-oaa : *ID # ADDRESS # ADDRESS*
dp-ci : *ID # ADDRESS # ADDRESS*
dp-ai : *ID # ADDRESS # ADDRESS*
dp-ots : *ID # ADDRESS # ADDRESS*
dp-oan : *ID # ADDRESS # ADDRESS*
dp-os : *ID # ADDRESS # ADDRESS*
dp-od : *ID # ADDRESS # ADDRESS*
dp-or : *ID # ADDRESS # ADDRESS*
dp-omc-sc : *ID # ADDRESS # ADDRESS*
dp-omc-oal : *ID # ADDRESS # ADDRESS*
dp-omc-oac : *ID # ADDRESS # ADDRESS*
dp-omc-os : *ID # ADDRESS # ADDRESS*

dp-oab : $ID \# ADDRESS \# ADDRESS$
dp-rsf : $ID \# ADDRESS \# ADDRESS$
dp-ocpb : $ID \# ADDRESS \# ADDRESS$
dp-ona : $ID \# ADDRESS \# ADDRESS$

end

imports

Usersigs, *Inapsigs*, *Networkdatas*, *Intrasigs*

sets

of atoms

swto-lock = { *origination-denied*, *collect-timeout*, *invalid-information*, *author-route-failure*, *o-active-failure*, *o-suspend-failure*, *route-busy*, *route-failure*, *unable-to-sel-route* }

swto-skip = { *route-selected*, *oreconnect*, *orig-auth-success*, *addr-available*, *call-setup-authorized*, *term-seized-event* }

variables

callid : $\rightarrow ID$

x, *y*, *z*, *z0*, *z1* : $\rightarrow ADDRESS$

definitions

$obcsm(callid, x, y) =$ **hide**(*swto-skip*, **encaps**(*swto-lock*, $onull(callid, x, y)$))

$onull(callid, x, y) = r(offhook(x))$
$\cdot\ dp\text{-}oat(callid, x, y)$
$+ r(inap\text{-}oc(callid, x))$
$\cdot\ dp\text{-}oat(callid, x, y)$
$+ r(inap\text{-}ica(callid, x))$
$\cdot\ s(special\text{-}tone(x))$
$\cdot\ r(offhook(x))$
$\cdot\ s(stop\text{-}special\text{-}tone(x))$
$\cdot\ dp\text{-}oat(callid, x, y)$
$+ s(inap\text{-}ONull(ind, callid, x))$
$\cdot\ onull(callid, x, y)$

$aoa(callid, x, y) = r(onhook(x))$
$\cdot\ dp\text{-}oab(callid, x, y)$
$+ origination\text{-}denied$
$\cdot\ oe(callid, x, y)$
$+ orig\text{-}auth\text{-}success$
$\cdot\ dp\text{-}oaa(callid, x, y)$

$ci(callid, x, y) = r(onhook(x))$
$\cdot\ dp\text{-}oab(callid, x, y)$
$+ s(dial\text{-}tone(x))$
$\cdot\ r(dial(x, y))$
$\cdot\ s(stop\text{-}dial\text{-}tone(x))$
$\cdot\ dp\text{-}ci(callid, x, y)$
$+ collect\text{-}timeout$
$\cdot\ oe(callid, x, y)$

$ai(callid, x, y) = r(onhook(x))$
$\cdot\ dp\text{-}oab(callid, x, y)$
$+ invalid\text{-}information$
$\cdot\ oe(callid, x, y)$

$$
\begin{aligned}
&\quad + \mathit{addr\text{-}available} \\
&\quad \cdot \ \mathit{dp\text{-}ai}(\mathit{callid},\ x,\ y) \\
\mathit{sr}(\mathit{callid},\ x,\ y) &= r(\mathit{onhook}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}oab}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{unable\text{-}to\text{-}sel\text{-}route} \\
&\quad \cdot \ \mathit{dp\text{-}rsf}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{route\text{-}selected} \\
&\quad \cdot \ \mathit{acs}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{route\text{-}busy} \\
&\quad \cdot \ \mathit{ai}(\mathit{callid},\ x,\ y) \\
\mathit{acs}(\mathit{callid},\ x,\ y) &= r(\mathit{onhook}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}oab}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{author\text{-}route\text{-}failure} \\
&\quad \cdot \ \mathit{oe}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{call\text{-}setup\text{-}authorized} \\
&\quad \cdot \ s(\mathit{initiate\text{-}tb}(\mathit{callid},\ y)) \\
&\quad \cdot \ \mathit{sc}(\mathit{callid},\ x,\ y) \\
\mathit{sc}(\mathit{callid},\ x,\ y) &= r(\mathit{onhook}(x)) \\
&\quad \cdot \ s(\mathit{calling\text{-}abandon}(\mathit{callid})) \\
&\quad \cdot \ \mathit{dp\text{-}oab}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{call\text{-}acceptance}(\mathit{callid})) \\
&\quad \cdot \ \mathit{dp\text{-}oan}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{terminating\text{-}busy}(\mathit{callid})) \\
&\quad \cdot \ s(\mathit{busy\text{-}tone}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}ocpb}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{called\text{-}alerting}(\mathit{callid})) \\
&\quad \cdot \ s(\mathit{back\text{-}ring}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}ots}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{flashhook}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}omc\text{-}sc}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{route\text{-}failure} \\
&\quad \cdot \ \mathit{sr}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{author\text{-}route\text{-}failure} \\
&\quad \cdot \ \mathit{oe}(\mathit{callid},\ x,\ y) \\
\mathit{oal}(\mathit{callid},\ x,\ y) &= r(\mathit{onhook}(x)) \\
&\quad \cdot \ s(\mathit{calling\text{-}abandon}(\mathit{callid})) \\
&\quad \cdot \ \mathit{dp\text{-}oab}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{call\text{-}acceptance}(\mathit{callid})) \\
&\quad \cdot \ s(\mathit{stop\text{-}back\text{-}ring}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}oan}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{called\text{-}noanswer}(\mathit{callid})) \\
&\quad \cdot \ \mathit{dp\text{-}ona}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{flashhook}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}omc\text{-}oal}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{route\text{-}failure} \\
&\quad \cdot \ \mathit{sr}(\mathit{callid},\ x,\ y) \\
\mathit{oac}(\mathit{callid},\ x,\ y) &= r(\mathit{onhook}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}od}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{called\text{-}suspend}(\mathit{callid})) \\
&\quad \cdot \ \mathit{dp\text{-}os}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{flashhook}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}omc\text{-}oac}(\mathit{callid},\ x,\ y)
\end{aligned}
$$

