Specification and Detection of Feature Interactions Using MSCs

Zhaoqiang Li

A Thesis
in
The Department
of
Electrical & Computer Engineering

Submitted as Partial Fulfillment of the Requirements
for the Degree
Master of Applied Science at

Concordia University
Montreal, Quebec, Canada

March, 2000

0-612-47827-0

ABSTRACT

Specification and Detection of
Feature Interactions Using MSCs

Zhaoqiang Li

New network architectures, such as the Intelligent Network (IN), have evolved in response to the changing needs and demands for advanced and sophisticated telecommunications services. However, as more services are introduced into the network, a new problem of interactions between various services/features, becomes more prominent. This problem arises when multiple services or features interfere with each other and produce unexpected results, which disturb the users.

This thesis presents my work in modeling features and detecting feature interactions using Message Sequence Charts (MSCs). The modeling technique is based on the Advanced Intelligent Network (AIN) architecture and call models. To effectively detect feature interactions, we propose an MSC feature specification style which embodies several important aspects of features directly related to feature interactions. Based on the modeling of features using the specification style, we propose a new approach for detecting feature interactions. This approach includes definitions, classification of feature interactions, and specific detection algorithms for various types of interactions. We developed a prototyped feature interaction detection tool to implement our approach. With this tool, we are able to detect many interactions described in the Bellcore feature interaction benchmark. Our detection technique has maintained its consistency and accuracy in detecting these interactions. However, some limitations of our approach prevent us from detecting certain types of interactions. Combining our feature specification style, detection approach and tool, we propose a general framework for feature specification and interaction detection for IN services.

Acknowledgments

First of all, I would like to express my deepest thanks to both of my supervisors for their indispensable encouragement, guidance and support throughout the period of this research. Dr. Ferhat Khendek has contributed a substantial portion of his time and effort in steering the research direction, holding periodic technical discussions, evaluating problem-solving strategies, reviewing research proposals as well as providing financial support to me. Dr. Rajni Patel has also made great efforts in directing and supporting my work both academically and financially. This research is carried out at Concordia University and partly funded by the National Science and Engineering Research Council (NSERC) of Canada.

Special thanks to members of our telecommunications software group, with whom I have had many fruitful discussions.

Among the people who also deserve my special thanks are Graduate Program Director Dr. Asim Al-Khalili who provided me with extensive guidance during my graduate study, graduate secretary Ms. Diane Moffat who provided me with miscellaneous help, and also technical staff of the department who maintained an excellent network environment for doing advanced studies and research.

Finally, I would like to extend my sincere gratitude and respect to my wife Jing, who has accompanied me in overcoming numerous difficulties and hardships. It is her extraordinary understanding, support and unselfish love that made the completion of this research and thesis possible.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

ACB	Automatic Call Back
AD	Adjunct
AIN	Advanced Intelligent Network
AMA	Automatic Message Accounting
ANC	Area Number Calling
AR	Automatic Recall
BCM	Basic Call Model
BCSM	Basic Call State Machine
CCAF	Call Control Agent Function
CCF	Call Control Function
CENTREX	Central Office Exchange Service
CFB	Call Forwarding on Busy
CFNoA	Call Forwarding on No-Answer
CFU	Call Forwarding Unconditional
CIC	Carrier Identification Code
COT	Continuity (message)
CPE	Customer Premise Equipment
CS	Capability Set
CW	Call Waiting
DP	Detection Point
DTMF	Dual-Tone Multi-Frequency digit collection
EBNF	Extended Backus Naur Form notation
ESC	Extended Sequence Chart
ESTELLE	Formal Description Technique based on Extended State Transition Model
FE	Functional Entity

FEA	Functional Entity Action
FMD	Follow Me Diversion
FSM	Finite State Machine
GUI	Graphical User Interface
HMSC	High-level Message Sequence Chart
IAM	Initial Address Message
IN	Intelligent Network
INCM	IN Conceptual Model
IP	Intelligent Peripheral
ISDN	Integrated Services Digital Network
ISO	International Standards Organization
ITU	International Telecommunications Union
ITU-T	ITU Technical Section (formerly known as CCITT)
LOTOS	Language Of Temporal Ordering Specification
MBS-ED	Multi-location Business Service-Extension Dialing
MDNL-DR	Multiple Directory Number Line with Distinctive Ringing
MSC	Message Sequence Chart
NANP	North American Number Plan
NAP	Network Access Point
O_BCSM	Originating Basic Call State Machine
OCM	Originating Call Model
OCS	Originating Call Screening
PCS	Personal Communication Service
PE	Physical Entity
PIC	Point In Call
POR	Point Of Return
POTS	Plain Old Telephone System

PSTN	Public Switched Telephone Network
REL	Release (message)
SCE	Service Creation Environment
SCEF	Service Creation Environment Function
SCF	Service Control Function
SCP	Service Control Point
SDF	Service Data Function
SDL	Specification and Description Language
SDP	Service Data Point
SF	Service Feature
SF	Special Form (of trace)
SFC	Special Function Call
SIB	Service Independent Building Block
SMAF	Service Management Access Function
SMF	Service Management Function
SMS	Service Management System
SN	Service Node
SRF	Specialized Resource Function
SS7	Signaling System Number 7
SSF	Service Switching Function
SSP	Service Switching Point
STP	Signaling Transfer Point
SUS	Suspend (message)
T_BCSM	Terminating Basic Call State Machine
TCAP	Transaction Capabilities Application Part
TCM	Terminating Call Model
TDP	Trigger Detection Point
TG	Trunk Group

TWC	Three Way Calling
VAD	Voice Activated Dialing

Chapter 1

Introduction

Over the past few decades, major changes in telecommunications have brought to the public an ever-wider range of services of ever-higher quality. Behind the scenes, advances in networking technology are largely responsible for these rapid changes, and the Intelligent Network (IN) architecture, first proposed by Bellcore in the 1980's and standardized by the International Telecommunications Union (ITU), is perhaps foremost among the advanced telecommunications architectures deployed today.

In order to manage and control telecommunication services (or features, as we shall call them hereafter) with maximal efficiency, Bellcore designers put the logic governing these features in computer nodes distributed throughout the entire network rather than in the switching element, thus allowing the architecture to function independently of the individual features. Major telephone companies adopted IN rapidly and, as a result, network operators began to develop and control features more efficiently, rapidly introducing new and sophisticated capabilities to the network and customizing these to meet individual needs of customers.

However, with more and more services and features introduced into the network, the problem of Feature Interactions becomes more and more prominent. Feature Interaction represents the phenomenon that different telecommunication features affect each other or the underlying service. Feature Interaction is a problem, though not one specific to IN features, and as IN facilitated the introduction of new features and deregulation brought in a host of new players in service creation, deployment, operation and management, the problem grew in prominence.

This thesis deals with the detection of feature interactions rather than the study of strategies for their resolution or avoidance. Our approach uses a formal description language MSC and formal methods. This not only facilitates common understanding of feature behavior, but also permits automation based on unambiguous interpretations.

While MSC provides sufficient mechanisms for describing a feature, different users may have different ways of using the language, resulting in very different specification styles and greater difficulty in automation. Instead of allowing the user to decide how to specify a feature, we propose a feature specification style in MSC that both facilitates automation and ensures some level of user-friendliness. Based on this specification style, we classify feature interactions into three categories: Blocking of Initial Triggers, Interaction between Traces and Violation of Static Requirements. We develop techniques and algorithms to detect each of the three categories of interactions. This study is based on the Advanced Intelligent Network (AIN) architecture and Basic Call Models. However, the principles derived here can be adapted easily to other network architectures.

1.1 Network Evolution

Prior to the 1980's, as Bell Atlantic [1] recalls, the network consisted of an assemblage of large switching systems and the telephone terminals connected to them (Figure 1-1). Switch vendors, for the most part, decided which services can be offered and how they were to be implemented and deployed into the network. New services could be introduced only when network operators had met with switch vendors, discussed the types of services customers required, negotiated the switching features that provided the services, and finally agreed on a generic release date for feature availability. When such a service was finally in place, the operators could not easily modify it to meet individual customer requirements. Often, the network operator had to negotiate the changes with the switch vendor and, as a result, it took years to plan and implement services. Understandably, a network operator could only offer a few basic services to business and residential subscribers.

During the mid-1980s, the market for traditional telephony services reached saturation point. In order to increase traffic volume and so generate additional revenue, operators needed to offer new services that would meet the expectations of ever more sophisticated subscribers. Demands for these advanced network services began to exceed the network capacity. Then, with deregulation of the industry in the U.S., start-up operating companies joined established overseas operators to raise the level of competition in the global telecommunications market. There was a scramble to protect market share and profitability as operators strove to offer subscribers a wider range of services more quickly and cost effectively than their competitors.

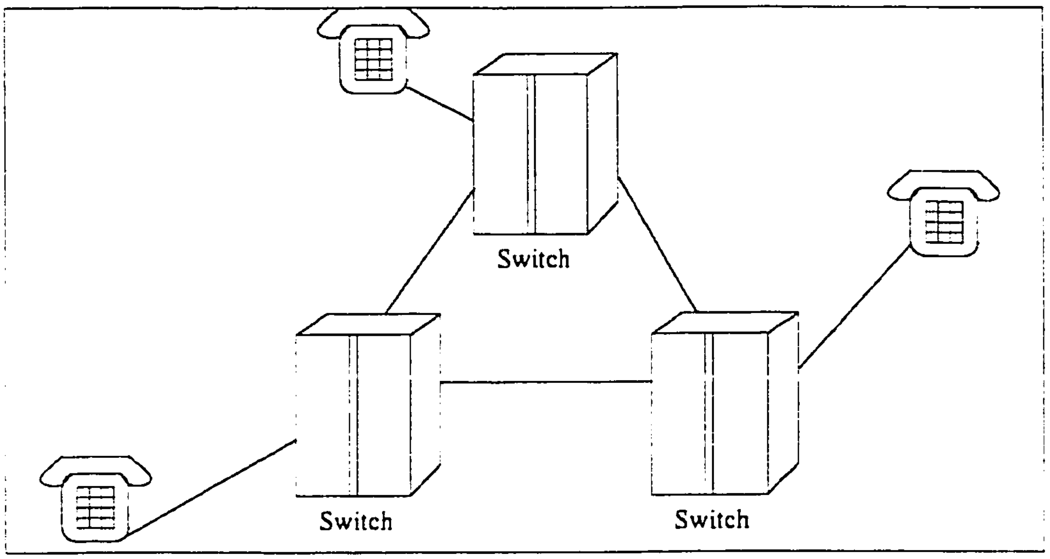


Figure 1-1: Plain Old Telephone System (POTS).

In response to the needs of operators to improve existing services and expedite the development of new services, Bell Communications Research (Bellcore) introduced the concept of Intelligent Networks[1]. In 1989, the International Telecommunications Union (ITU) incorporated ideas from the AIN work done by Bellcore, together with ideas from European operators and vendors, and proposed a first standard IN architecture called CS-1. Since then, Bellcore and ITU-T have coordinated their development efforts.

The introduction of IN satisfied the needs of operators to develop and deploy services rapidly, to obtain vendor independence and to open interface for service creation. Since IN follows an incremental approach that provides a feasible solution for service providers, telephone companies all over the world have deployed it as the standard architecture for telephony systems. In Chapter 2, we will examine IN architectural concepts, particularly AIN concepts introduced by Bellcore.

[1] Later, Bellcore changed this concept to Advanced Intelligent Networks (AIN), as we shall call it hereafter.

1.2 Telecommunication Features

Although there is a trend within the telecommunications industry today towards making clear distinctions between services and service features, as pointed out in the SCORE project [2], the words "service" and "feature" are not yet used consistently.

Bellcore researchers [3, 5] suggest that "a network provides one (or more) services", and that "each service can be augmented with several features. For example, the Plain Old Telephone System (POTS) is a service on which Automatic Call-Back and Call Waiting are features." For the remainder of this thesis, since we have not addressed interaction between "services" as defined by Bellcore, we will only speak about "feature interaction".

As an example, we illustrate the telephone feature "Area Number Calling" (ANC) in Figure 1-2. This feature is useful for companies or businesses that advertise one telephone number but want their customers calls routed to the nearest or most convenient business location. In IN architecture, the feature logic is located in the Service Control Point (SCP). The SCP uses feature data (e.g. ZIP codes) to make a match between the calling telephone numbers and their geographical locations, then the Service Switching Point (SSP) routes the call to the company or business location that is closest to or most convenient for the calling party.

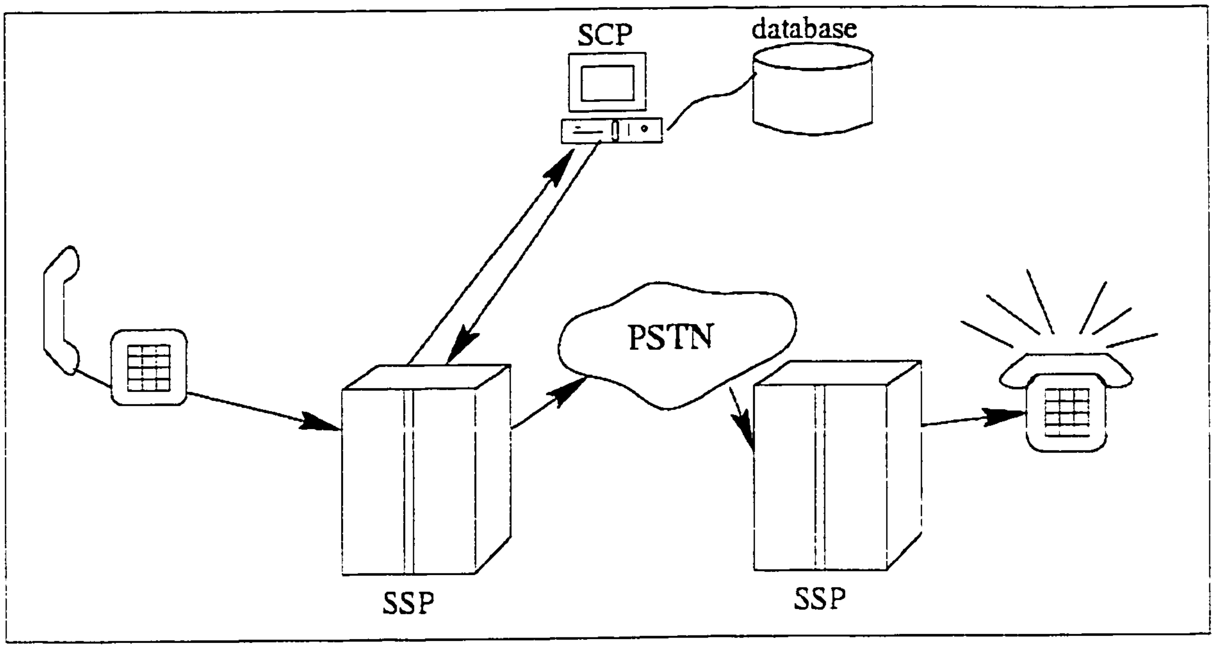


Figure 1-2: ANC Service.