$$
\begin{aligned}
&\quad + \mathit{o\text{-}active\text{-}failure} \\
&\quad \cdot \ \mathit{oe}(\mathit{callid},\ x,\ y) \\
\mathit{os}(\mathit{callid},\ x,\ y) &= r(\mathit{onhook}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}od}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{called\text{-}disconn}(\mathit{callid})) \\
&\quad \cdot \ \mathit{dp\text{-}od}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{called\text{-}reanswer}(\mathit{callid})) \\
&\quad \cdot \ \mathit{oreconnect} \\
&\quad \cdot \ \mathit{dp\text{-}or}(\mathit{callid},\ x,\ y) \\
&\quad + r(\mathit{flashhook}(x)) \\
&\quad \cdot \ \mathit{dp\text{-}omc\text{-}os}(\mathit{callid},\ x,\ y) \\
&\quad + \mathit{o\text{-}suspend\text{-}failure} \\
&\quad \cdot \ \mathit{oe}(\mathit{callid},\ x,\ y) \\
\mathit{oe}(\mathit{callid},\ x,\ y) &= r(\mathit{onhook}(x)) \\
&\quad \cdot \ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}oat}(\mathit{callid},\ x,\ y) &= s(\mathit{inap\text{-}OAttempt}(\mathit{req},\ \mathit{callid},\ x)) \\
&\quad \cdot \ (\\
&\qquad\quad r(\mathit{inap\text{-}ci}(\mathit{callid})) \\
&\qquad \cdot \ \mathit{ci}(\mathit{callid},\ x,\ y) \\
&\qquad + \mathbf{sum}(z \ \mathbf{in} \ \mathit{ADDRESS}, \\
&\qquad\qquad\quad r(\mathit{inap\text{-}conn}(\mathit{callid},\ x,\ z)) \\
&\qquad\qquad \cdot \ (\\
&\qquad\qquad\qquad\quad [\mathit{eq}(z,\ \mathit{adr0}) = \mathit{true}] \rightarrow \\
&\qquad\qquad\qquad\qquad \mathit{ci}(\mathit{callid},\ x,\ y) \\
&\qquad\qquad\qquad + [\mathit{eq}(z,\ \mathit{adr0}) = \mathit{false}] \rightarrow \\
&\qquad\qquad\qquad\qquad \mathit{ai}(\mathit{callid},\ x,\ z) \\
&\qquad\qquad) \\
&\qquad) \\
&\qquad + (\\
&\qquad\qquad\quad r(\mathit{inap\text{-}cont}(\mathit{callid})) \\
&\qquad\qquad + r(\mathit{inap\text{-}cwa}(\mathit{callid})) \\
&\qquad) \\
&\qquad \cdot \ \mathit{aoa}(\mathit{callid},\ x,\ y) \\
&\qquad + r(\mathit{inap\text{-}rc}(\mathit{callid},\ x)) \\
&\qquad \cdot \ \mathit{onull}(\mathit{callid},\ x,\ y) \\
&\quad) \\
&\quad + (\\
&\qquad\quad s(\mathit{inap\text{-}OAttempt}(\mathit{ind},\ \mathit{callid},\ x)) \\
&\qquad + \mathbf{skip}) \\
&\quad \cdot \ \mathit{aoa}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}oaa}(\mathit{callid},\ x,\ y) &= s(\mathit{inap\text{-}OAuthorized}(\mathit{req},\ \mathit{callid},\ x)) \\
&\quad \cdot \ (\\
&\qquad\quad \mathbf{sum}(\mathit{z0} \ \mathbf{in} \ \mathit{ADDRESS}, \\
&\qquad\qquad\quad r(\mathit{inap\text{-}ai}(\mathit{callid},\ \mathit{z0})) \\
&\qquad\qquad \cdot \ \mathit{ai}(\mathit{callid},\ x,\ \mathit{z0}) \\
&\qquad) \\
&\qquad + r(\mathit{inap\text{-}ci}(\mathit{callid})) \\
&\qquad \cdot \ \mathit{ci}(\mathit{callid},\ x,\ y) \\
&\qquad + \mathbf{sum}(z \ \mathbf{in} \ \mathit{ADDRESS}, \\
&\qquad\qquad\quad r(\mathit{inap\text{-}conn}(\mathit{callid},\ x,\ z)) \\
&\qquad\qquad \cdot \ (\\
&\qquad\qquad\qquad\quad [\mathit{eq}(z,\ \mathit{adr0}) = \mathit{true}] \rightarrow
\end{aligned}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ci(callid,\ x,\ y) \\
\qquad\qquad\qquad\qquad\qquad\qquad + [eq(z,\ adr0) = false] \rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ai(callid,\ x,\ z) \\
\qquad\qquad\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad + (\\
\qquad\qquad\qquad\qquad\qquad\qquad r(inap\text{-}cont(callid)) \\
\qquad\qquad\qquad\qquad\qquad + r(inap\text{-}cwa(callid)) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\cdot\ ci(callid,\ x,\ y) \\
\qquad\qquad\qquad + r(inap\text{-}rc(callid,\ x)) \\
\qquad\qquad\qquad\cdot\ onull(callid,\ x,\ y) \\
\qquad\qquad\qquad + \mathbf{sum}(z1\ \mathbf{in}\ ADDRESS, \\
\qquad\qquad\qquad\qquad\qquad\qquad r(inap\text{-}sr(callid,\ x,\ z1)) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ sr(callid,\ x,\ z1) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad) \\
\qquad\qquad + (\\
\qquad\qquad\qquad\qquad s(inap\text{-}OAuthorized(ind,\ callid,\ x)) \\
\qquad\qquad\qquad\qquad + \mathbf{skip}) \\
\qquad\qquad\cdot\ ci(callid,\ x,\ y) \\
dp\text{-}ci(callid,\ x,\ y) = s(inap\text{-}Collected(req,\ callid,\ x)) \\
\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad \mathbf{sum}(z0\ \mathbf{in}\ ADDRESS, \\
\qquad\qquad\qquad\qquad\qquad\qquad r(inap\text{-}ai(callid,\ z0)) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ ai(callid,\ x,\ z0) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad + r(inap\text{-}ci(callid)) \\
\qquad\qquad\qquad\qquad\cdot\ ci(callid,\ x,\ y) \\
\qquad\qquad\qquad\qquad + \mathbf{sum}(z\ \mathbf{in}\ ADDRESS, \\
\qquad\qquad\qquad\qquad\qquad\qquad r(inap\text{-}conn(callid,\ x,\ z)) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad [eq(z,\ adr0) = true] \rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ci(callid,\ x,\ y) \\
\qquad\qquad\qquad\qquad\qquad\qquad + [eq(z,\ adr0) = false] \rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ai(callid,\ x,\ z) \\
\qquad\qquad\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad + (\\
\qquad\qquad\qquad\qquad\qquad\qquad r(inap\text{-}cont(callid)) \\
\qquad\qquad\qquad\qquad\qquad + r(inap\text{-}cwa(callid)) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad\cdot\ ai(callid,\ x,\ y) \\
\qquad\qquad\qquad\qquad + r(inap\text{-}rc(callid,\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ onull(callid,\ x,\ y) \\
\qquad\qquad\qquad\qquad + \mathbf{sum}(z1\ \mathbf{in}\ ADDRESS, \\
\qquad\qquad\qquad\qquad\qquad\qquad r(inap\text{-}sr(callid,\ x,\ z1)) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ sr(callid,\ x,\ z1) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad) \\
\qquad\qquad + (\\
\qquad\qquad\qquad\qquad s(inap\text{-}Collected(ind,\ callid,\ x))
\end{array}
$$

$$
\begin{aligned}
&\quad + \mathbf{skip})\\
&\quad \cdot\ ai(callid,\ x,\ y)\\
dp\text{-}ai(callid,\ x,\ y) &= s(inap\text{-}Analysed(req,\ callid,\ x))\\
&\quad \cdot\ (\\
&\qquad \mathbf{sum}(z0\ \mathbf{in}\ ADDRESS,\\
&\qquad\quad r(inap\text{-}ai(callid,\ z0))\\
&\qquad\quad \cdot\ ai(callid,\ x,\ z0)\\
&\qquad)\\
&\qquad + r(inap\text{-}ci(callid))\\
&\qquad \cdot\ ci(callid,\ x,\ y)\\
&\qquad + \mathbf{sum}(z\ \mathbf{in}\ ADDRESS,\\
&\qquad\quad r(inap\text{-}conn(callid,\ x,\ z))\\
&\qquad\quad \cdot\ (\\
&\qquad\qquad [eq(z,\ adr0) = true] \rightarrow\\
&\qquad\qquad\quad ci(callid,\ x,\ y)\\
&\qquad\qquad + [eq(z,\ adr0) = false] \rightarrow\\
&\qquad\qquad\quad ai(callid,\ x,\ z)\\
&\qquad\quad)\\
&\qquad)\\
&\qquad + (\\
&\qquad\quad r(inap\text{-}cont(callid))\\
&\qquad\quad + r(inap\text{-}cwa(callid))\\
&\qquad)\\
&\qquad \cdot\ sr(callid,\ x,\ y)\\
&\qquad + r(inap\text{-}rc(callid,\ x))\\
&\qquad \cdot\ onull(callid,\ x,\ y)\\
&\qquad + \mathbf{sum}(z1\ \mathbf{in}\ ADDRESS,\\
&\qquad\quad r(inap\text{-}sr(callid,\ x,\ z1))\\
&\qquad\quad \cdot\ sr(callid,\ x,\ z1)\\
&\qquad)\\
&\quad)\\
&\quad + (\\
&\qquad s(inap\text{-}Analysed(ind,\ callid,\ x))\\
&\qquad + \mathbf{skip})\\
&\quad \cdot\ sr(callid,\ x,\ y)\\
dp\text{-}rsf(callid,\ x,\ y) &= s(inap\text{-}RouteFail(req,\ callid,\ x))\\
&\quad \cdot\ (\\
&\qquad \mathbf{sum}(z0\ \mathbf{in}\ ADDRESS,\\
&\qquad\quad r(inap\text{-}ai(callid,\ z0))\\
&\qquad\quad \cdot\ ai(callid,\ x,\ z0)\\
&\qquad)\\
&\qquad + r(inap\text{-}ci(callid))\\
&\qquad \cdot\ ci(callid,\ x,\ y)\\
&\qquad + (\\
&\qquad\quad r(inap\text{-}cont(callid))\\
&\qquad\quad + r(inap\text{-}cwa(callid))\\
&\qquad)\\
&\qquad \cdot\ oe(callid,\ x,\ y)\\
&\qquad + r(inap\text{-}rc(callid,\ x))\\
&\qquad \cdot\ onull(callid,\ x,\ y)\\
&\qquad + \mathbf{sum}(z1\ \mathbf{in}\ ADDRESS,\\
&\qquad\quad r(inap\text{-}sr(callid,\ x,\ z1))
\end{aligned}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\cdot\ \mathit{sr}(\mathit{callid},\ x,\ \mathit{z1}) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad) \\
\qquad\qquad + (\\
\qquad\qquad\qquad\qquad \mathit{s}(\mathit{inap\text{-}RouteFail}(\mathit{ind},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad + \mathbf{skip}) \\
\qquad\qquad\cdot\ \mathit{oe}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}ocpb}(\mathit{callid},\ x,\ y) = \mathit{s}(\mathit{inap\text{-}OBusy}(\mathit{req},\ \mathit{callid},\ x)) \\
\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad \mathbf{sum}(\mathit{z0}\ \mathbf{in}\ \mathit{ADDRESS}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{r}(\mathit{inap\text{-}ai}(\mathit{callid},\ \mathit{z0})) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ \mathit{ai}(\mathit{callid},\ x,\ \mathit{z0}) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad + \mathit{r}(\mathit{inap\text{-}ci}(\mathit{callid})) \\
\qquad\qquad\qquad\cdot\ \mathit{ci}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad + \mathbf{sum}(\mathit{z}\ \mathbf{in}\ \mathit{ADDRESS}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{r}(\mathit{inap\text{-}conn}(\mathit{callid},\ x,\ \mathit{z})) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\mathit{eq}(\mathit{z},\ \mathit{adr0}) = \mathit{true}] \rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{ci}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + [\mathit{eq}(\mathit{z},\ \mathit{adr0}) = \mathit{false}] \rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{ai}(\mathit{callid},\ x,\ \mathit{z}) \\
\qquad\qquad\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad + (\\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{r}(\mathit{inap\text{-}cont}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad\qquad + \mathit{r}(\mathit{inap\text{-}cwa}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\cdot\ \mathit{oe}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad + \mathit{r}(\mathit{inap\text{-}rc}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad + \mathbf{sum}(\mathit{z1}\ \mathbf{in}\ \mathit{ADDRESS}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{r}(\mathit{inap\text{-}sr}(\mathit{callid},\ x,\ \mathit{z1})) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ \mathit{sr}(\mathit{callid},\ x,\ \mathit{z1}) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad) \\
\qquad\qquad + (\\
\qquad\qquad\qquad\qquad \mathit{s}(\mathit{inap\text{-}OBusy}(\mathit{ind},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad + \mathbf{skip}) \\
\qquad\qquad\cdot\ \mathit{oe}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}ona}(\mathit{callid},\ x,\ y) = \mathit{s}(\mathit{inap\text{-}ONoAnswer}(\mathit{req},\ \mathit{callid},\ x)) \\
\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad \mathbf{sum}(\mathit{z0}\ \mathbf{in}\ \mathit{ADDRESS}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{r}(\mathit{inap\text{-}ai}(\mathit{callid},\ \mathit{z0})) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ \mathit{ai}(\mathit{callid},\ x,\ \mathit{z0}) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad + \mathit{r}(\mathit{inap\text{-}ci}(\mathit{callid})) \\
\qquad\qquad\qquad\cdot\ \mathit{ci}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad + (\\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{r}(\mathit{inap\text{-}cont}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad\qquad + \mathit{r}(\mathit{inap\text{-}cwa}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad) \\
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\cdot\ \mathit{oe}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad+\mathit{r}(\mathit{inap\text{-}rc}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad+\mathbf{sum}(\mathit{z1}\ \mathbf{in}\ \mathit{ADDRESS}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\mathit{r}(\mathit{inap\text{-}sr}(\mathit{callid},\ x,\ \mathit{z1})) \\
\qquad\qquad\qquad\qquad\qquad\quad\cdot\ \mathit{sr}(\mathit{callid},\ x,\ \mathit{z1}) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad) \\
\qquad\qquad\qquad+(\\
\qquad\qquad\qquad\qquad\qquad\mathit{s}(\mathit{inap\text{-}ONoAnswer}(\mathit{ind},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad+\mathbf{skip}) \\
\qquad\qquad\qquad\cdot\ \mathit{oe}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}od}(\mathit{callid},\ x,\ y) = \mathit{s}(\mathit{inap\text{-}ODisc}(\mathit{req},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad\qquad\mathbf{sum}(\mathit{z0}\ \mathbf{in}\ \mathit{ADDRESS}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\mathit{r}(\mathit{inap\text{-}ai}(\mathit{callid},\ \mathit{z0})) \\
\qquad\qquad\qquad\qquad\qquad\quad\cdot\ \mathit{ai}(\mathit{callid},\ x,\ \mathit{z0}) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad+\mathit{r}(\mathit{inap\text{-}ci}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{ci}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad+(\\
\qquad\qquad\qquad\qquad\qquad\qquad\mathit{r}(\mathit{inap\text{-}cont}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad\qquad\quad+\mathit{r}(\mathit{inap\text{-}cwa}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{s}(\mathit{calling\text{-}disconn}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad+\mathit{r}(\mathit{inap\text{-}rc}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad+\mathbf{sum}(\mathit{z1}\ \mathbf{in}\ \mathit{ADDRESS}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\mathit{r}(\mathit{inap\text{-}sr}(\mathit{callid},\ x,\ \mathit{z1})) \\
\qquad\qquad\qquad\qquad\qquad\quad\cdot\ \mathit{sr}(\mathit{callid},\ x,\ \mathit{z1}) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad) \\
\qquad\qquad\qquad+(\\
\qquad\qquad\qquad\qquad\qquad\mathit{s}(\mathit{inap\text{-}ODisc}(\mathit{ind},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad+\mathbf{skip}) \\
\qquad\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad\qquad\mathit{s}(\mathit{calling\text{-}disconn}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad+\mathit{r}(\mathit{called\text{-}disconn}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad+\mathit{r}(\mathit{onhook}(x)) \\
\qquad\qquad\qquad) \\
\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}oab}(\mathit{callid},\ x,\ y) = \mathit{s}(\mathit{inap\text{-}OAbandon}(\mathit{req},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\cdot\ \mathit{r}(\mathit{inap\text{-}rc}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad+(\\
\qquad\qquad\qquad\qquad\qquad\mathit{s}(\mathit{inap\text{-}OAbandon}(\mathit{ind},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad+\mathbf{skip}) \\
\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}omc\text{-}sc}(\mathit{callid},\ x,\ y) = \mathit{s}(\mathit{inap\text{-}OMid}(\mathit{req},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad\qquad\mathit{r}(\mathit{inap\text{-}dl}(\mathit{callid},\ x))
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}ml}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{sc}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}mgc}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ r(\mathit{offhook}(x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{oal}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad) \\
\qquad\qquad + (\\
\qquad\qquad\qquad\qquad s(\mathit{inap\text{-}OMid}(\mathit{ind},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad + \mathbf{skip}) \\
\qquad\qquad\cdot\ \mathit{sc}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}omc\text{-}oal}(\mathit{callid},\ x,\ y) = s(\mathit{inap\text{-}OMid}(\mathit{req},\ \mathit{callid},\ x)) \\
\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad r(\mathit{inap\text{-}dl}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}ml}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{oal}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}mgc}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ r(\mathit{offhook}(x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{oal}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad) \\
\qquad\qquad + (\\
\qquad\qquad\qquad\qquad s(\mathit{inap\text{-}OMid}(\mathit{ind},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad + \mathbf{skip}) \\
\qquad\qquad\cdot\ \mathit{oal}(\mathit{callid},\ x,\ y) \\
\mathit{dp\text{-}omc\text{-}oac}(\mathit{callid},\ x,\ y) = s(\mathit{inap\text{-}OMid}(\mathit{req},\ \mathit{callid},\ x)) \\
\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad r(\mathit{inap\text{-}ci}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{ci}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + (\\
\qquad\qquad\qquad\qquad\qquad\qquad r(\mathit{inap\text{-}cont}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}cwa}(\mathit{callid})) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{oac}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}rc}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + \mathbf{sum}(\mathit{z1}\ \mathbf{in}\ \mathit{ADDRESS}, \\
\qquad\qquad\qquad\qquad\qquad\qquad r(\mathit{inap\text{-}sr}(\mathit{callid},\ x,\ \mathit{z1})) \\
\qquad\qquad\qquad\qquad\qquad\cdot\ \mathit{sr}(\mathit{callid},\ x,\ \mathit{z1}) \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}dl}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}ml}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{oac}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad\qquad + r(\mathit{inap\text{-}mgc}(\mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad\cdot\ r(\mathit{offhook}(x)) \\
\qquad\qquad\qquad\qquad\cdot\ \mathit{oac}(\mathit{callid},\ x,\ y) \\
\qquad\qquad\qquad) \\
\qquad\qquad + (\\
\qquad\qquad\qquad\qquad s(\mathit{inap\text{-}OMid}(\mathit{ind},\ \mathit{callid},\ x)) \\
\qquad\qquad\qquad\qquad + \mathbf{skip}) \\
\qquad\qquad\cdot\ \mathit{oac}(\mathit{callid},\ x,\ y)
\end{array}
$$