1.3 The Problem of Feature Interactions

As operators introduce more features into networks and feature interaction problem becomes more prominent, they can be expected to give rise to major difficulties in understanding and mastering how features cooperate and interact, how they share resources and what the resulting behavior of the network will be. Each new feature has the potential to interact negatively with existing features, annoying customers or, in the worst scenario, causing total system breakdown. Muller et al. [6] has described how difficult it will become to assess a priori that:

- For the customer (subscriber or end-user), the features shall behave properly in regard to information received from the network (expected to be correct and user-friendly), information sent to the network (expected to be taken into account), waiting time (expected to be short) and results (expected to match each feature requirements).

- For the service provider, the features shall remain available, reliable and in accordance with user requirements, data shall remain consistent, and the call records shall permit the subscriber to be charged for the use of these features.
- For the network operator, the services shall not create any deadlock or abnormal resource consumption in the network and charging should be consistent.

To date, we do not have a standardized formal definition of feature interactions. There are, however, several formal and informal definitions in use today that partly describe the nature of the problem. In the introduction to the proceedings of the Second Feature Interaction Workshop [4], a feature interaction is defined in this way:

> "*A feature interaction occurs when the behavior of one feature is changed by the behavior of another. In many cases, this can lead to unexpected or undesired behavior which affects the quality of the service provided to telecommunications users.*"

This definition suggests that some interactions may be expected and desired while others may be unexpected or undesired.

In our thesis, we use the following informal definition:

> *A feature interaction is said to take place if the behavior of any one of the participating features is not preserved.*

This definition arises from our methodology, that is, checking for the preservation of feature behaviors. We do not distinguish between expected or unexpected interactions and additional actions would have to be undertaken to detect harmful interactions.

1.4 Formal Methods for Detecting Feature Interactions

Our description of formal methods will follow closely that provided by Clarke and Wing [7]. In [7], Clarke and Wing describe formal methods as "mathematically-based languages, techniques, and tools for specifying and verifying software or hardware systems". They acknowledge that formal methods do not guarantee absolute correctness, but assert that their use can increase our understanding of a system "by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go undetected".

For our purposes, formal methods consist of "formal specification" and "formal verification".

Formal specification describes a system and its desired properties using a language with mathematically-defined syntax and semantics. This language allows us a deeper understanding of the system being specified, and using such a language, developers uncover "design flaws, inconsistencies, ambiguities and incompleteness"[7].

Formal verification confirms that the design meets various correctness and quality standards. During the verification process, testers can detect subtle design errors using automatic model checking or theorem proving, thus providing evidence for safety and quality assurances.

Formal methods provide a potential solution to feature interaction detection for three reasons [2]:

- Formal description techniques provide precise and unambiguous descriptions of features. This makes a comprehensive and reliable approach to interaction detection possible. Formal verification techniques open up the possibility of automating the detection method.
- Formalization allows a better understanding of the real expected requirements of the behavior of features. As part of this process, as we achieve a better understanding of the informal definition of features, many interactions can be detected, leading to the redefinition of features.
- We expect the number of features introduced in networks to grow rapidly, and manual approaches to interaction detection will become increasingly inadequate. Formal specification allows the automation of interaction detection and reduces the severity of the combinatorial problem.

Researchers currently use many formal approaches to detect feature interactions. In Chapter 3, we will present several of these formal approaches that are related to our own approach.

1.5 Summary of Our Approach

Our approach employs formal description techniques to specify feature behavior. Every system may be understood at different levels of abstraction and, although our specification explores the feature behavior mainly at the requirement level, we require additional information about the feature in question. This requirement differentiates our approach from several related approaches that use requirement specifications only.

1.5.1 Underlying Methodology

The principle behind our approach is simple: to detect a feature interaction, we must take into consideration several important properties of features, namely, triggers, traces and static requirements. Otherwise, we risk detecting false interactions or failing to detect existing interactions, as happens in the case of several related approaches.

According to IN terminology, a **trigger** is a special mechanism on the Basic Call Model that, when activated, results in suspension of normal call processing and invocation of feature-specific logic.

The **trace of a feature** is defined as the complete sequence of transitions on a specific IN Call Model during the lifetime of the call and when the feature is enabled. A feature may contain several traces, each corresponding to an IN Originating or Terminating Basic Call State Machine (BCSM).

A **static requirement** is a restriction on the occurrence of certain events or combinations of events or a restriction preventing the call processing from reaching a certain system state. As an example, a common static requirement on almost all features is that callers are not allowed to connect to themselves.

In our approach, we design a specification style for features that take these three properties into consideration. We classify feature interactions according to these properties, and we develop techniques to confirm the preservation of these properties and, in so doing, we detect feature interactions.

1.5.2 Feature Specification Style

A good specification language is an important prerequisite for the implementability and applicability of an approach. Due to the popularity of Message Sequence Charts (MSC) [8] in the telecommunications industry and the rich structural concepts defined in it, we choose MSC as our specification language. However, this does not satisfy all our needs. Since MSC is a rich language, different investigators can have very different specification styles for the same feature behavior. To facilitate automation and common understanding, we have designed a style for feature specifications that not only allows developers to specify feature properties, but helps provide developers with a simple template to accelerate the process.

This specification style consists of two parts: the structures for feature specifications, and the naming conventions for all structures involved in the specification. We selected the structure for feature specifications based on the Advanced Intelligent Network (AIN) architecture and Basic Call Model (BCM). By making an intuitive mapping between the elements present in the AIN structure/call model and the MSC structural concepts, we are able to represent feature behavior. However, since the naming of the structures is still subject to interpretation, we designed the naming conventions, which help the feature specifiers communicate with each other and with the tool. Similar to creating a new programming language on top of the specification language in use, the naming conventions define a set of reserved keywords and syntax for many structures.

With the help of the structures and their naming conventions, we can ensure that the information contained in any specification is unique as long as the specification follows our specification style.

1.5.3 Classification of Feature Interactions

In our approach, we define feature interaction as "failure to preserve any kind of feature behavior". We measure the preservation of feature behavior using three feature properties: trigger, traces, and static requirements. Following this definition, we classify feature interactions into three categories:

i. Blocking of triggers: This happens if the trigger of any feature is blocked.
ii. Interaction between traces: This happens if the internal traces of any feature are modified.
iii. Violation of static requirements: This happens if static requirements on a feature, e.g. restrictions on certain connections, are not respected.

1.5.4 Summary of Detection Algorithms

1. Internal Representation of Feature Behavior

Let us assume we have feature specifications in a high-level language like MSC. To detect feature interactions, we extract the useful information contained in the specifications and transform it to a machine-friendly form such as the Finite State Machine (FSM) model.

An FSM is a state-transition model which can be used to model dynamic system behavior. Every FSM consists of states, and transitions that link the states. In addition, events can be specified on the transitions to indicate what triggers the transition.

Using the FSM representation, we develop detection algorithms for each type of feature interactions.

2. Algorithm for Detecting the Blocking of a Trigger

We identify some situations where the blocking of a trigger will occur, some situations where it will not occur and some situations where no interaction can happen. We base this identification on observations of trigger criteria, trigger positions and the positions that traces of a feature pass by (as described in the AIN Basic Call Model). The algorithm simply simulates this identification process and tells us if it finds the blocking of a trigger. Where there is no blocking of triggers, the algorithm guides us as to what steps we should take to detect other types of interactions.

3. Algorithm for Detecting Interaction between Traces

To detect this type of interaction, we must first use FSMs to represent and identify the overlapping parts of the two traces. These overlappings are the only parts where interaction between traces can happen and, once we have identified these overlapping parts, peer traces (i.e. traces which correspond to the same Basic Call State Machine) have already synchronized in the first states of their FSM models.

Next, we must examine how each trace follows its transitions and identify those divergences that might lead to non-preservation of transition behavior. To do this, we develop an algorithm that simulates a negotiating process for traces. We develop simple rules that define agreement, disagreement, and determine, in the case of agreement, what is the agreement trace.

For instance, to determine the agreement between peer traces, we must examine each pair of peer transitions. We show that, for any pair of peer transitions, we must consider two properties: the "next state" and the "actions taken to reach the next state". When we consider the "next state" property, we must take into account the fact that, in many cases, a trace jumps to an earlier state simply to collect extra information, then comes back to the current state when that task is complete. If, as a result, the pair of transitions under study have different next states, we do not consider this as a divergence of destinations, since the call processing tends to follow the backward transition. When it returns, all requirements of the other trace (i.e. the trace without the backward transition) is preserved. Therefore, we consider this type of backward transition <u>not</u> as a transition but as an "action taken to reach the next state ". We then propose a special form of trace, one in which we define "transition" as the giant step by which the current state reaches the state which, according to AIN Basic Call Model, comes later than the current state. We treat all contributing events, including backward transitions, simply as actions on the giant step. Then, in this special form of trace, we can prove that two transitions agree on their next states (i.e. there is no divergency on transition destinations) if and only if their next states are the same.

For transition actions, we develop a similar algorithm to the algorithm for traces. This algorithm takes each action as a state. Between the states, there are no actions and it is thus relatively easy to decide the agreement. However, we change the rules for deciding agreements to reflect the nature of actions.

Pairs of peer transitions are examined one pair at a time. Meanwhile, the agreement trace is generated gradually based on transition agreements. However, if any time during the process, a disagreement is encountered, then two traces disagree and we detect an interaction between traces.

4. Algorithm for Detecting Violation of Static Requirements

Once we have completed the algorithm for detecting interaction between traces and found no interactions, we forward the generated agreement trace to this algorithm to detect any violation of static requirements. These usually take the form of a pattern that should not appear in the combined trace. We therefore perform a search on the combined trace for the violation pattern. If we find the pattern, we detect a violation of the static requirement. Otherwise, there is no interaction detected at this level, although there might still be interactions at lower levels.

1.6 Organization of the Thesis

The rest of the thesis is organized as follows:

Chapter 2: Intelligent Networks

In Chapter 2, we provide a brief description of basic IN concepts and architectural models. We pay more attention to Bellcore AIN concepts. We describe concepts of IN that are directly related to our detection techniques in greater detail.

Chapter 3: MSC and Feature Interaction Detection

In Chapter 3, we provide a brief tutorial on the formal description technique, the MSC language. As in Chapter 2, we will put greater emphasis on concepts in MSC that are directly related to our feature specification style. We then describe some of the related feature interaction detection techniques.

Chapter 4: Feature Specification Style

Chapter 4 provides a detailed description of our specification style in MSC. In this chapter, we will first introduce some of the basic concepts related to feature interaction detection. We then present an intuitive mapping of these concepts to MSC structures. This provides us with the means to represent these concepts using MSC structures. To eliminate the possibility of ambiguity, we develop naming conventions for these structures and, as a conclusion to the specification style presented in Chapter 4, we provide some comments on the specification style.

Chapter 5: Classification and Detection of Feature Interactions

In Chapter 5, we present our main contribution, addressing our methodology and techniques for detecting feature interactions. This chapter begins with the classification of different categories of interactions. We then derive the detection techniques for each category of interactions. To facilitate automation, we also provide algorithms for implementing all the detection techniques. We provide several examples to illustrate our approach.

Chapter 6: Tool and Application

We start this chapter with a brief introduction to the architecture of our feature interaction detection tool prototype. We then illustrate the strengths and explore some of the weaknesses of our approach by presenting a comprehensive benchmark. This benchmark is based on the complete Feature Interaction Benchmark presented by Bellcore. We test our tool with this benchmark.

Chapter 7: Conclusion

As a conclusion of the thesis. Chapter 7 summarizes the main contributions of our work and points toward several directions for future development.

Chapter 2

Intelligent Networks

In this chapter, we introduce the concepts of IN. We lay greater emphasis on the Bellcore AIN concepts since they are more directly relevant to the specification and detection techniques presented in this thesis. However, since Bellcore AIN is highly consistent and aligned with ITU-T IN, most of the terms introduced in this chapter apply to both Bellcore AIN and ITU-T IN. In the rest of the chapter, without further indication, we use the term IN to denote both Bellcore AIN and ITU-T IN.

In the first section of the chapter, we introduce the goals behind the IN. Then, we introduce the different models involved in IN: the architectural model, the conceptual model and the call model.

2.1 IN Goals

As we have seen in Chapter 1, the demand for new network services and features led directly to the creation of IN, and a fundamental goal of IN is to support the rapid creation and provisioning of services and features for the end customer. As Duran [9] points out, IN is designed to "allow the introduction of new capabilities in the telecommunications network and to facilitate and accelerate in a cost-effective manner, service implementation and provisioning, in a multivendor environment." In the Bellcore AIN releases (and similarly in ITU-T IN recommendations), the following objectives are addressed:

- Rapid creation of new services, from conception to deployment.

- Broader range of services: IN should go beyond traditional voice and data bearer services to a much broader range, including information services, broadband and multimedia bearer capabilities.
- Applicability to all telecommunications networks, e.g., Public Switched Telephone Networks (PSTNs), Integrated Services Digital Networks (ISDNs), Packet Switched Public Data Networks and Mobile Networks.
- Independence from network-specific developments by equipment suppliers: IN should allow service-providers to define, implement and efficiently manage their own services independently from switch vendors. In addition, IN should enable a multivendor, competitive environment and ensure that the services will work correctly and consistently on any vendors equipment, and across equipments of several vendors.
- Maximum compatibility with existing networks: IN should be introduced starting from existing networks. Otherwise, it will not be feasible for service providers.
- Evolution to reflect implementation experiences, new technological opportunities and market evolution.

To meet these objectives, Bellcore has proposed a generic requirement for an incremental approach to the definition and development of the AIN. Each increment, referred to as a release, represents the set of marketable and maintainable AIN capabilities expected to be available in a particular time frame. Each AIN release is intended to be backward compatible to the previous release and on an evolutionary path towards the target AIN architecture.

2.2 IN Architectural Model

2.2.1 Evolution of IN Architecture

Before the introduction of IN, as Black [10] points out, switches distributed throughout the network contained not only the network intelligence but the basic call processing and database processing capabilities (see Figure 2-1).

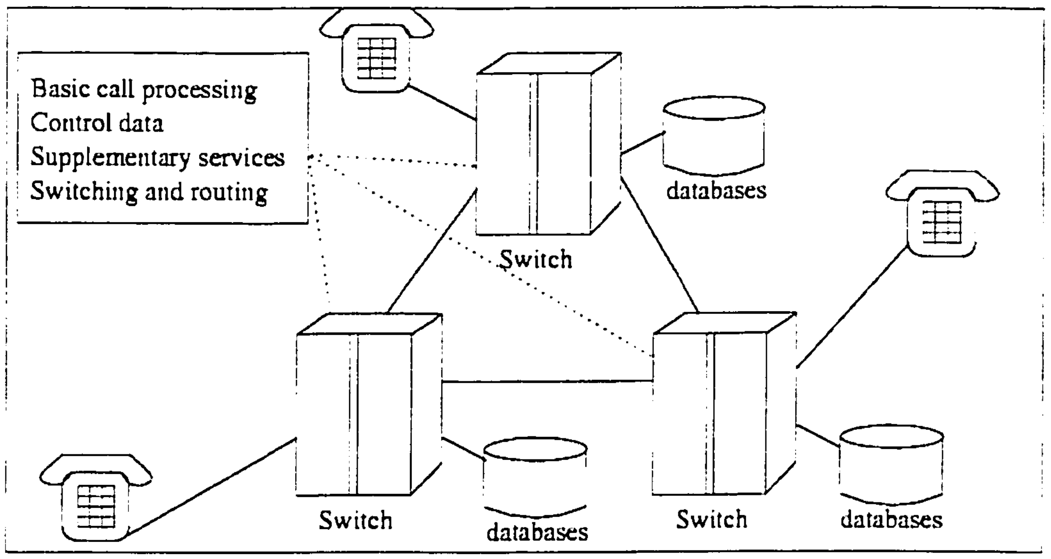


Figure 2-1: Early Network Architecture.

There was no overarching model for these switches, originating as they did with various manufacturers and vendors, each of whom had their own ideas as how to use these switches. A good deal of time and money was spent to update the software and database contents across this heterogeneous switching environment. Since each switch was redundantly configured with identical control databases and software, change meant maintaining multiple copies of this data and supporting software at every node. To make matters worse, as Black explains, if a service provider decided to offer a new service, say call forwarding, developers wrote software from the bottom up solely for that service. The vendor was forced to change the basic call processing modules.

Change to these large systems was cumbersome and complex, not surprisingly, vendors found it hard to respond quickly to their customers' requests, hard to make timely releases to software, and even harder to build new services. Bell researchers [1] state that two to four years were not unusual for the latter task.

Network architecture was subsequently re-thought and the result was the basic intelligent network architecture, shown in Figure 2-2.

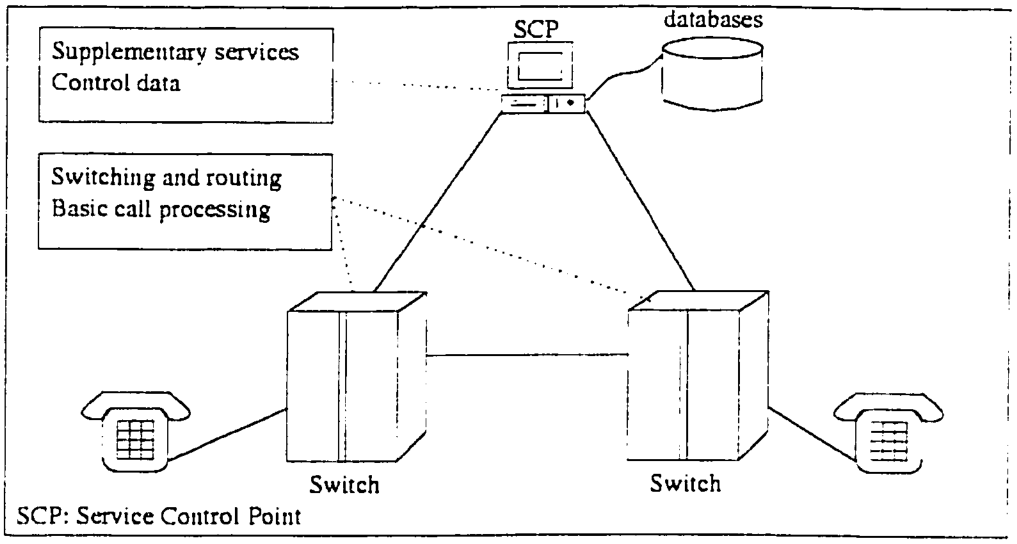


Figure 2-2 Basic Intelligent Network Architecture.

In this model, user services, rather than residing redundantly at each switch, are handled by only a few processors — sometimes just one — and are available to many users. In the example shown in Figure 2-2, the single databases module is linked to the Service Control Point (SCP) and so accessible to the entire network. The SCP is an example of the specialized machines that are now introduced to handle various functions.

Although, in Figure 2-2, we are already looking at a rudimentary IN architecture — what Black calls the "service-independent environment" that was to replace the "service-specific environment", developers went on to introduce IN-specific components — the Service Creation Environment (SCE), the Service Management System (SMS) the Intelligent Peripheral (IP), the Adjunct (AD), and the Network

Access Point (NAP) — that would enhance networks by permitting the rapid creation and efficient maintenance of new services. Their position in the IN topology is illustrated in Figure 2-3.

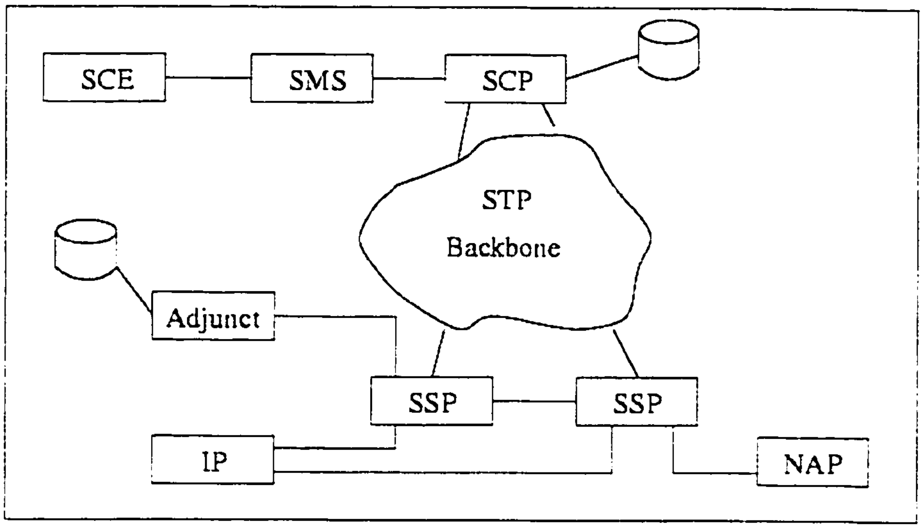


Figure 2-3 AIN Components.

2.2.2 IN Physical Components

Here is a list, drawn extensively from Black's book [10], of the network components with their basic functions:

Service Switching Point (SSP)

The SSP continues to play the role it had in earlier systems, represented by Figure 2-1: it is the central element for switching resources, for signaling and for interfacing with other network components. As the point of entry for all IN services, it must provide mechanisms to detect user requests for IN services while interacting with other IN components, such as the SCP, to carry out these requests.

Service Control Point (SCP)

The SCP is connected to the SSP by the SS7 network. It contains service logic programs and associated data that enable the IN services and carries out the network-based transaction processing.

Service Creation Environment (SCE)

The SCE provides design and implementation tools that enable developers to create and modify services in the SCP.

Service Management System (SMS)

This database management system manages the master database that controls the IN customer services, including the continual processes of database maintenance, backup and recovery, log management, and audit trails.

Intelligent Peripheral (IP)

The application-independent IP is connected to one or more SSPs. It supports basic services like tone generation, voice recognition, playback, compression, call control, record and DTMF (Dual-Tone Multi-Frequency digit collection) detection and collection.

Adjunct (AD)

Adjunct performs the same operations as an SCP, but it is configured for a single switch where a few fast-response services are required through a high-speed link.

Network Access Point (NAP)

The NAP, a switch without IN functions, is connected to an SSP, and interfaces to trunks with SS7 messages or frequency tones. Based on the called and calling number received at the NAP, it may route the call to its attached SSP or IN services.

Signaling Transfer Point (STP)

The IN STPs perform two functions beyond their usual operations. First, they can employ pseudo-addressing that enables them to balance the load between two or more SCPs. Second, they can employ alternate routing in the event of a problem in the network.

2.2.3 IN Functional Entities and Physical Nodes

Figure 2-4 shows the IN physical nodes and their functional entities. Again following Black's book [10], we can summarize these entities as follows:

Service Switching Function (SSF): This function:

- recognizes calls that require IN service processing and interacts with call processing and the service logic to deal with these calls.
- generates functions that mediate interactions between the CCF and the SCF when associated with the CCF.
- further increases the capability of the CCF to recognize IN service control triggers.
- manages the signaling between the CCF and the SCF and, if required, modifies call processing functions in the CCF to handle requests for IN-provided service usage under control of the SCF.
- is managed by the SMF.

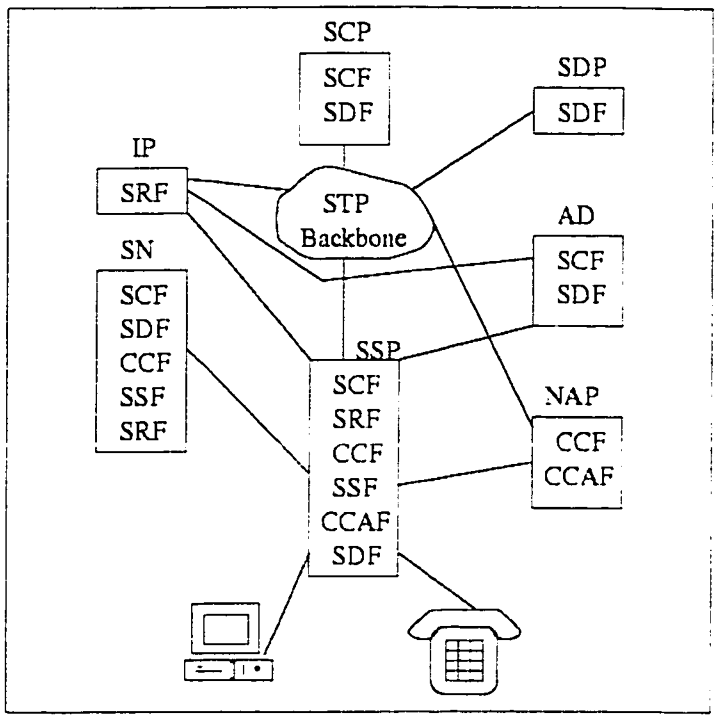


Figure 2-4: IN Physical Entities and Functional Entities.

Call Control Function (CCF): This function:

- may establish and control bearer services on behalf of network users.
- may deal with requests from the CCAF to establish and dismantle connections. associating the CCAF functional entities to each connection instance.
- participates in trigger operations (by passing events to the SSF).
- is managed by the SMF.

Call Control Agent Function (CCAF): This function:

- provides users with access to the services.
- represents the users to call processing
- represents the interface between the user and the network control functions.
- interacts with the user for activities pertaining to an IN operation.
- accesses the CCF's service-providing capabilities for call or an IN service.
- relays CCF information about the call or service back to the user.

Service Control Function (SCF): This function:

- contains IN service logic that provides the logical control applied to a call that entails an IN service.
- handles service-related processing activities such as analysis, translation, screening, and routing.
- interacts with other functional entities to obtain information as required to process a call or IN service.
- is managed by the SMF.

Service Data Function (SDF): This function:

- handles service-related and network data.
- provides the SCF with a logical view of the data.
- contains data that relates to the provision or operation of IN service and may include access to user-defined service-related data.
- is managed by the SMF.

Specialized Resource Function (SRF): This function:

- provides end-user interaction with the IN by providing control over resources such as DTMF receivers, voice recognition capabilities, protocol conversion, and announcements.
- is managed by the SMF.

Service Management Function (SMF): This function:

- provides the service provisioning, deployment, and management control.
- allows access to IN functional entities for the transfer of information that relates to service logic and service data.

Service Management Access Function (SMAF): This function:

- controls access to service management functions.

Service Creation Environment Function (SCEF): This function:

- supports the creation, verification, and testing of new IN services.

As shown in Figure 2-4, an SSP contains a CCF and an SSF and, if it is acting as a local exchange, it also contains a CCAF. As options, it may contain an SCF, an SRF and an SDF.

The NAP is not a fully functional IN node, but contains only the CCAF and CCF functional entities.

The SCP contains an SCF and an SDF and can sometimes access data in a service data point (SDP).

The SDP contains the SDF and the customer data that is usually accessed during the execution of an IN service.

The IP provides a number of specialized services in the IN network, for example: voice recognition, announcements etc. and, as might be expected, contains the SRF.

The adjunct (AD) is a physical entity that is equivalent to an SCP, and contains the same functional entities as an SCP.

The service node (SN) provides interactions with users and controls IN services. It communicates directly with SSPs and contains an SCF, SDF, SRF and an SSF/CCF. In the SN, the SSF/CCF is not accessible by external SCFs and is closely coupled to the specific SCF operating within the SN.

2.3 IN Conceptual Model (INCM)

The INCM is a term introduced in the ITU-T IN, though many of the concepts were later adopted by Bellcore AIN. The INCM represents the entire IN process. This model is published in ITU-T Recommendation [11] Q.1201, with supplementary information in Q.1203, Q.1204, and Q.1213. For the development of so-called IN capability sets (CS), the model is highly useful.

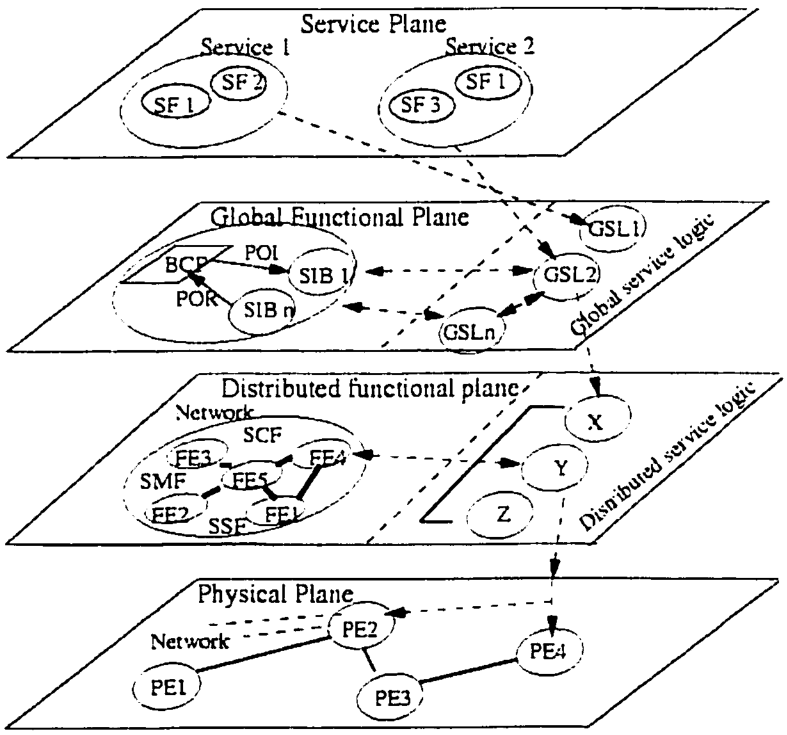


Figure 2-5: IN Conceptual Model (INCM).

As shown in Figure 2-5, there are four planes defined in INCM. Each of the four INCM planes provides a schematized view of IN capabilities (see Figure2-5).

1. The **service plane** depicts the IN process from the perspective of network services. It does not attempt to address implementation of these services.

Within this plane, the model groups various services (e.g. credit card calling) into units called "service features" or "SF" (e.g. user authentication).

2. The **global functional plane** represents the functions of an IN network considered as a single entity. This plane contains the call-processing model and depicts the service-independent building blocks (SIBs), which global service logic combines to build service features in the service plane, above.