$$
\begin{aligned}
&\mathit{dp\text{-}omc\text{-}os}(\mathit{callid},\ x,\ y) = \mathit{s}(\mathit{inap\text{-}OMid}(\mathit{req},\ \mathit{callid},\ x)) \\
&\qquad \cdot\ (\\
&\qquad\qquad \mathit{r}(\mathit{inap\text{-}ci}(\mathit{callid})) \\
&\qquad\qquad \cdot\ \mathit{ci}(\mathit{callid},\ x,\ y) \\
&\qquad\qquad +\ (\\
&\qquad\qquad\qquad \mathit{r}(\mathit{inap\text{-}cont}(\mathit{callid})) \\
&\qquad\qquad\qquad +\ \mathit{r}(\mathit{inap\text{-}cwa}(\mathit{callid})) \\
&\qquad\qquad) \\
&\qquad\qquad \cdot\ \mathit{os}(\mathit{callid},\ x,\ y) \\
&\qquad\qquad +\ \mathit{r}(\mathit{inap\text{-}rc}(\mathit{callid},\ x)) \\
&\qquad\qquad \cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
&\qquad\qquad +\ \mathbf{sum}(\mathit{z1}\ \mathbf{in}\ \mathit{ADDRESS}, \\
&\qquad\qquad\qquad \mathit{r}(\mathit{inap\text{-}sr}(\mathit{callid},\ x,\ \mathit{z1})) \\
&\qquad\qquad\qquad \cdot\ \mathit{sr}(\mathit{callid},\ x,\ \mathit{z1}) \\
&\qquad\qquad) \\
&\qquad\qquad +\ \mathit{r}(\mathit{inap\text{-}dl}(\mathit{callid},\ x)) \\
&\qquad\qquad \cdot\ \mathit{onull}(\mathit{callid},\ x,\ y) \\
&\qquad\qquad +\ \mathit{r}(\mathit{inap\text{-}ml}(\mathit{callid},\ x)) \\
&\qquad\qquad \cdot\ \mathit{os}(\mathit{callid},\ x,\ y) \\
&\qquad\qquad +\ \mathit{r}(\mathit{inap\text{-}mgc}(\mathit{callid},\ x)) \\
&\qquad\qquad \cdot\ \mathit{r}(\mathit{offhook}(x)) \\
&\qquad\qquad \cdot\ \mathit{os}(\mathit{callid},\ x,\ y) \\
&\qquad\quad) \\
&\qquad +\ (\\
&\qquad\qquad\qquad \mathit{s}(\mathit{inap\text{-}OMid}(\mathit{ind},\ \mathit{callid},\ x)) \\
&\qquad\qquad +\ \mathbf{skip}) \\
&\qquad \cdot\ \mathit{r}(\mathit{flashhook}(x)) \\
&\qquad \cdot\ \mathit{os}(\mathit{callid},\ x,\ y) \\
&\mathit{dp\text{-}ots}(\mathit{callid},\ x,\ y) = \mathbf{skip} \\
&\qquad \cdot\ \mathit{oal}(\mathit{callid},\ x,\ y) \\
&\mathit{dp\text{-}oan}(\mathit{callid},\ x,\ y) = (\\
&\qquad\qquad\qquad \mathit{s}(\mathit{inap\text{-}OAnswer}(\mathit{ind},\ \mathit{callid},\ x)) \\
&\qquad\qquad +\ \mathbf{skip}) \\
&\qquad \cdot\ \mathit{oac}(\mathit{callid},\ x,\ y) \\
&\mathit{dp\text{-}os}(\mathit{callid},\ x,\ y) = (\\
&\qquad\qquad\qquad \mathit{s}(\mathit{inap\text{-}OSuspend}(\mathit{ind},\ \mathit{callid},\ x)) \\
&\qquad\qquad +\ \mathbf{skip}) \\
&\qquad \cdot\ \mathit{os}(\mathit{callid},\ x,\ y) \\
&\mathit{dp\text{-}or}(\mathit{callid},\ x,\ y) = \mathbf{skip} \\
&\qquad \cdot\ \mathit{oac}(\mathit{callid},\ x,\ y)
\end{aligned}
$$

end *OBCSM*

---

process module *TBCSM*
−−This is the PSF describtion of TBCSM in CS2, Q.1214
−−
−−TBCSM needs to communicate with the called party, OBCSM, and IN, with the
−−signals from signal set USERSIG, INTRASIG and INAPSIG, respectively
−−
−−In this original TBCSM model, the communication directly between
−−the calling party and the TBCSM is forbidden,
−−i.e., such communication must be compeleted via OBCSM
begin
exports
begin
atoms
s : *INTRASIG*
s : *INAPSIG*
s : *USERSIG*
s : *BCSMCVSSIG*
r : *INTRASIG*
r : *INAPSIG*
r : *USERSIG*
termination-denied, *presentation-failure*, *call-rejected*, *t-active-failure*,
t-suspend-failure, *ss7-failure*, *treconnect*, *term-auth-success*, *rec-time-expire*
processes
tbcsm : *ID # ADDRESS # ADDRESS*
tnull : *ID # ADDRESS # ADDRESS*
ata : *ID # ADDRESS # ADDRESS*
sf : *ID # ADDRESS # ADDRESS*
pc : *ID # ADDRESS # ADDRESS*
tal : *ID # ADDRESS # ADDRESS*
tac : *ID # ADDRESS # ADDRESS*
ts : *ID # ADDRESS # ADDRESS*
te : *ID # ADDRESS # ADDRESS*
dp-tat : *ID # ADDRESS # ADDRESS*
dp-taa : *ID # ADDRESS # ADDRESS*
dp-fsa : *ID # ADDRESS # ADDRESS*
dp-ca : *ID # ADDRESS # ADDRESS*
dp-tan : *ID # ADDRESS # ADDRESS*
dp-tra : *ID # ADDRESS # ADDRESS*
dp-tmc : *ID # ADDRESS # ADDRESS*
dp-tab : *ID # ADDRESS # ADDRESS*
dp-ts : *ID # ADDRESS # ADDRESS*
dp-td : *ID # ADDRESS # ADDRESS*
dp-tb : *ID # ADDRESS # ADDRESS*
dp-tna : *ID # ADDRESS # ADDRESS*
end
imports
Usersigs, *Inapsigs*, *Networkdatas*, *Intrasigs*
sets
of atoms
swtt-lock = { *termination-denied*, *presentation-failure*, *call-rejected*,

$t\text{-}active\text{-}failure,\ t\text{-}suspend\text{-}failure,\ ss7\text{-}failure\ \}$

$swtt\text{-}skip = \{\ treconnect,\ term\text{-}auth\text{-}success\ \}$

variables

$callid \quad : \rightarrow ID$

$x,\ y \quad : \rightarrow ADDRESS$

definitions

$tbcsm(callid,\ x,\ y) = \mathbf{hide}(swtt\text{-}skip,$
$\quad \mathbf{encaps}(swtt\text{-}lock,$
$\quad\quad tnull(callid,\ x,\ y)$
$\quad)$
$)$

$tnull(callid,\ x,\ y) = r(initiate\text{-}tb(callid,\ y))$
$\quad \cdot\ dp\text{-}tat(callid,\ x,\ y)$
$\quad + s(inap\text{-}TNull(ind,\ callid,\ x))$
$\quad \cdot\ tnull(callid,\ x,\ y)$

$ata(callid,\ x,\ y) = r(calling\text{-}abandon(callid))$
$\quad \cdot\ dp\text{-}tab(callid,\ x,\ y)$
$\quad + termination\text{-}denied$
$\quad \cdot\ te(callid,\ x,\ y)$
$\quad + term\text{-}auth\text{-}success$
$\quad \cdot\ dp\text{-}taa(callid,\ x,\ y)$