3. The **distributed functional plane** models the distributed functions of IN, defines the functional entities (FEs), describes the actions of FEs (FEAs) in those cases where an FE represents a grouping of related FEAs, and describes the relationships between FEs.

4. The **physical plane** represents the physical IN-structured network, depicting the various physical entities, the functional entities embodied within them and the protocols by which the physical entities communicate with one another.

The arrows in Figure 2-5 show the pathways along which the various planes of the model interact. The service features in the service plane are realized in the global functional plane with global service logic and SIBs. The SIBs in the global functional plane are present in the FEs in the distributed functional plane, and a SIB may be instantiated in more than one FE. The FEs in the distributed functional plane are mapped to the PEs in the physical plane. Each PE may contain one to several FEs.

2.4 IN Call Model

The call model represents the sequence of procedures executed by an IN to set up, manage and clear an IN session between IN components. The call model allows both ends of a session, regardless of the specific vendor's machine, to share a common view of the on-going phases and operations of the IN operation. As Duran [9] points out, it defines "the sets of states (or Point in Call, PICs), transitions, and detection points, which are used to illustrate the different states that a call can go through, from origination (the user picks up the phone) to termination (the user hangs up)."

In the sections that follow, we will attempt to define and expand on the different terminology employed in the Bellcore AIN call model. Since the Bellcore AIN call model differ only slightly with the ITU-T IN call model in terms of terminology, most of these terms will apply to the ITU-T IN call model as well.

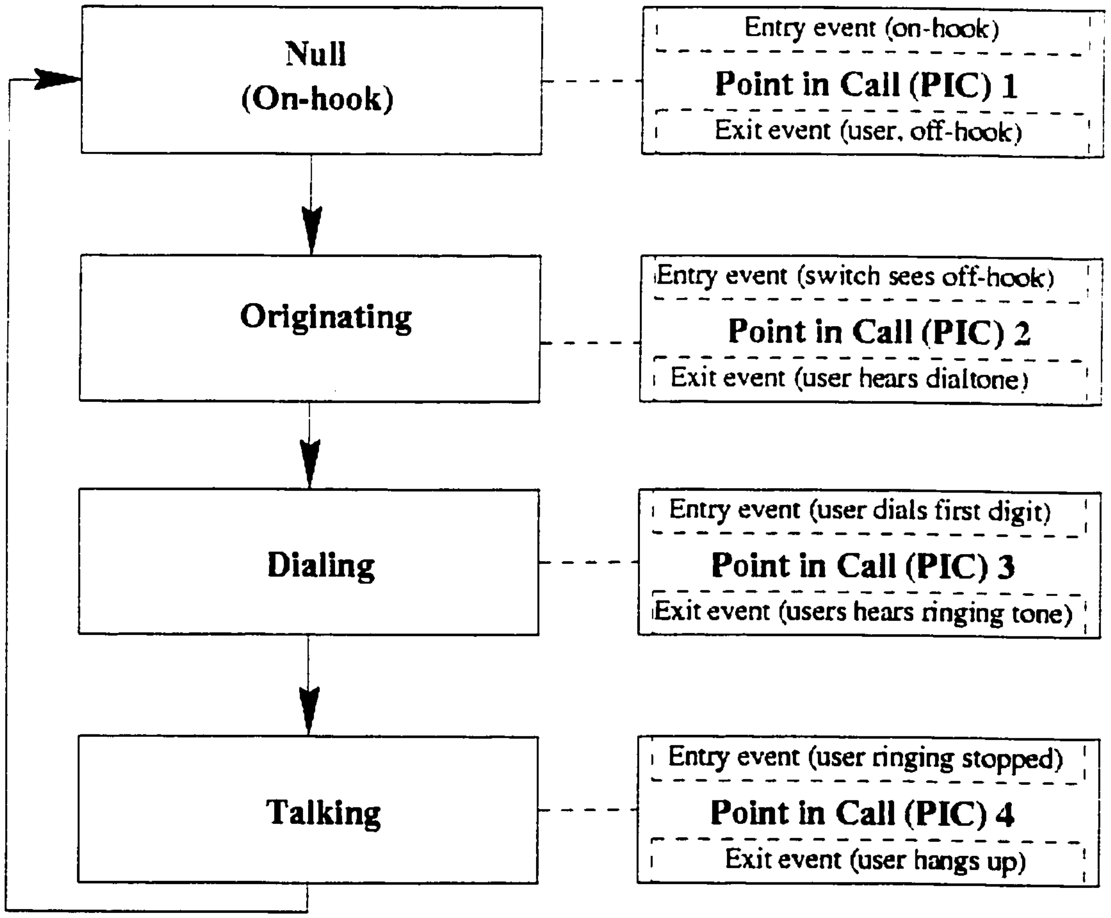

Null
(On-hook)
Entry event (on-hook)
Point in Call (PIC) 1
Exit event (user, off-hook)
Originating
Entry event (switch sees off-hook)
Point in Call (PIC) 2
Exit event (user hears dialtone)
Dialing
Entry event (user dials first digit)
Point in Call (PIC) 3
Exit event (users hears ringing tone)
Talking
Entry event (user ringing stopped)
Point in Call (PIC) 4
Exit event (user hangs up)

Figure 2-6: Basic Call Model (BCM).

2.4.1 Basic Call Model

Figure 2-6 depicts the basic call model. As a high-level finite state machine, it is organized around actions (and aggregations of actions) called Points In Call (PICs) that divide the entire call processing into phases. We shall see in later chapters how we use these phases to identify the activation period of a feature. In addition to internal actions performed in a PIC, a PIC contains also an entry point and an exit point. These are the points where call processing is directed in or out of the PIC node.

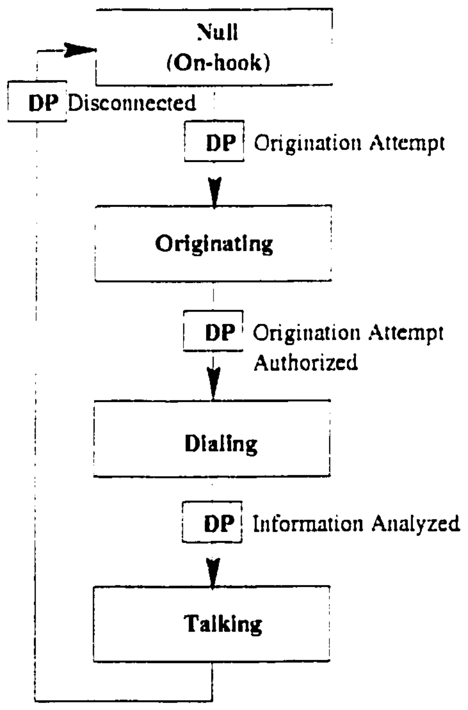


Figure 2-7: Detection Points (DPs).

2.4.2 Detection Points (DPs)

"Detection points", as Black [10] describes them, "operate between the PICs to delineate the points in the model where call processing is suspended and other actions are invoked; for example, the action of sending a message to another node. Query messages are associated with specific DPs and when a specific query message is received by a node, it knows the exact stage of a call that has been completed in the transmitting node."

As illustrated in Figure 2-7, a DP is located in between the PICs (acting as transition points) in the call model.

2.4.3 Triggers

DPs are associated with triggers. A DP containing at least one trigger is called a Trigger Detection Point (TDP). A trigger lists a set of criteria that contain a condition that must be satisfied before a message is generated. In addition, the trigger must include the address of the node that will receive the message.

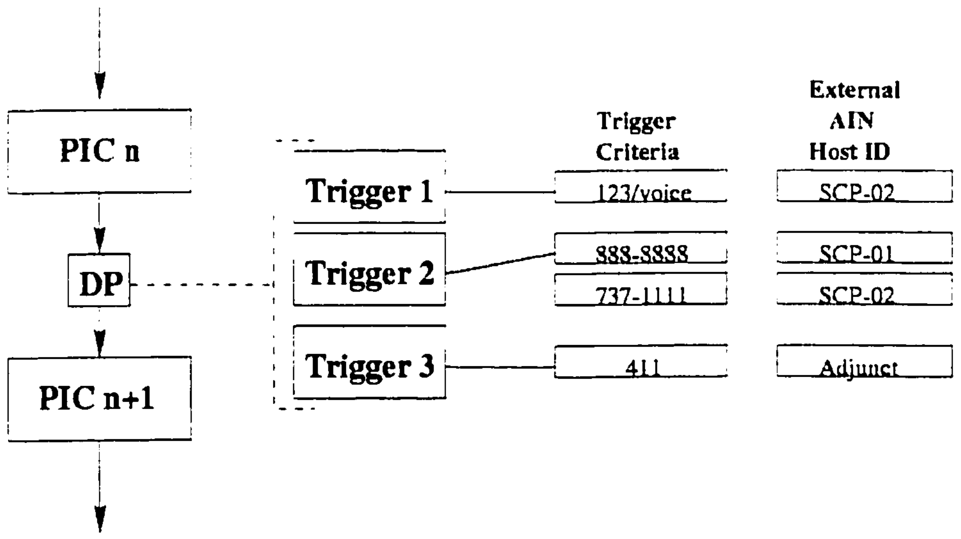


Figure 2-8 Triggers and Detection Points.

A combination of DP and trigger criteria, if satisfied, suspends call processing at the node (Figure 2-8). This suspension triggers the creation and sending of a message to the relevant recipient node. Operations remain suspended until a response returns from the remote node. If none of the trigger criteria are satisfied, call processing proceeds directly to the next PIC.

2.4.4 Originating and Terminating Call Models

The call model divides into two parts: the Originating Call Model (OCM) and the Terminating Call Model (TCM). Both models describe a state machine for the call processing logic. As Black explains, "An AIN operation begins with the invocation of the OCM, then the TCM is started based on the satisfaction of trigger criteria. Once the TCM starts, both the OCM and the TCM operate in parallel, although the specific instance of the OCM may be suspended when the TCM is operating."

The OCM initiates the call, performs call validation of the calling party, and controls the call to its completion. The TCM validates the called party and terminates the call.

2.4.5 Examples of AIN Operations

The call model defines the exchange of information between a Service Switching Point (SSP) and a Service Control Point (SCP). As figure 2-9 depicts, an incoming call is processed through the Originating Call Model (OCM) module by executing the state logic with Points in Call (PICs) and Detection Points (DPs). This figure illustrates the basic events required to evoke trigger operations. In event 1, the SSP receives information (i.e., dialed digits etc.). In event 2 the SSP executes the PICs until (for example) the "information analyzed" detection point is reached, invoking the trigger logic (depicted in event 3). The SSP now analyses the various trigger criteria and, in event 4, trigger 3 and its criteria are satisfied.

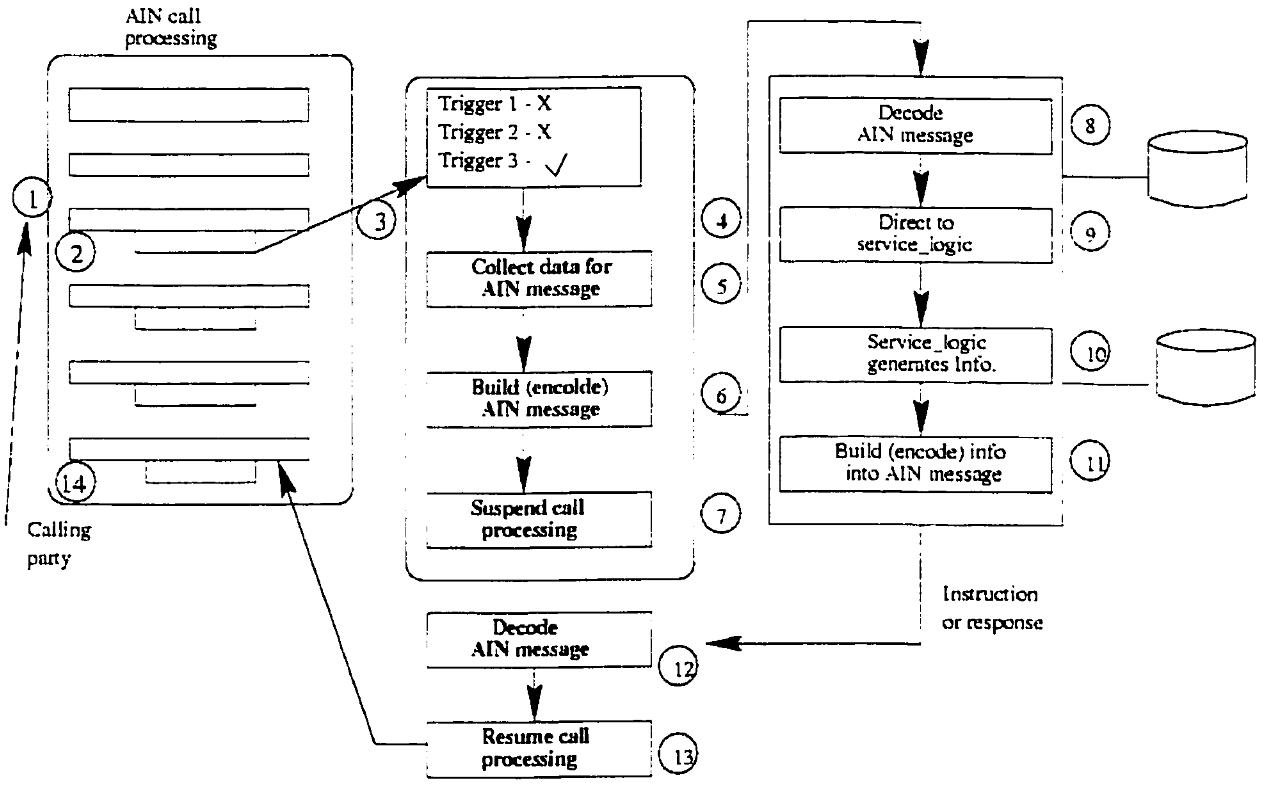


Figure 2-9: Example of AIN Operation.