$sf(callid,\ x,\ y) = r(calling\text{-}abandon(callid))$
$\quad \cdot\ dp\text{-}tab(callid,\ x,\ y)$
$\quad + r(userbusy(y))$
$\quad \cdot\ dp\text{-}tb(callid,\ x,\ y)$
$\quad + r(useridle(y))$
$\quad \cdot\ dp\text{-}fsa(callid,\ x,\ y)$

$pc(callid,\ x,\ y) = r(calling\text{-}abandon(callid))$
$\quad \cdot\ dp\text{-}tab(callid,\ x,\ y)$
$\quad + r(offhook(y))$
$\quad \cdot\ s(call\text{-}acceptance(callid))$
$\quad \cdot\ dp\text{-}tan(callid,\ x,\ y)$
$\quad + ss7\text{-}failure$
$\quad \cdot\ sf(callid,\ x,\ y)$
$\quad + presentation\text{-}failure$
$\quad \cdot\ s(non\text{-}presentation(callid))$
$\quad \cdot\ te(callid,\ x,\ y)$
$\quad + s(alert\text{-}user(y))$
$\quad \cdot\ dp\text{-}ca(callid,\ x,\ y)$
$\quad + s(CWtone(y))$
$\quad \cdot\ dp\text{-}ca(callid,\ x,\ y)$

$tal(callid,\ x,\ y) = s(called\text{-}alerting(callid))$
$\quad \cdot\ ($
$\quad\quad r(offhook(y))$
$\quad\quad \cdot\ s(call\text{-}acceptance(callid))$
$\quad\quad \cdot\ dp\text{-}tan(callid,\ x,\ y)$
$\quad\quad + r(noanswer(y))$
$\quad\quad \cdot\ dp\text{-}tna(callid,\ x,\ y)$
$\quad\quad + r(calling\text{-}abandon(callid))$
$\quad\quad \cdot\ dp\text{-}tab(callid,\ x,\ y)$
$\quad\quad + r(inap\text{-}ml(callid,\ y))$
$\quad\quad \cdot\ s(stop\text{-}CWtone(x))$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\cdot\ s(call\text{-}acceptance(callid)) \\
\qquad\qquad\qquad\qquad\cdot\ dp\text{-}tan(callid,\ x,\ y) \\
\qquad\qquad\qquad) \\
\qquad\qquad + r(calling\text{-}abandon(callid)) \\
\qquad\qquad\cdot\ dp\text{-}tab(callid,\ x,\ y) \\
\qquad\qquad + call\text{-}rejected \\
\qquad\qquad\cdot\ s(called\text{-}rejected(callid)) \\
\qquad\qquad\cdot\ te(callid,\ x,\ y) \\
tac(callid,\ x,\ y) = r(calling\text{-}disconn(callid)) \\
\qquad\qquad\cdot\ dp\text{-}td(callid,\ x,\ y) \\
\qquad\qquad + r(flashhook(y)) \\
\qquad\qquad\cdot\ dp\text{-}tmc(callid,\ x,\ y) \\
\qquad\qquad + r(onhook(y)) \\
\qquad\qquad\cdot\ dp\text{-}ts(callid,\ x,\ y) \\
\qquad\qquad + t\text{-}active\text{-}failure \\
\qquad\qquad\cdot\ te(callid,\ x,\ y) \\
ts(callid,\ x,\ y) = r(calling\text{-}disconn(callid)) \\
\qquad\qquad\cdot\ dp\text{-}td(callid,\ x,\ y) \\
\qquad\qquad + rec\text{-}time\text{-}expire \\
\qquad\qquad\cdot\ dp\text{-}td(callid,\ x,\ y) \\
\qquad\qquad + r(offhook(y)) \\
\qquad\qquad\cdot\ dp\text{-}tra(callid,\ x,\ y) \\
\qquad\qquad + t\text{-}suspend\text{-}failure \\
\qquad\qquad\cdot\ te(callid,\ x,\ y) \\
te(callid,\ x,\ y) = \mathbf{skip} \\
dp\text{-}td(callid,\ x,\ y) = s(inap\text{-}TDisc(req,\ callid,\ y)) \\
\qquad\qquad\cdot\ r(inap\text{-}rc(callid,\ y)) \\
\qquad\qquad\cdot\ tnull(callid,\ x,\ y) \\
\qquad\qquad + (\\
\qquad\qquad\qquad\qquad s(inap\text{-}TDisc(ind,\ callid,\ y)) \\
\qquad\qquad\qquad + \mathbf{skip}) \\
\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad r(onhook(y)) \\
\qquad\qquad\qquad + s(called\text{-}disconn(callid)) \\
\qquad\qquad) \\
\qquad\qquad\cdot\ tnull(callid,\ x,\ y) \\
dp\text{-}tat(callid,\ x,\ y) = s(inap\text{-}TAttempt(req,\ callid,\ y)) \\
\qquad\qquad\cdot\ (\\
\qquad\qquad\qquad\qquad r(inap\text{-}at(callid)) \\
\qquad\qquad\qquad\cdot\ ata(callid,\ x,\ y) \\
\qquad\qquad\qquad + (\\
\qquad\qquad\qquad\qquad\qquad\qquad r(inap\text{-}cont(callid)) \\
\qquad\qquad\qquad\qquad\qquad + r(inap\text{-}cwa(callid)) \\
\qquad\qquad\qquad) \\
\qquad\qquad\qquad\cdot\ ata(callid,\ x,\ y) \\
\qquad\qquad\qquad + r(inap\text{-}sf(callid)) \\
\qquad\qquad\qquad\cdot\ sf(callid,\ x,\ y) \\
\qquad\qquad\qquad + r(inap\text{-}conn(callid,\ x,\ y)) \\
\qquad\qquad\qquad\cdot\ sf(callid,\ x,\ y) \\
\qquad\qquad\qquad + r(inap\text{-}rc(callid,\ y)) \\
\qquad\qquad\qquad\cdot\ tnull(callid,\ x,\ y) \\
\qquad\qquad) \\
\end{array}
$$

$$
\begin{aligned}
&\quad + (\\
&\qquad\quad s(\mathit{inap\text{-}TAttempt}(\mathit{ind},\ \mathit{callid},\ y))\\
&\qquad + \mathbf{skip})\\
&\quad \cdot\ \mathit{ata}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}taa}(\mathit{callid},\ x,\ y) &= s(\mathit{inap\text{-}TAuthorized}(\mathit{req},\ \mathit{callid},\ y))\\
&\quad \cdot\ (\\
&\qquad\quad r(\mathit{inap\text{-}at}(\mathit{callid}))\\
&\qquad \cdot\ \mathit{ata}(\mathit{callid},\ x,\ y)\\
&\qquad + (\\
&\qquad\qquad\quad r(\mathit{inap\text{-}cont}(\mathit{callid}))\\
&\qquad\qquad + r(\mathit{inap\text{-}cwa}(\mathit{callid}))\\
&\qquad)\\
&\qquad \cdot\ \mathit{sf}(\mathit{callid},\ x,\ y)\\
&\qquad + r(\mathit{inap\text{-}sf}(\mathit{callid}))\\
&\qquad \cdot\ \mathit{sf}(\mathit{callid},\ x,\ y)\\
&\qquad + r(\mathit{inap\text{-}rc}(\mathit{callid},\ y))\\
&\qquad \cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
&\quad)\\
&\quad + (\\
&\qquad\quad s(\mathit{inap\text{-}TAuthorized}(\mathit{ind},\ \mathit{callid},\ y))\\
&\qquad + \mathbf{skip})\\
&\quad \cdot\ \mathit{sf}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}fsa}(\mathit{callid},\ x,\ y) &= s(\mathit{inap\text{-}FacilAvail}(\mathit{req},\ \mathit{callid},\ y))\\
&\quad \cdot\ (\\
&\qquad\quad (\\
&\qquad\qquad\quad r(\mathit{inap\text{-}cont}(\mathit{callid}))\\
&\qquad\qquad + r(\mathit{inap\text{-}cwa}(\mathit{callid}))\\
&\qquad)\\
&\qquad \cdot\ \mathit{pc}(\mathit{callid},\ x,\ y)\\
&\qquad + r(\mathit{inap\text{-}sf}(\mathit{callid}))\\
&\qquad \cdot\ \mathit{sf}(\mathit{callid},\ x,\ y)\\
&\qquad + r(\mathit{inap\text{-}conn}(\mathit{callid},\ x,\ y))\\
&\qquad \cdot\ \mathit{sf}(\mathit{callid},\ x,\ y)\\
&\qquad + r(\mathit{inap\text{-}rc}(\mathit{callid},\ y))\\
&\qquad \cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
&\quad)\\
&\quad + (\\
&\qquad\quad s(\mathit{inap\text{-}FacilAvail}(\mathit{ind},\ \mathit{callid},\ y))\\
&\qquad + \mathbf{skip})\\
&\quad \cdot\ \mathit{pc}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}tna}(\mathit{callid},\ x,\ y) &= s(\mathit{inap\text{-}TNoAnswer}(\mathit{req},\ \mathit{callid},\ y))\\
&\quad \cdot\ (\\
&\qquad\quad r(\mathit{inap\text{-}at}(\mathit{callid}))\\
&\qquad \cdot\ \mathit{ata}(\mathit{callid},\ x,\ y)\\
&\qquad + (\\
&\qquad\qquad\quad r(\mathit{inap\text{-}cont}(\mathit{callid}))\\
&\qquad\qquad + r(\mathit{inap\text{-}cwa}(\mathit{callid}))\\
&\qquad)\\
&\qquad \cdot\ s(\mathit{called\text{-}noanswer}(\mathit{callid}))\\
&\qquad + r(\mathit{inap\text{-}sf}(\mathit{callid}))\\
&\qquad \cdot\ \mathit{sf}(\mathit{callid},\ x,\ y)\\
&\qquad + r(\mathit{inap\text{-}rc}(\mathit{callid},\ y))
\end{aligned}
$$