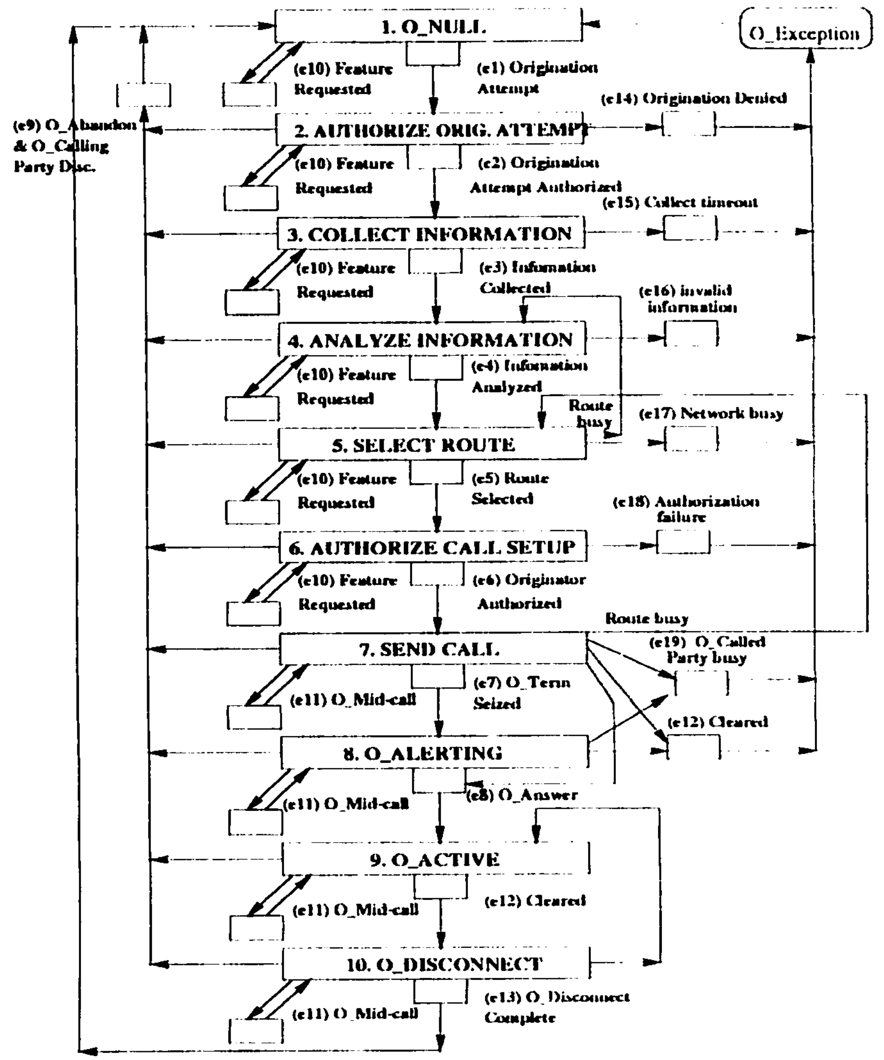
If the trigger 3 criteria are satisfied, the AIN node now assembles the required information and builds a query message, as depicted in Figure 2-9, that contains all the fields necessary to identify the message (addresses, type of message, etc.). In event 6, this message is coded into the SS7 TCAP query message and sent to the relevant SCP. After the query is sent, event 7 shows that the SSP suspends call processing on this particular call to await the response from the SCP node.

In event 8, Figure 2-9, the SCP decodes the incoming TCAP message. Based on the analysis of the fields in the message, it executes the specific software module to service the query (event 9). This module executes the operation and generates the parameters that are used to create the response message (event 10). Finally, in event 11 the SCP creates a TCAP response message and sends this message to the originating SSP.

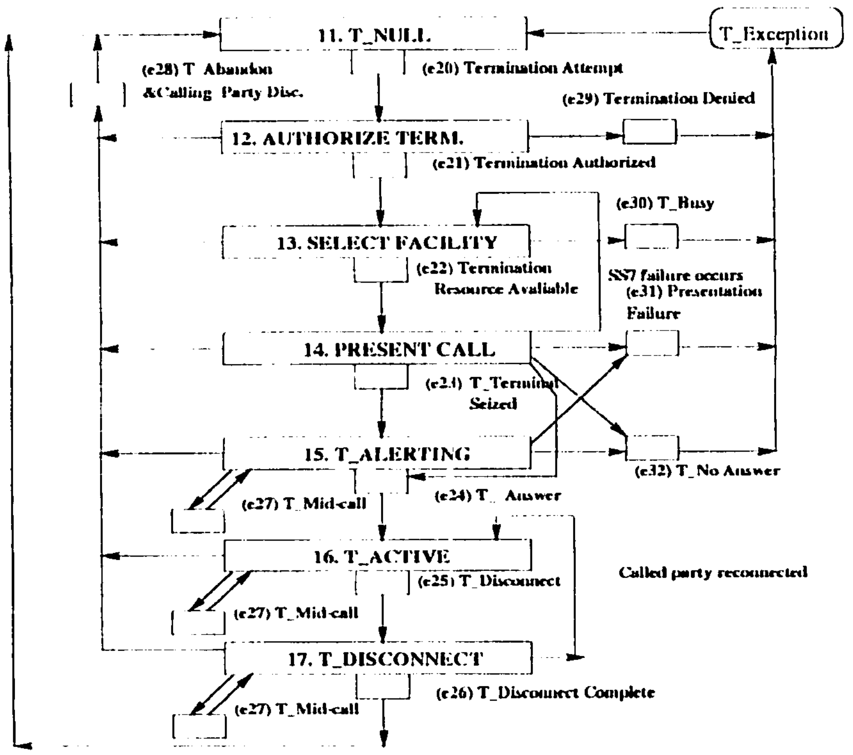
When the response arrives, the SSP reactivates the suspended call (see Figure 5-9), decodes the message as depicted in event 12, which may entail the execution of other actions (such as "do not charge for this call", "replace the dialed number with a different number" etc.). Then, the SSP begins the activation of the call processing at a specific PIC based on the information processed in event 12, which, in some instances, may assume a different point in the call than is shown in the sequential state diagram. Here, we call the activation of a different PIC as "warping". Whatever is the case, call processing is resumed as illustrated in event 13.

2.4.6 Bellcore AIN Call Model Releases

Figure 2-10 illustrates the originating and terminating basic call state machines of AIN 0.1. Please refer to Appendix A for detailed descriptions of PICs, TDPs and triggers that are defined in the originating and terminating basic call state machines of AIN 0.1.



Bellcore Originating Basic Call Model



Bellcore Terminating Basic Call Model

Figure 2-10: AIN 0.1 Basic Call Models.

Chapter 3

MSC and Feature Interaction Detection

Message Sequence Charts (MSC) [8] is the name given to a trace language that uses message interchange to specify and describe the communication behavior of system components and their environment. The language, particularly in its graphical format, makes an intuitive presentation of communication behavior. We employ it in conjunction with other languages to support methodologies for system specification, design, simulation, testing, and documentation.

The most important application of MSC is as an overview specification for the communication behavior of real-time systems, in particular, telecommunication switching systems. Using MSCs, we can readily specify selected system traces — primarily standard cases — and build non-standard cases covering exceptional behavior. This allows MSC's use in requirement specification, interface specification, simulation and validation, test case specification and documentation of real-time systems. Employed in connection with other specification languages — in particular Specification and Description Language (SDL) [12] — MSCs provide a basis for the design of SDL systems.

In the first three sections of this chapter, we review the definitions and formal semantic descriptions of the MSC concepts as outlined in the MSC Standard [8], and provide specific examples and diagrams to apply these to the subject of this thesis.

In the last section of the chapter (Related Work on Feature Interaction Detection), we present several related approaches to the detection of features interactions and identify some of the limitations of these approaches. This will prepare us for the development of our approach, presented in later chapters, which addresses these limitations.

3.1 The History of MSC Standardization

Industry and international standardization bodies such as ITU and ISO/IEC have long used MSCs. Although they followed different conventions and were labeled with a variety of names (Arrow Diagrams, Extended Sequence Charts, Information Flow Diagrams, Message Flow Diagrams, Time Sequence Diagrams). These MSC variants differed largely with respect to minor semantic differences and terminology. Standardization was clearly viable.

Though MSCs may play an important role in the years to come, the SDL user guidelines of 1988 [13] devoted only a short section to MSCs. Grabowski and Rudolph pointed out the language's greater applicability in their 1989 paper *Putting Extended Sequence Charts to Practice* [14], presented at the SDL Forum in Lisbon. They termed MSCs, enhanced by SDL symbols and a few further constructs, "Extended Sequence Charts" (ESCs). They presented ESCs as tools that would refine and extend MSCs, allowing the ultimate derivation of SDL specifications, and highlighted the role of MSCs and ESCs within the whole software life cycle, from requirement specification to test case specification.

MSCs aroused great interest at the1989 SDL Forum, leading to the suggestion that they be standardized in graphical and textual representation within the ITU-T. The ITU-T approved standardization at the meeting in Helsinki in June, 1990 and decided to concentrate first on the basic language constructs of MSCs, that is, message flow diagrams without further extensions such as SDL symbols, and in particular to work out a clear semantics for these constructs. Through this restriction, the ITU-T hopes to avoid undue overlap with SDL.

At the same meeting, Tilanus presented a first attempt at the formalization of MSCs (updated in [15]). He focussed this formalization on equivalence relations for MSCs and on the merging of instances within MSCs to provide a formal relationship between different levels of abstraction, pointing out that, by merging of instances, we obtain a more general time ordering for events than was originally defined for the basic language of MSCs. These early investigations have influenced the inclusion of higher level concepts contained in the final MSC recommendation [16].

The Helsinki conference decided not to prepare a separate recommendation for standardizing MSCs, which were to be covered as part of the new *SDL Methodology Guidelines* aimed at the effective use of SDL. But it was soon recognized that the standardization of MSCs would go beyond the SDL guidelines. In addition to SDL [12], LOTOS [17], and ESTELLE [18], a comprehensive classification called for a fourth Standard Descriptive Technique, one in which MSC was described as a new specification language that might be used in combination with other languages for system development. This designation was subsequently formalized at the next ITU-T meeting, in Geneva in February, 1991.

The Geneva meeting also agreed to include further language constructs such as conditions (representing system states), timer constructs and some higher level structural concepts (e.g. Macros) that would go beyond pure MSCs. Researchers further elaborated on these concepts prior to the ITU-T meeting December 1991 in Recife, adjusting the language constructs to cover the needs of other ITU-T recommendations employing Message-, Signal-, and Information-Flow Diagrams.

At the ITU-T meeting in Recife, a thorough and critical review of the draft MSC recommendation by Swedish Telecom formed the core agenda. A first selection of higher level constructs took place, retaining coregion, substructure, macro, but postponing the MSC language constructs for remote procedure calls and grouping of instances to the next study period. In addition, the conference decided to include a "create and stop" for MSC instances. The Recife meeting also modified the form of the draft MSC recommendation to bring it into line with the SDL recommendation [12].

The session of ITU-T Study Group X, Geneva, May 1992, approved a new MSC recommendation with a few changes. In particular, it renamed "substructure" as "submsc" and found the macro concept to be insufficiently mature and hence postponed adopting it to the next ITU-T study period.

From the earliest attempts at MSC standardization, developers had considered combining MSCs with composition mechanisms from process algebra. This work was forwarded by development of the ObjectGeode tool. But it was only with the development of the formal MSC semantics that developers were able to realize the potential of this approach and the evolution of an MSC language gained impetus during the ITU study period from 1993 to 1996. The condition-based composition mechanisms of MSC'92 were not dropped but incorporated into process algebra based techniques in MSC'96 [8].

MSC'96 became a powerful synthesis of concepts taken from process algebra, petri nets and, beyond that, from object oriented modeling, where MSC may become a standard for the formulation of Use Cases, as is under discussion within the *Unified Method for Object Oriented Development* [19]. MSC'96 new language concepts — generalized ordering, inline expression, reference, High Level MSC (HMSC) — has increased its applicability. The specification of Use Case [20], i.e., of main scenarios together with all accompanying side cases, is one of the most promising candidates for the application of MSC'96 [21] and may overcome the restriction of MSC to the specification of only a few selected scenarios, which was considered as the major shortcoming of MSC'96.

3.2 MSC Language Constructs

In this section, we describe the language constructs of MSC'96. The language of MSC includes all constructs necessary to specify pure message flow. These language constructs are Instance, Message, Environment, Condition, Timer, Action, Instance Creation and Stop, Coregion, Instance Decomposition (Submsc), Inline Expression and High-Level MSC.

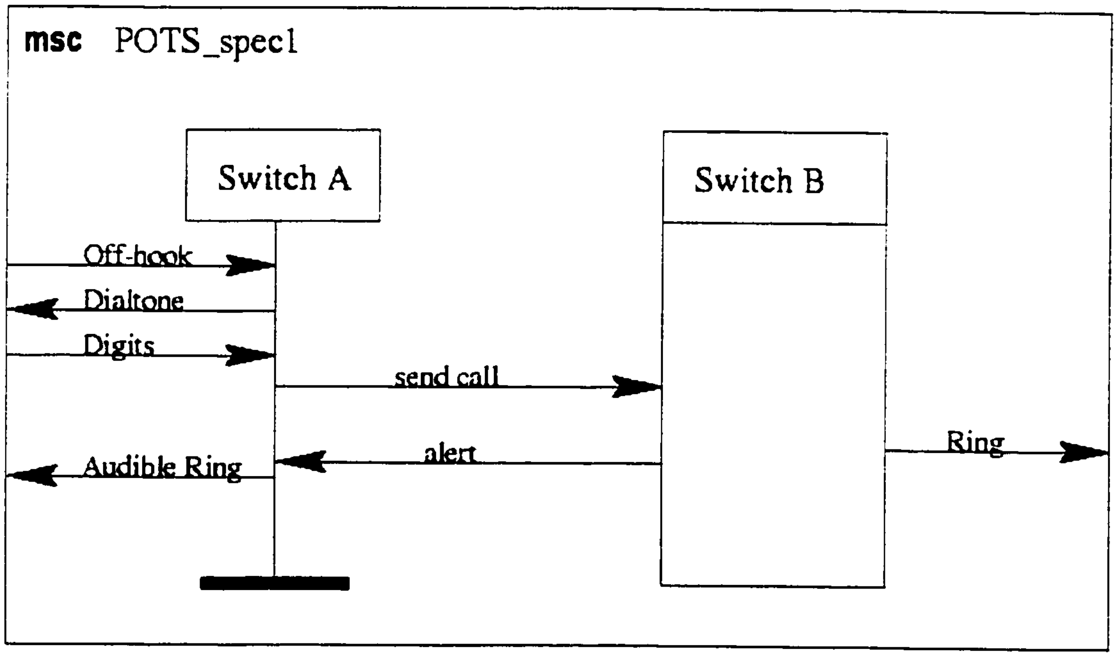


Figure 3-1: A POTS Specification Using Basic MSC Constructs.

3.2.1 Instance and Message

The most basic language constructs are instances. We use instances to specify communicating entities. The instance of an entity is an object which has the properties of this entity. In the context of SDL, an entity may be an SDL process, block or service. Within the instance heading, the entity name may be specified in addition to the instance name. Within the instance body, the ordering of events is specified. In graphical representation, instances are represented by vertical lines or alternatively by columns. Figure 3-1 presents an MSC specification of a POTS call. The switches of the telephone network are represented by MSC instances.

According to Z.120, a message within an MSC specifies "a relation between an output and an input. The output may come from either the environment (through a gate) or an instance, and an input is to either the environment (a gate) or an instance ." A message is presented by arrows (Figure 3-1) in graphical format. The head of the message arrow denotes the message consumption, the opposite end of the arrow denotes the message sending. In addition to the message name, message parameters in parentheses may be assigned to a message.

Along each instance axis (column), we assume a total ordering of the described communication events. Events of different instances are ordered only via messages, since a message must be sent before it is consumed.

3.2.2 Environment

Within an MSC, we use the frame symbol to represent the system environment. This is indicated by a rectangle which forms the boundary of the MSC diagram (Figure 3-1). Unlike instances, we assume no ordering of communication events on the environment. In Figure 3-1, all user interactions with the telephone system are assumed to be coming to/from the environment, as shown by the messages drawn to/from the boundary of the MSC.

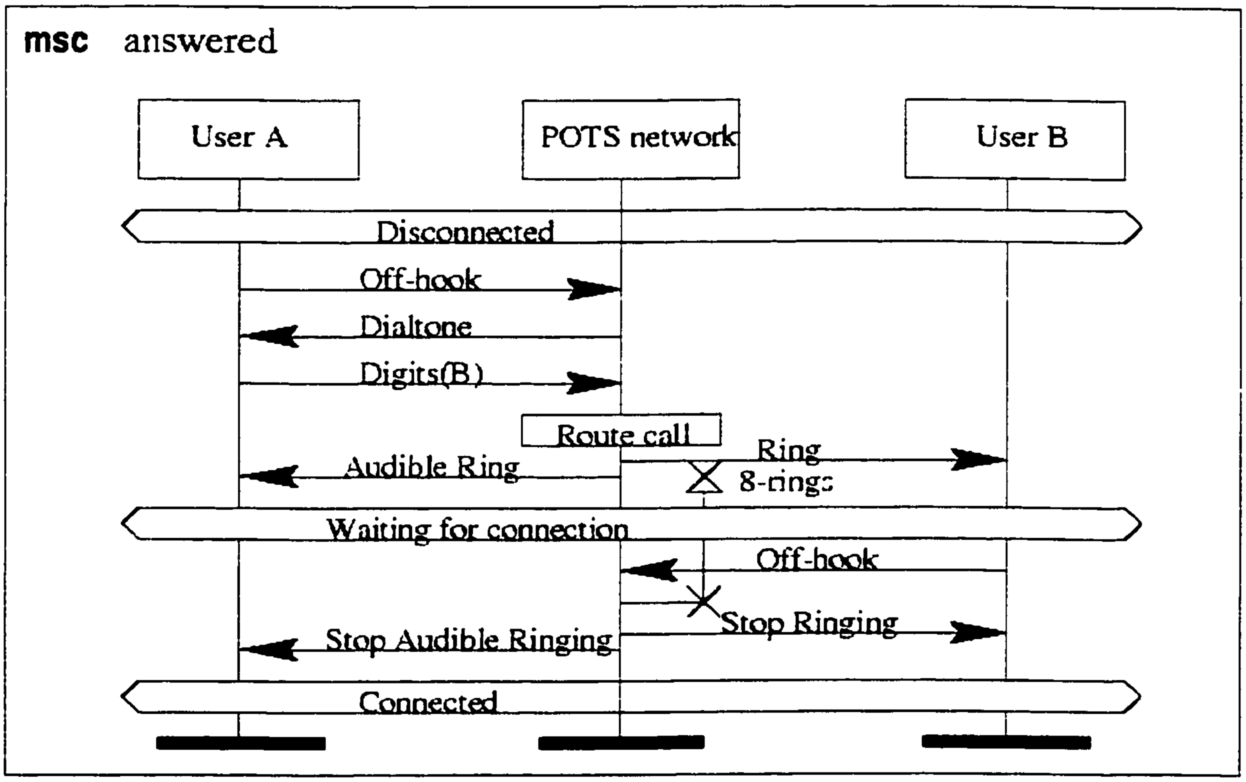


Figure 3-2: POTS Specification of the "answered" Scenario.

3.2.3 Condition

Z.120 describes condition as "either a global system state (global condition) referring to all instances contained in the MSC or a state referring to a subset of instances (non-global condition). In the second case the condition may be local, i.e. attached to just one instance." We can use conditions to emphasize important states within an MSC or for the composition and decomposition of MSCs. In the graphical format, conditions are represented by hexagons covering the involved instances (Figure 3-2).

In Figure 3-2, we present another specification of the same POTS call. In this specification, we separate the users from the environment and consider them to be actors in the call. At the same time, we consider the network one actor regardless of how the network is comprised. The instance "POTS network" is not covered by the conditions "Disconnected" and "Connected". It is therefore not involved in the states to which the conditions refer. The condition "Waiting for connection" covers all three instances; therefore, it represents a global system state.

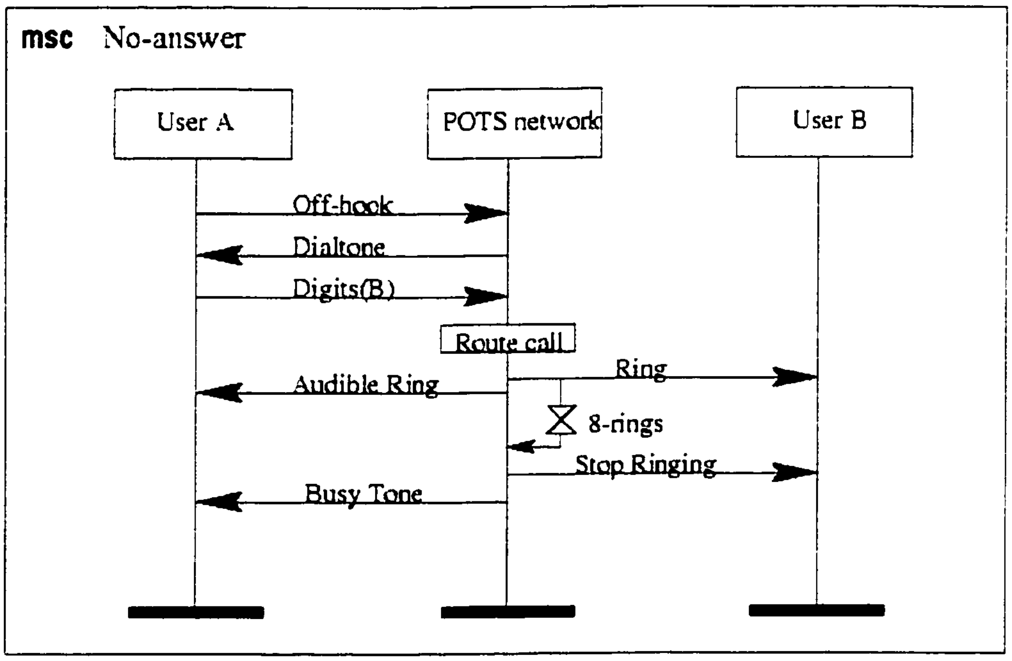


Figure 3-3: POTS Specification of the "no-answer" Scenario.

3.2.4 Timer

In MSCs, we may specify either the setting of a timer and a subsequent time-out due to timer expiration, or the setting of a timer and a subsequent timer reset (time supervision). In addition, the individual timer constructs — timer setting, reset/time-out — may be used separately, for example, where timer expiration or time supervision is split between different MSCs. In the graphical representation, the set symbol has the form of an hour glass connected with the instance axis by a (bent) line symbol. Time-out is described by a message arrow, pointing at the instance, attached to the hour glass symbol (Figure 3-3). The reset symbol has the form of a cross symbol which is connected with the instance by a (bent) line (Figure 3-2). In Figure 3-3, another scenario for the POTS call was specified: if the call is not answered within a certain period of time (represented by the expiration symbol of the 8-rings timer), the network will give up presenting the call.

3.2.5 Action

Z.120 describes action as "an internal activity of an instance". In graphical format, an action is represented by a rectangle containing an arbitrary text. In Figure 3-2 and 3-3, the "route call" action represents an internal action performed by the network.

3.2.6 Instance Creation and Stop

Most communication systems are dynamic, with instances appearing and disappearing during the system's lifetime and making the creation and termination of instances within communication systems common events. A system designer obviously needs features to describe such events.

The corresponding MSC language elements are shown in Figure 3-4. The "create" symbol is a dashed arrow which may be associated with textual parameters. A create arrow originates from a "father" instance and points at the instance head of the "child" instance. The termination of an instance is represented graphically by a cross (stop symbol) at the end of the instance axis.

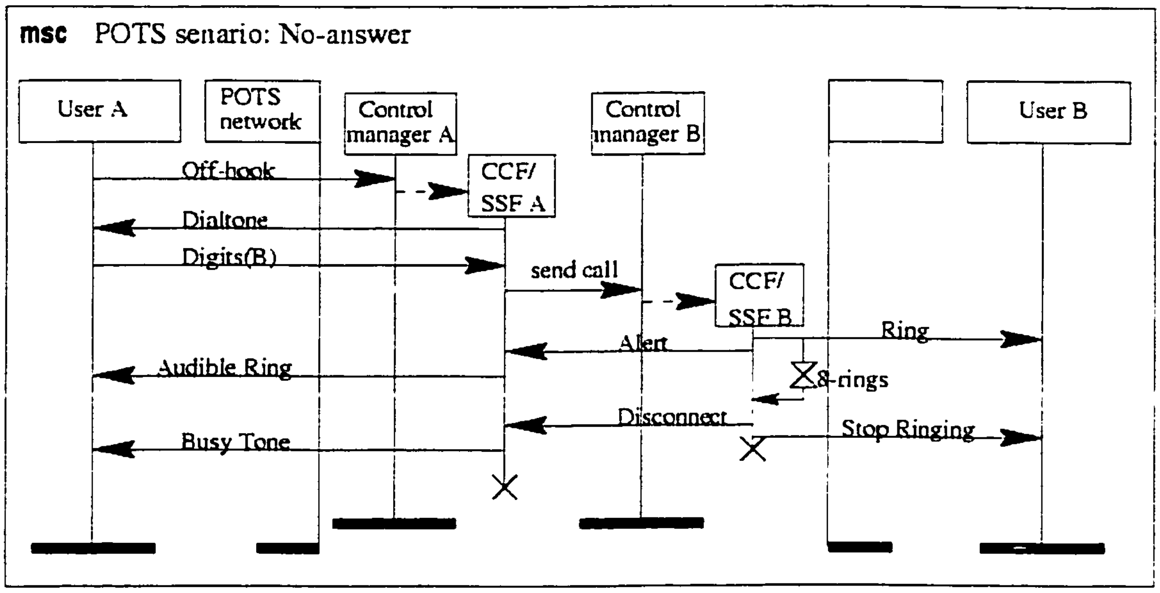


Figure 3-4: Instance Creation, Stop and Submsc.

Figure 3-4 shows some internal details of the call set-up and release process for the call scenario in Figure 3-3. When user A goes off-hook. the local switching element spawns a child CCF/SSF process which will be responsible for setting up the call. providing call supervision and termination of the call. When the call reaches the local switch element of the called party, another instance of the CCF/SSF is created which serves like an agent for the called party. When the call is terminated, both processes terminate and return the resources to the system.

3.2.7 Instance Decomposition (submsc)

A submsc is an MSC that refines another MSC instance. The submsc represents a decomposition of this instance but does not affect its observable behavior. The messages addressed to and coming from the exterior of the submsc are characterized by the messages connected with the submsc border (frame symbol). Their connection with the external instances is provided by the messages sent and consumed by the corresponding decomposed instance, using message-name identification. It must be possible to map the external behavior of the submsc to the messages of the decomposed instance and the ordering of message events specified along the decomposed instance must be preserved in the submsc. In Figure 3-4, the instances "Control manager A", "Control manager B", "CCF/SSF A" and "CCF/SSF B" constitute the submsc of the instance "POTS network".

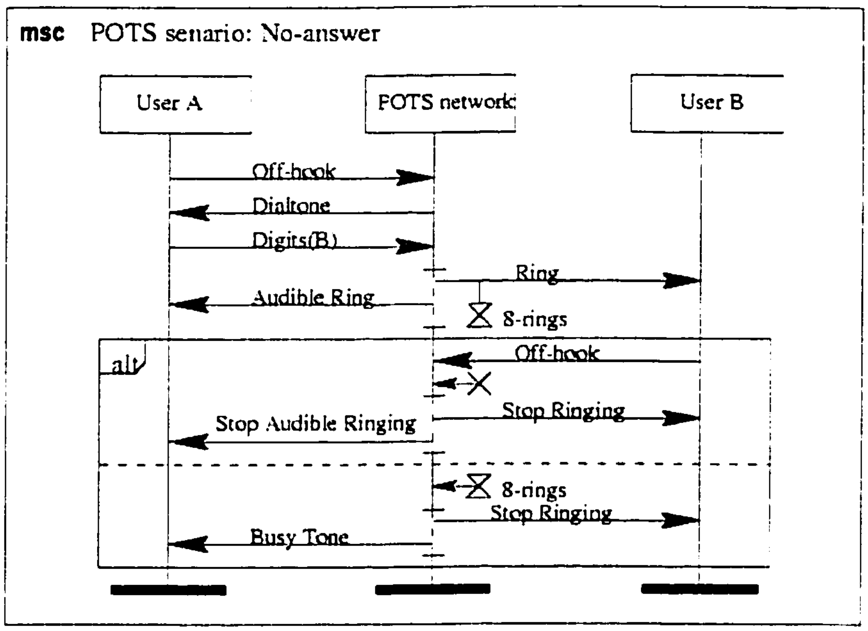


Figure 3-5: Coregions and Inline Expressions.

3.2.8 Coregion

Along an MSC instance, message events are totally ordered. This may not be appropriate for instances referring to a higher level than SDL processes. For this reason, Z.120 introduces a so-called coregion denoting a section of an MSC instance where the specified communication events are not ordered. Within any coregion, only sending (origin of message arrows) or only consumption events (arrow heads) may be specified. In Figure 3-5, the dashed line segments containing the messages "Ring" and "Audible Ring", "Stop Ringing" and "Stop Audible Ringing", "Stop Ringing" and "Busy Tone" are coregions. For example, the coregion containing "Ring" and "Audible Ring" implies that the ordering of these two messages is not important.

3.2.9 Inline Expression

The composition of event structures may be defined inside of an MSC through the use of inline operator expressions. The operators refer to alternative, parallel composition, iteration, exception and optional regions. The inline expression is represented graphically by a rectangle with dashed horizontal lines as separators. There are currently 5 operator keywords defined for inline expression — alt, par, loop, opt, exc — that denote alternative composition, parallel composition, iteration, optional region and exception, respectively. The operator keyword is placed in the left upper corner. Figure 3-5 shows an inline expression with the keyword alt. This means that only one of the two msc sections separated by the dashed line will be executed.

3.2.10 HMSC

High-level MSCs provide a means to graphically define how a set of basic MSCs can be combined. An intuitive, non-standard name for HMSCs might be "road maps", which suggests how HMSCs are used in practice.