$$
\begin{aligned}
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ s(\mathit{called\text{-}noanswer}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad\quad)\\
&\quad\quad\quad\quad\quad+(\\
&\quad\quad\quad\quad\quad\quad\quad\quad s(\mathit{inap\text{-}TNoAnswer}(\mathit{ind},\ \mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\quad\quad\quad+\mathbf{skip})\\
&\quad\quad\quad\quad\quad\cdot\ s(\mathit{called\text{-}noanswer}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}tb}(\mathit{callid},\ x,\ y) &= s(\mathit{inap\text{-}TBusy}(\mathit{req},\ \mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\cdot\ (\\
&\quad\quad\quad\quad\quad\quad\quad\quad r(\mathit{inap\text{-}sf}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ \mathit{sf}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad\quad\quad\quad+r(\mathit{inap\text{-}mvc}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ \mathit{pc}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad\quad\quad\quad+r(\mathit{inap\text{-}rc}(\mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad\quad)\\
&\quad\quad\quad\quad\quad+(\\
&\quad\quad\quad\quad\quad\quad\quad\quad s(\mathit{inap\text{-}TBusy}(\mathit{ind},\ \mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\quad\quad\quad+\mathbf{skip})\\
&\quad\quad\quad\quad\quad\cdot\ s(\mathit{terminating\text{-}busy}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}ts}(\mathit{callid},\ x,\ y) &= s(\mathit{inap\text{-}TSuspend}(\mathit{req},\ \mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\cdot\ r(\mathit{inap\text{-}rc}(\mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad+(\\
&\quad\quad\quad\quad\quad\quad\quad\quad s(\mathit{inap\text{-}TSuspend}(\mathit{ind},\ \mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\quad\quad\quad+\mathbf{skip})\\
&\quad\quad\quad\quad\quad\cdot\ (\\
&\quad\quad\quad\quad\quad\quad\quad\quad s(\mathit{called\text{-}suspend}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ \mathit{ts}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad\quad\quad\quad+r(\mathit{calling\text{-}disconn}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad\quad)\\
\mathit{dp\text{-}ca}(\mathit{callid},\ x,\ y) &= \mathbf{skip}\\
&\quad\quad\quad\quad\quad\cdot\ \mathit{tal}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}tan}(\mathit{callid},\ x,\ y) &= \mathbf{skip}\\
&\quad\quad\quad\quad\quad\cdot\ \mathit{tac}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}tra}(\mathit{callid},\ x,\ y) &= s(\mathit{called\text{-}reanswer}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\cdot\ \mathit{tac}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad+r(\mathit{calling\text{-}disconn}(\mathit{callid}))\\
&\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}tab}(\mathit{callid},\ x,\ y) &= \mathbf{skip}\\
&\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
\mathit{dp\text{-}tmc}(\mathit{callid},\ x,\ y) &= s(\mathit{inap\text{-}TMid}(\mathit{req},\ \mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\cdot\ (\\
&\quad\quad\quad\quad\quad\quad\quad\quad r(\mathit{inap\text{-}ml}(\mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ \mathit{tac}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad\quad\quad\quad+r(\mathit{inap\text{-}dl}(\mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ \mathit{tnull}(\mathit{callid},\ x,\ y)\\
&\quad\quad\quad\quad\quad\quad\quad\quad+r(\mathit{inap\text{-}mgc}(\mathit{callid},\ y))\\
&\quad\quad\quad\quad\quad\quad\quad\quad\cdot\ s(\mathit{alert\text{-}user}(x))
\end{aligned}
$$

$\cdot$ $r(offhook(x))$
$\cdot$ $tac(callid,\ x,\ y)$
)
$+$ (
$s(inap\text{-}TMid(ind,\ callid,\ y))$
$+$ **skip**)
$\cdot$ $tac(callid,\ x,\ y)$

end *TBCSM*

Appendix C: PSF Process Modules: IN Services

---

process module *POT*
−−This is the PSF description for a plain old telephone call −−
−−The channel from OBCSM to TBCSM is named as ob−tb.
−−The channel from TBCSM to OBCSM is named as tb−ob.
−−The channel from TBCSM to USER is named as tb−user.
−−The channel from OBCSM to USER is named as ob−user.
−−The channel from OBCSM to CVS is called ob−cvs
−−The channel from TBCSM to CVS is called tb−cvs
−−The process mixed−call is the one which support both
−−IN transitions and POTs transitions
begin
exports
begin
atoms
ccalling-ob, *cob-calling*, *ccalled-tb*, *ctb-called* : *USERSIG*
cob-tb, *ctb-ob* : *INTRASIG*
cob-cvs, *ctb-cvs* : *BCSMCVSSIG*
processes
POT : *ID # ADDRESS # ADDRESS*
test
end
imports

OBCSM
{
renamed by
[
$s \rightarrow sOB$, $r \rightarrow rOB$
]
},
TBCSM
{
renamed by
[
$s \rightarrow sTB$, $r \rightarrow rTB$
]
},
User
{
renamed by
[
$s \rightarrow sUser$, $r \rightarrow rUser$
]
}
sets
of atoms
$usersig = \{ sUser(sig) \mid sig \textbf{ in } USERSIG \}$
$+ \{ rUser(sig) \mid sig \textbf{ in } USERSIG \}$
$+ \{ rOB(sig) \mid sig \textbf{ in } USERSIG \}$

$$
\begin{aligned}
&\quad + \{sOB(sig) \mid sig \textbf{ in } USERSIG\ \} \\
&\quad + \{rTB(sig) \mid sig \textbf{ in } USERSIG\ \} \\
&\quad + \{sTB(sig) \mid sig \textbf{ in } USERSIG\ \} \\
&intrasig = \{\ sOB(sig) \mid sig \textbf{ in } INTRASIG\ \} \\
&\quad + \{sTB(sig) \mid sig \textbf{ in } INTRASIG\ \} \\
&\quad + \{rOB(sig) \mid sig \textbf{ in } INTRASIG\ \} \\
&\quad + \{rTB(sig) \mid sig \textbf{ in } INTRASIG\ \} \\
&bcsmcvssig = \{\ sOB(sig) \mid sig \textbf{ in } BCSMCVSSIG\ \} \\
&\quad + \{sTB(sig) \mid sig \textbf{ in } BCSMCVSSIG\ \} \\
&enc\text{-}inap = \{\ rOB(sig),\ sOB(sig),\ rTB(sig),\ sTB(sig) \mid sig \textbf{ in } INAPSIG\ \} \\
&enc = usersig \\
&\quad + intrasig \\
&\quad + bcsmcvssig \\
&\quad + enc\text{-}inap
\end{aligned}
$$