The ITU-T standard Z.120 defines a set of HMSC constructs. However, different msc tool sets may have their own notations for HMSC. Figure 3-6 shows HMSC language constructs as defined in the ObjectGeode tool set [22]:

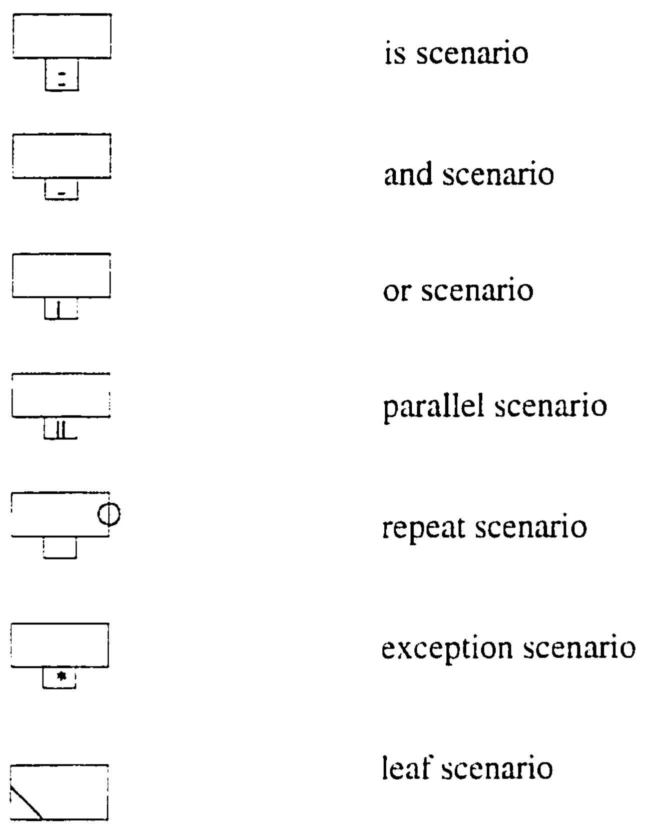


Figure 3-6: HMSC Constructs Defined in ObjectGeode.

Figure 3-7 shows an HMSC specified using ObjectGeode. This HMSC contains an "or" scenario with two "leaf" scenarios. The answered leaf scenario corresponds to the msc call scenario specification presented in Figure 3-2. The no-answer leaf scenario corresponds to the msc call scenario specification in Figure 3-3. Together, the HMSC describes a typical POTS call. And the behavior specified is equivalent to that of Figure 3-5.

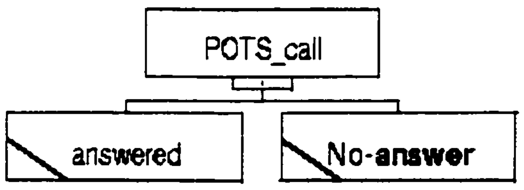


Figure 3-7: An HMSC Containing an OR Scenario.

3.3 Using MSC to Specify Telecommunication Features

To specify a telecommunication feature in MSC, any specification must identify two basic elements: the parties involved in the communication and the messages exchanged between the parties during the communication process.

There are different abstraction levels within the system; therefore, different viewpoints may result in quite different specifications. For example, in the case of a typical telephone call, if we consider all telephone and network equipment as belonging to a single entity and all entities outside of this system as the environment, then the resulting specification will contain only one instance (as shown in Figure 3-8). All communication events exchanged between the system and its environment are modeled as messages. Figure 3-8 shows the specification at the highest level of abstraction. If we separate the users from the environment, then our specification would look like the specification shown in Figure 3-2. In this case, we can distinguish between the behavior of different users and the behavior of the network as a whole. However, we are not aware of any communication events that happen inside the telephone network. The network structure is completely invisible to the viewer and we cannot tell whether the network is an intelligent network or just a plain old telephone network, even though the feature under specification may be an IN feature. Figure 3-4 shows some internal details of the network, that is, the SSF/CCF spawned to handle user requests. However, the means by which different network elements coordinate to fulfill the user's requirements are still hidden to the viewer. We still cannot distinguish between an IN feature and a non-IN feature. In order to specify an IN feature, we need to have more details about the network structure.

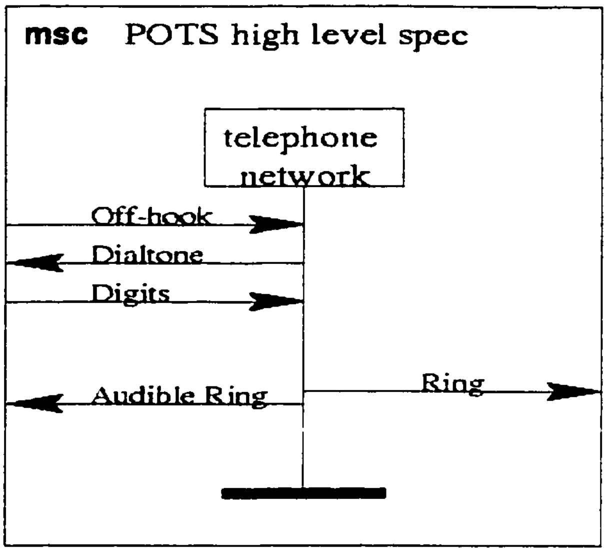


Figure 3-8: A High-Level Specification of a Telephone Call.

Weng[23] established a good framework for specifying IN features. In his specification, the network is broken down into physical entities that participate in an IN feature. Standard names like SSF/CCF, SCF, IP etc. are used for instance naming. The messages exchanged use names in TCAP or other protocols, thus complying with IN standards. In the section that follows, we provide an example of the specification procedure and explain the concepts used in the specification.

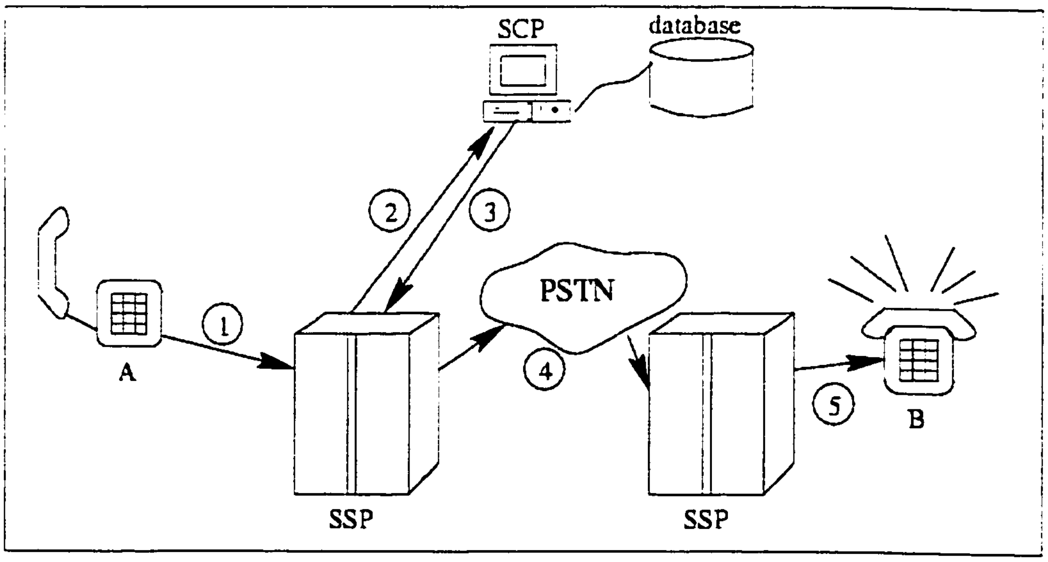


Figure 3-9: The IN Automatic Call Back Feature.

Figure 3-9 shows an IN Automatic Call Back (ACB) feature. The corresponding MSC specification is illustrated in Figure 3-10. ACB is a feature that allows a user to redial the last call that the user made. As shown in Figure 3-9, the following steps are taken to make an ACB call:

1. The calling party dials ACB access code
2. The SSP processes the call and realizes that the call needs IN processing. It builds a query message in TCAP format, Query with Permission (InfoAnalysed) and sends the query to SCP (shown in Figure 3-10 by the Qwp message). It includes the following parameters in the message:
   - Trigger Criteria: SFC
   - Calling Party ID: A
   - Collected Digits: XX
   - Carrier: 0110(local carrier)
3. Upon receipt of query message, SCP analyzes the message, identifies that it needs ACB service. It does necessary database querying to find the last number that the customer called (which happens to be B) generate billing information and sends back a response message Response(Analyze Route) to SSP. In the message, the following parameters are included:
   - Last Called DN
   - AMA Slip ID #
4. After receiving this message, the SSP routes the call to B. in Figure 3-10, the AR message shows this response message.
5. B receives the ACB call.

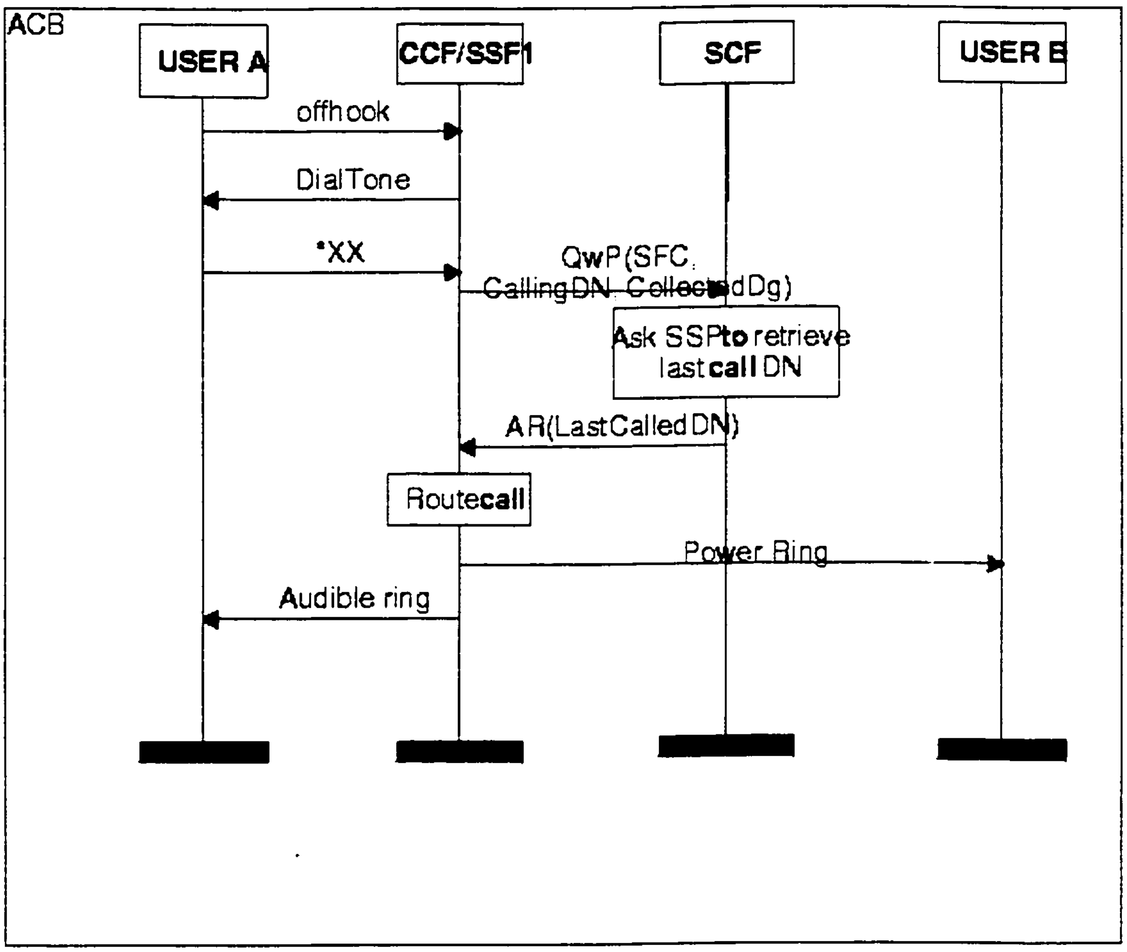


Figure 3-10: MSC Specification of the ACB Feature.

Although the method presented above effectively specifies IN features, it is inadequate for the detection of feature interactions. For this purpose, we will require more details on certain important properties of the feature that are not shown in this specification style. In Chapter 4: the Feature Specification Style, we discuss how we add those details to this specification, creating a unique specification style.

3.4 Related Work on Feature Interaction Detection

There are currently many approaches in the detection of feature interactions. According to their methods of application, these approaches can be classified as either off-line detection approaches or on-line detection approaches. According to their modeling techniques used, they can be classified as either formal approaches or pragmatic approaches. Our approach to be presented in later chapters, in particular, belongs to the category of off-line formal detection approaches.

In this section of the thesis, we do not intend to provide an overview of all existing approaches. For that purpose, please refer to Keck and Kuehn [38] for a comprehensive survey of many of the existing approaches. Instead, we will concentrate on several approaches that use similar modeling techniques to our approach.

In Bergstra and Bouma [24], an MSC-like language, "interworking" is used to describe features and detect feature interactions. Interworkings are like MSCs with the difference that interworkings use synchronous communications. In graphical format, interworkings are called interworking diagrams, which are similar to the graphical format of MSCs. In the case of interworkings, they define several composition operators, specifically sequencing and interworking merge. These are comparable to vertical (sequential) and horizontal (parallel) composition as defined in MSCs. They describe the merging of two interworkings as the synchronization of both diagrams in respect to all actions they have in common. The merging of two interworkings is called "merge-inconsistent" if they cannot synchronize in respect to all communication actions they have in common. In their paper *Models for Feature Description and Interaction* [24], Bergstra and Bouma state that there is a feature interaction between two features if and only if their interworkings are merge-inconsistent.

Bellcore's paper [25] describes a feature as a sequence of traces. The authors model the composition of two features as the union of their traces. They define a projection operator, which gives a focused view of the combined behavior as seen from one of the features. They then define feature interactions according to the trace equivalence criteria, as follows: if, after projecting the combined behavior onto events recognized by one feature, the resulting set of traces is different from the original set of traces of this feature, then there is a feature interaction.

There are drawbacks to both the Bellcore and Bergstra and Bouma's approaches, since both try to model feature interactions as conflicts between traces and fail to consider other properties of the feature, for example, triggers and activation periods. In the following, we discuss three of the drawbacks:

1. If two features can actually be executed one after another without any interaction, both approaches may mistake this situation as a feature interaction. For example, consider the situation between Voice Activated Dial (VAD) feature and the Follow Me Diversion (FMD) feature. VAD is a feature that allows a caller to use his/her voice to dial a number that he/she wants to reach. The FMD feature, on the other hand, forwards all incoming calls to a directory number where the customer of FMD can be reached. As shown in the MSC diagram in Figure 3-11, the trigger "Offhook-Immediate" of VAD is launched before routing and routing, when it takes place, fulfils VAD's goal. The trigger "Termination-Attempt" of FMD is launched only after routing, as illustrated in the MSC diagram in Figure 3-12. The activation periods of the two features clearly do not overlap and, as we can see in the MSC diagram in Figure 3-13, we can combine these two features without any interference. Yet by looking only at the traces, both the Bellcore and the Bergsta and Bouma's approaches will erroneously conclude that there is a feature interaction.