communications

$$
\begin{aligned}
&sUser(sig) \mid rOB(sig) = ccalling\text{-}ob(sig) \textbf{ for } sig \textbf{ in } USERSIG \\
&rUser(sig) \mid sOB(sig) = cob\text{-}calling(sig) \textbf{ for } sig \textbf{ in } USERSIG \\
&sUser(sig) \mid rTB(sig) = ccalled\text{-}tb(sig) \textbf{ for } sig \textbf{ in } USERSIG \\
&rUser(sig) \mid sTB(sig) = ctb\text{-}called(sig) \textbf{ for } sig \textbf{ in } USERSIG \\
&sOB(sig) \mid rTB(sig) = cob\text{-}tb(sig) \textbf{ for } sig \textbf{ in } INTRASIG \\
&sTB(sig) \mid rOB(sig) = ctb\text{-}ob(sig) \textbf{ for } sig \textbf{ in } INTRASIG
\end{aligned}
$$

variables

$$
\begin{aligned}
&callid : \rightarrow ID \\
&x,\ y : \rightarrow ADDRESS
\end{aligned}
$$

definitions

$$
\begin{aligned}
&POT(callid,\ x,\ y) = \textbf{encaps}(enc, \\
&\qquad idle(x) \\
&\qquad \parallel\ idle(y) \\
&\qquad \parallel\ obcsm(callid,\ x,\ y) \\
&\qquad \parallel\ tbcsm(callid,\ x,\ y) \\
&\quad)
\end{aligned}
$$

end POT

---

process module *CW*
−−This is the PSF description of Call Waiting service −−
begin
exports
begin
atoms
r, *s* : *INAPSIG*
processes
CW, *CW-null* : *ID* # *ID* # *ID* # *ADDRESS* # *ADDRESS* # *ADDRESS*
CW-active, *CW-cw*, *CW-ch* : *ID* # *ID* # *ID* # *ADDRESS* # *ADDRESS* # *ADDRESS*
end
imports
Inapsigs
variables
svcid, *callxy*, *callxz* : → *ID*
x, *y*, *z* : → *ADDRESS*
definitions
CW(*svcid*, *callxy*, *callxz*, *x*, *y*, *z*) = *CW-null*(*svcid*, *callxy*, *callxz*, *x*, *y*, *z*)
CW-null(*svcid*, *callxy*, *callxz*, *x*, *y*, *z*) = *r*(*inap-TBusy*(*req*, *callxz*, *x*))
· *CW-active*(*svcid*, *callxy*, *callxz*, *x*, *y*, *z*)
CW-active(*svcid*, *callxy*, *callxz*, *x*, *y*, *z*) = *s*(*inap-mvc*(*callxz*))
· *CW-cw*(*svcid*, *callxy*, *callxz*, *x*, *y*, *z*)
CW-cw(*svcid*, *callxy*, *callxz*, *x*, *y*, *z*) = (
r(*inap-OMid*(*req*, *callxy*, *x*))
+ *r*(*inap-TMid*(*req*, *callxy*, *x*))
)
· *s*(*inap-ml*(*callxz*, *x*))
· *CW-ch*(*svcid*, *callxz*, *callxy*, *x*, *z*, *y*)
+ *r*(*inap-TAbandon*(*ind*, *callxz*, *x*))
+ *r*(*inap-OAbandon*(*ind*, *callxy*, *x*))
+ *r*(*inap-ODisc*(*ind*, *callxy*, *x*))
+ *r*(*inap-TDisc*(*ind*, *callxy*, *x*))
CW-ch(*svcid*, *callxy*, *callxz*, *x*, *y*, *z*) = (
r(*inap-OMid*(*req*, *callxy*, *x*))
+ *r*(*inap-TMid*(*req*, *callxy*, *x*))
)
· *s*(*inap-ml*(*callxz*, *x*))
· *CW-ch*(*svcid*, *callxz*, *callxy*, *x*, *z*, *y*)
+ (
r(*inap-ODisc*(*ind*, *callxy*, *x*))
+ *r*(*inap-TSuspend*(*ind*, *callxy*, *x*))
+ *r*(*inap-OAbandon*(*ind*, *callxy*, *x*))
+ *r*(*inap-TAbandon*(*ind*, *callxy*, *x*))
)
· *s*(*inap-mgc*(*callxz*, *x*))
· **skip**
+ (
r(*inap-OAbandon*(*req*, *callxz*, *z*))
+ *r*(*inap-ODisc*(*req*, *callxz*, *z*))

$$
\begin{array}{l}
\quad + r(inap\text{-}TDisc(req,\ callxz,\ z)) \\
\quad + r(inap\text{-}TSuspend(req,\ callxz,\ z)) \\
\quad + r(inap\text{-}TNoAnswer(req,\ callxz,\ z)) \\
\quad) \\
\cdot\ s(inap\text{-}dl(callxz,\ x)) \\
\cdot\ s(inap\text{-}rc(callxz,\ z)) \\
+ r(inap\text{-}OAbandon(ind,\ callxy,\ y)) \\
\cdot\ r(inap\text{-}TAbandon(req,\ callxy,\ x)) \\
\cdot\ s(inap\text{-}rc(callxy,\ x)) \\
\cdot\ s(inap\text{-}ml(callxz,\ x)) \\
+ r(inap\text{-}ODisc(ind,\ callxy,\ y)) \\
\cdot\ r(inap\text{-}TDisc(req,\ callxy,\ x)) \\
\cdot\ s(inap\text{-}rc(callxy,\ x)) \\
\cdot\ s(inap\text{-}ml(callxz,\ x)) \\
+ r(inap\text{-}TSuspend(ind,\ callxy,\ y)) \\
\cdot\ r(inap\text{-}ODisc(req,\ callxy,\ x)) \\
\cdot\ s(inap\text{-}rc(callxy,\ x)) \\
\cdot\ s(inap\text{-}ml(callxz,\ x))
\end{array}
$$

end CW

---

process module *TWC*
−−This is the PSF description of Three Way Calling service −−
begin
exports
begin
atoms
r, s : *INAPSIG*
processes
TWC, TWC-m-orig, TWC-mstable, TWC-transfer, TWC-ch : *ID # ID # ID # ADDRESS # ADDRESS # ADDRESS*
end
imports
Inapsigs
variables
svcid, callxy, callxz : $\rightarrow$ *ID*
x, y, z : $\rightarrow$ *ADDRESS*
definitions
$$
\begin{aligned}
&TWC(svcid,\ callxy,\ callxz,\ x,\ y,\ z) = (\\
&\qquad r(inap\text{-}TMid(req,\ callxy,\ x))\\
&\qquad + r(inap\text{-}OMid(req,\ callxy,\ x))\\
&\quad)\\
&\quad \cdot\ s(inap\text{-}oc(callxz,\ x))\\
&\quad \cdot\ TWC\text{-}m\text{-}orig(svcid,\ callxy,\ callxz,\ x,\ y,\ z)\\
&TWC\text{-}m\text{-}orig(svcid,\ callxy,\ callxz,\ x,\ y,\ z) = r(inap\text{-}OAuthorized(ind,\ callxz,\ x))\\
&\quad \cdot\ TWC\text{-}ch(svcid,\ callxz,\ callxy,\ x,\ z,\ y)\\
&\quad + r(inap\text{-}OAbandon(ind,\ callxz,\ x))\\
&\quad \cdot\ s(inap\text{-}ml(callxy,\ x))\\
&\quad + (\\
&\qquad r(inap\text{-}TNoAnswer(req,\ callxy,\ y))\\
&\qquad + r(inap\text{-}TSuspend(req,\ callxy,\ y))\\
&\qquad + r(inap\text{-}OAbandon(req,\ callxy,\ y))\\
&\qquad + r(inap\text{-}ODisc(req,\ callxy,\ y))\\
&\quad)\\
&\quad \cdot\ s(inap\text{-}rc(callxy,\ y))\\
&\quad \cdot\ s(inap\text{-}dl(callxy,\ x))\\
&TWC\text{-}ch(svcid,\ callxy,\ callxz,\ x,\ y,\ z) = (\\
&\qquad r(inap\text{-}OMid(req,\ callxy,\ x))\\
&\qquad + r(inap\text{-}TMid(req,\ callxy,\ x))\\
&\quad)\\
&\quad \cdot\ s(inap\text{-}ml(callxz,\ x))\\
&\quad \cdot\ TWC\text{-}ch(svcid,\ callxz,\ callxy,\ x,\ z,\ y)\\
&\quad + (\\
&\qquad r(inap\text{-}ODisc(ind,\ callxy,\ x))\\
&\qquad + r(inap\text{-}TSuspend(ind,\ callxy,\ x))\\
&\qquad + r(inap\text{-}OAbandon(ind,\ callxy,\ x))\\
&\qquad + r(inap\text{-}TAbandon(ind,\ callxy,\ x))\\
&\quad)\\
&\quad \cdot\ s(inap\text{-}mgc(callxz,\ x))\\
&\quad + (\\
&\qquad r(inap\text{-}OAbandon(req,\ callxz,\ z))
\end{aligned}
$$

$$
\begin{array}{l}
\quad + r(inap\text{-}ODisc(req,\ callxz,\ z)) \\
\quad + r(inap\text{-}TDisc(req,\ callxz,\ z)) \\
\quad + r(inap\text{-}TSuspend(req,\ callxz,\ z)) \\
\quad + r(inap\text{-}TNoAnswer(req,\ callxz,\ z)) \\
\quad) \\
\cdot\ s(inap\text{-}dl(callxz,\ x)) \\
\cdot\ s(inap\text{-}rc(callxz,\ z)) \\
+ r(inap\text{-}OAbandon(ind,\ callxy,\ y)) \\
\cdot\ r(inap\text{-}TAbandon(req,\ callxy,\ x)) \\
\cdot\ s(inap\text{-}rc(callxy,\ x)) \\
\cdot\ s(inap\text{-}ml(callxz,\ x)) \\
+ r(inap\text{-}ODisc(ind,\ callxy,\ y)) \\
\cdot\ r(inap\text{-}TDisc(req,\ callxy,\ x)) \\
\cdot\ s(inap\text{-}rc(callxy,\ x)) \\
\cdot\ s(inap\text{-}ml(callxz,\ x)) \\
+ r(inap\text{-}TSuspend(ind,\ callxy,\ y)) \\
\cdot\ r(inap\text{-}ODisc(req,\ callxy,\ x)) \\
\cdot\ s(inap\text{-}rc(callxy,\ x)) \\
\cdot\ s(inap\text{-}ml(callxz,\ x)) \\
+ r(inap\text{-}TNoAnswer(ind,\ callxy,\ y)) \\
\cdot\ r(inap\text{-}ONoAnswer(req,\ callxy,\ x)) \\
\cdot\ s(inap\text{-}rc(callxy,\ x)) \\
\cdot\ s(inap\text{-}ml(callxz,\ x))
\end{array}
$$

end TWC

```
process module AR
--This is the PSF description of AR service --
begin
    exports
    begin
        atoms
            r, s            : INAPSIG
        processes
            AR, AR-Null : ID # ID # ADDRESS # ADDRESS
            AR-Active   : ID # ID # ADDRESS # ADDRESS
            AR-Await    : ID # ID # ID # ADDRESS # ADDRESS # ADDRESS
    end
    imports
        Inapsigs
    variables
        svcid, callxy, callyz : → ID
        x, y, z               : → ADDRESS
    definitions
        AR(svcid, callxy, x, y) = AR-Null(svcid, callxy, x, y)
        AR-Null(svcid, callxy, x, y) = sum(callyz in ID,
                                    sum(z in ADDRESS,
                                        r(inap-TAttempt(ind, callxy, y))
                                      · AR-Await(svcid, callxy, callyz, x, y, z)
                                    )
                                )
        AR-Await(svcid, callxy, callyz, x, y, z) = (
                                    r(inap-TNull(ind, callyz, y))
                                  + r(inap-ONull(ind, callyz, y))
                                )
                              · AR-Active(svcid, callxy, x, y)
        AR-Active(svcid, callxy, x, y) = sum(z in ADDRESS,
                                    sum(callyz in ID,
                                        (
                                            r(inap-OAttempt(ind, callyz, y))
                                          + r(inap-CAccept(ind, callyz, y))
                                          + r(inap-TAnswer(ind, callyz, y))
                                        )
                                      · AR-Await(svcid, callxy, callyz, x, y, z)
                                    )
                                )
                              + s(inap-ica(callxy, y))
                              · r(inap-OAuthorized(req, callxy, y))
                              · s(inap-sr(callxy, y, x))
                              · AR-Null(svcid, callxy, x, y)
end AR
```

11 Appendix D: List of Service States

Service	State	Comments
CW	CW_Null_x	The initial state of CW service
	$\text{CW\_call\_hold}_{x,y,z}$	Holding the call from z, x is in the conversation with y
	$\text{CW\_call\_wait}_{x,y,z}$	A call from z to x is waiting; x is in the conversation with y.
	x: CW subscriber	
TWC	TWC_Null_x	The initial state of TWC service
	$\text{TWC\_call\_hold}_{x,y,z}$	Holding the call from z, x is in the conversation with y
	$\text{TWC\_M\_orig}_{x,y}$	The subscriber x has put y on hold to initiate 3-party call.
	x: TWC subscriber	
CFB	CFB_Null_x	The initial state of the CFB service
	$\text{CFB\_forward}_{x,y,z}$	The subscriber x has fowarded an incoming call to z when in connection with y.
	x: CFB subscriber	
AR	AR_Null_x	The initial state of the AR service
	$\text{AR\_Await}_{x,y,z}$	AR is monitoring the subscriber x to become idle.
	$\text{AR\_Active}_{x,y}$	Terminal x is free, and AR is initiating a call from x to y.
	x: AR subscriber	
CCBS	CCBS_Null_x	The initial state of the CCBS service
	$\text{CCBS\_Await}_{x,y,z}$	CCBS is monitoring the subscriber x to become idle.
	$\text{CCBS\_Active}_{x,y}$	Terminal x is free, and CCBS is initiating a call from x to y.
	x: CCBS subscriber	

12 Appendix E: List of BCSM States

Below we give brief description of BCSM states. See Q.1224 for more details.

OBCSM States

State	Comments
Points in Call (PICs)	
O_Null	No call exists.
Authorize_Origination_Attempt	The originating terminal rights is being checked.
Collect_Information	The switch is collecting initial information package/dialing string from the calling party.
Analyze_Information	The information collected from the calling party is being analyzed.
Select_Route	Routing address and call type are being interpreted.
Authorize_Call_Setup	The authority of the calling party to place this particular call is verified.
Send_Call	The switch sends an indication of the desire to set up a call to the specified called party to the TBCSM.
O_Alerting	Waiting for the called party to answer the call.
O_Active	Connection established between the calling and the called party.
O_Suspended	The called party has disconnected, and the connection between the calling party and the called party is maintained by the network.
O_Exeption	Default handling of the exception condition is being provided.
Detection Points (DPs)	
Origination_Attempt	The calling party has lifted the handset and the terminal rights have not been checked.
Origination_Attempt_Authorized	The calling party's right to initiate the call has been authorized.
Collected_Information	Information has been collected from the calling party.
Analyzed_Information	The collected information has been analyzed.
O_Term_Seized	
O_Abandon	The calling party disconnects.
O_Answer	The answer indication from the TBCSM is received by the OBCSM.
O_Suspend	An indication from the TBCSM is received that the called party has suspended the call.
O_Re-answer	An indication from the TBCSM is received that the called party has re-answered the call.
O_Mid_Call	The calling party flashhooks.
O_Disconnect	Either the call is cleared from the terminating BCSM or the originating facility disconnects.
O_Called_Party_Busy	The OBCSM receives a report of the user busy event from the TBCSM.
O_No_Answer	The called party does not answer the call.
Route_Select_Failure	The call can not be routed to the called party.

TBCSM States

State	Comments
Points in Call (PICs)	
T_Null	No call exists.
Authorize_Termination_Attempt	The switch is verifying the authority to route this call to the called party.
Select_Facility	The busy/idle status of the terminating access is determined.
Present_Call	Terminating resource is informed of the incoming call.
T_Alerting	The called party is being alerted and an indication is sent to the OBCSM about this.
T_Active	The called party has accepted and answered the call.
T_Suspended	The called party has disconnected and the physical resources associated with the call remain connected.
T_Exeption	Default handling of the exception condition is being provided.
Detection Points (DPs)	
Termination_Attempt	An indication has been received from OBCSM that a call is placed to TBCSM.
Termination_Attempt_Authorized	The terminating authorization checks have been completed successfully.
Facility_Selected_and_Available	The switch determines that there is enough resource to place the call to the called party.
Call_Accepted	The call is accepted by the TBCSM.
T_Abandon	An indication is received from the OBCSM that the calling party has abandoned the call.
T_Answer	The switch detects an answer indication from the terminating facility.
T_Suspend	The switch detects that the called party on-hooks
T_Re-answer	The called off-hooks to re-answer the call.
T_Mid_Call	The called party flash-hooks.
T_Disconnect	Either the call is cleared from the OBCSM or the called party disconnects.
T_Busy	The switch determines that the called party is busy.
T_No_Answer	The called party does not answer the call.