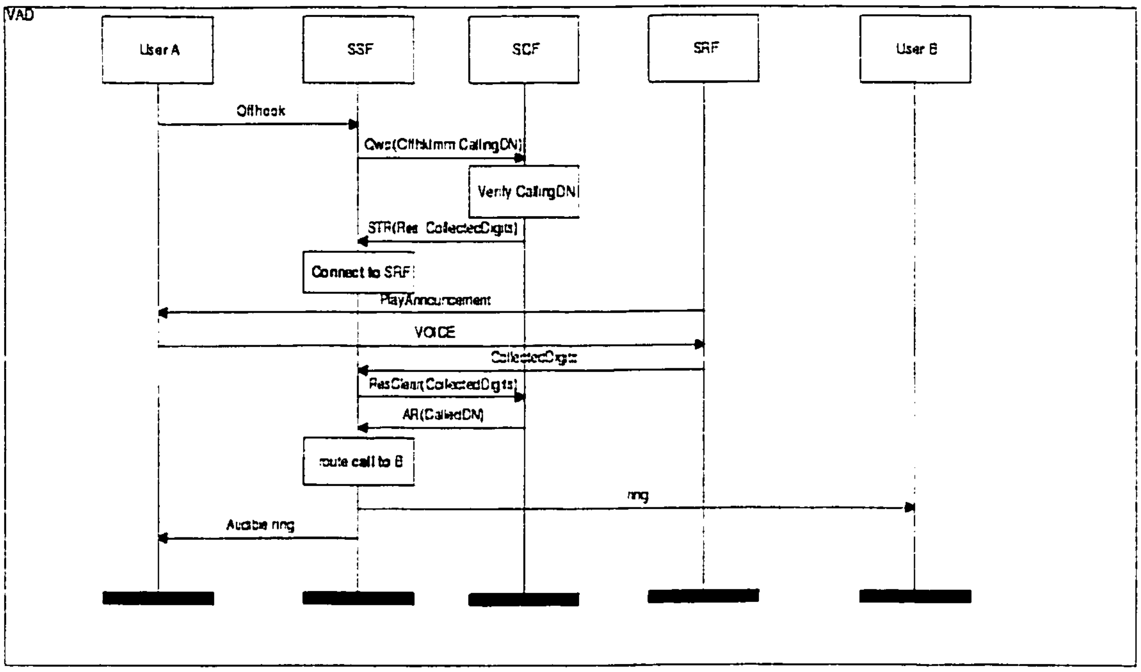


Figure 3-11: Voice Activated Dial.

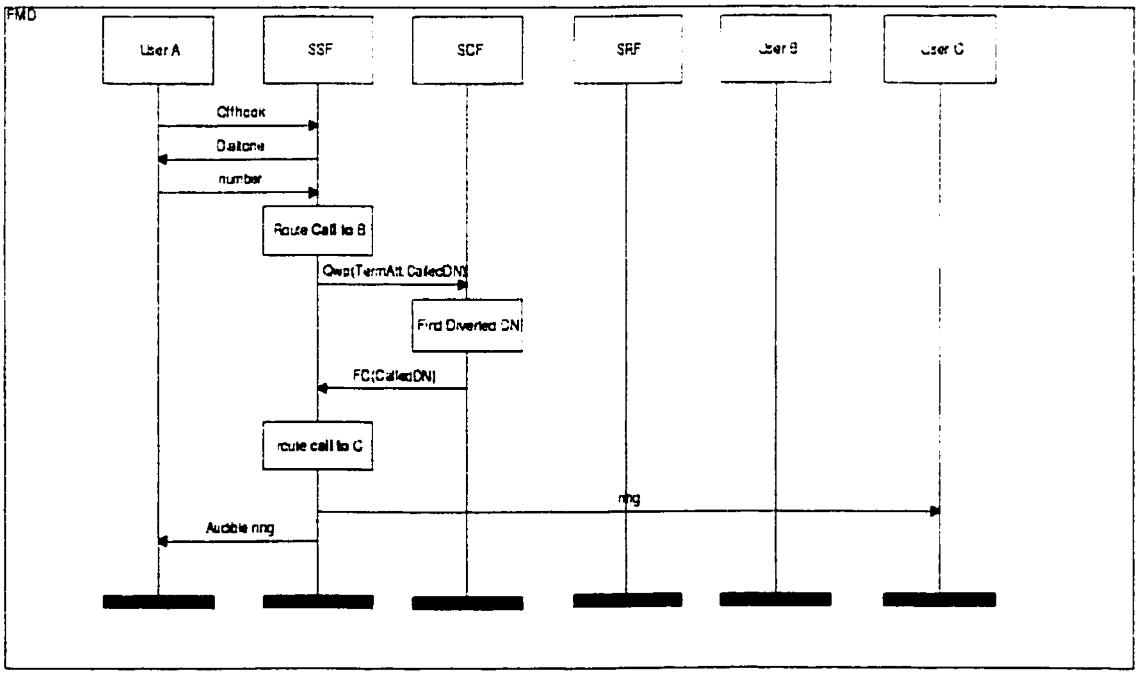


Figure 3-12: Follow Me Diversion.

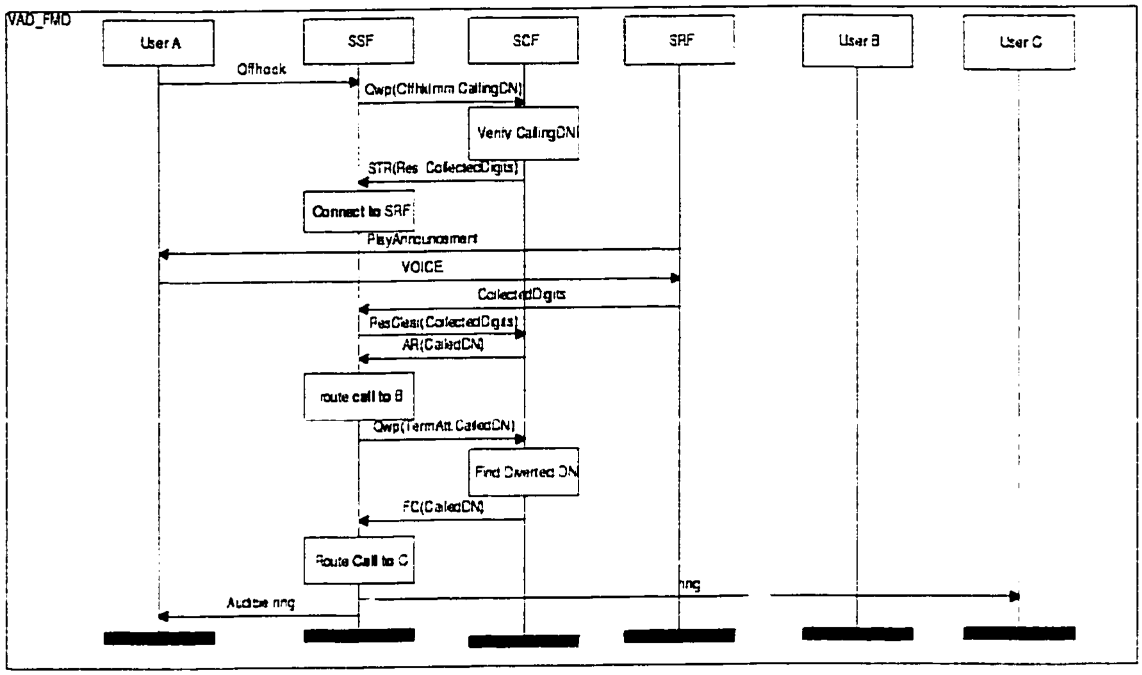


Figure 3-13: VAD and FMD Combined with No Feature Interaction.

2. If the triggers of two features are launched in different call contexts, and these contexts cannot coexist, then in any situation, only one of the features can be launched. There is no way that these two features can coexist, so there is no feature interaction. However, the Bellcore and Bergsta and Bouma's approaches may again mistakenly identify this situation as feature interaction. Let us consider an example: Call Forwarding on No Answer (CFNoA) is a feature that forwards the incoming call to another number if the call is not answered in a certain period of time. Call Forwarding on Busy (CFB) is a feature that forwards the incoming call when the customer is busy. The trigger criteria (or context) for CFNoA is that the line is not busy and the call is presented but not answered after a certain period of time. The trigger criteria (or context) for CFB feature is that the line is busy. Obviously, these are two call contexts which cannot coexist and CFNoA and CFB cannot coexist in a single two-party call. They cannot possibly interact in this situation. However, by only looking at the traces, one may erroneously make the wrong surmise that there is a feature interaction.

3. As viewed from a higher level, two features may have exactly the same traces, but different triggers cause them to interact. In this situation, if we simply examine the traces of the two features, we cannot identify the interaction. For example, as shown in Figure 3-14 (from [24]), in the context that the called party is busy, Call Forwarding on Busy (CFB) and Call Forwarding Unconditional (CFU, it is actually another name for FMD) have exactly the same traces, yet their triggers may interact. In fact, when enabled on the same subscriber, CFB cannot be initiated when there is CFU because the trigger Termination-Attempt for CFU is always reached first.

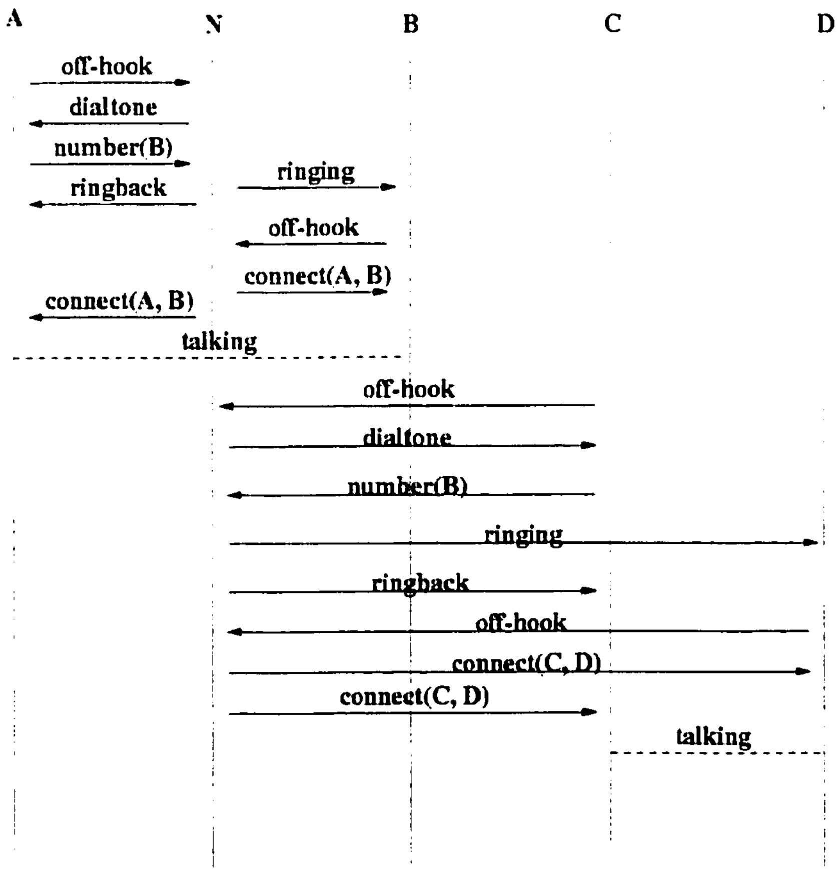


Figure 3-14: The Interworkings for Both CFB and CFU [24].

From the above illustrations, we can conclude that it is not enough to detect interactions by simply looking at the traces. Other properties of the feature, such as positions of triggers, the activation periods of two features, and the trigger criteria, should also be considered.

In later chapters of this thesis, we will show how our approach has addressed the limitations of the Bellcore and Bergstra and Bouma's approaches and present our contribution to feature interaction detection.

Chapter 4

The Feature Specification Style

As we have seen, a given specification language may offer a variety of ways to describe the same feature. However, in order to implement detection techniques using such a language, we must work out a specification style — the set of rules for feature specifications — that can be easily recognized by the detection tool and by others using the language.

In this thesis, we use MSC as the language of specification because MSC is a powerful and standardized language. However, our underlying detecting techniques are independent from the specification language, and — provided we develop a specification style for another language — it is possible to apply our techniques to that language.

Our MSC specification style consists of two parts: the structures for feature specifications, and the naming conventions for these structures. With the help of the structures and their naming conventions, the tool we develop can extract useful information about the feature. We can ensure that the information contained in any specification is unique provided that the specification follows our specification style.

This chapter describes our specification style in MSC. We begin with the introduction of several important properties that are related to feature interaction detection, then introduce the structural elements of the specification style and their corresponding MSC constructs. Following this, we present our naming conventions for all these structural elements. To help readers quickly master the specification style, we provide a feature specification procedure. A summary of the strengths and weaknesses of this specification style is presented at the end of this chapter.

4.1 Concepts Related to Feature Interaction Detection

As described in Chapter 3, there are several related approaches to feature interaction detection. All these approaches have a common drawback: they ignore several important properties of features. Our work suggests that, to detect a feature interaction, these key properties — triggers, traces, and static requirements — should be taken into consideration. To avoid ambiguity, we clarify our notations as follows:

Initial Trigger: This is the first trigger on the BCSM that activates a feature. For example, the initial trigger for Freephone is the trigger SDS-11 (special digits string, 11 digits), while the initial trigger for CFU is TermAtt (Termination Attempt). In the VAD feature presented in Chapter 3, the initial trigger is Offhook-Immediate.

As we have seen in Chapter 2, each trigger resides in a specific TDP and has a corresponding trigger criteria that specifies the situations in which this trigger will be launched. The Offhook-Immediate trigger for VAD, for instance, resides in the TDP: Origination Attempt. The trigger criteria is that the user picks up the phone.

GOAL: We have coined this term to specify the uppermost (earliest) point in the BCSM at which the objective of a feature can be considered as fulfilled. For example, the GOAL for VAD is reached when the called party directory number is collected successfully. Note that this concept of "GOAL" is not the same as the IN concept we term "Point of Return" (POR). GOAL is only a logical concept representing the point where the objective of a feature can be considered as fulfilled, whereas POR refers to the specific moment that the normal call processing is resumed in the BCSM and call control is returned to the switching function. GOAL and POR, however, may sometimes refer to the same point in the call.

Activation Period: This is the period beginning at the Initial Trigger of the feature and ending at the GOAL of the feature. Again, for VAD, this period begins at the launching of the Offhook-Immediate trigger and ends after the digits are successfully collected.

Trace: A trace of a feature is defined as the sequence of all transitions taken by a specific BCSM during the lifetime of a call with this feature enabled. A feature can have several traces, each corresponding to a BCSM that participates in the activities of the feature.

Peer Traces: Peer traces may occur in two features. The term refers to traces that have the same corresponding BCSM. For each connection, regardless of how many features are enabled, each SSP spawns only one instance of BCSM, in other words, one O_BCSM in the originating SSP and one T_BCSM in the terminating SSP. This results in features sharing the common pair of BCSMs (and peer traces) if they share the same connection. A direct consequence of this sharing is feature interactions between peer traces. Some features, e.g. Three Way Calling, may maintain several connections at the same time. In this case, we should distinguish the peer traces in each connection with an additional connection number.

Static Requirement: A Static requirement of a feature is a restriction on the occurrence of certain events or combination of events or on the reaching of a certain system state. As an example, a common static requirement on almost all features is that callers are not allowed to connect to themselves. This restricts both the connection signal and the state in which callers are connected to themselves.

4.2 Feature Concepts and Their Corresponding MSC Structures

The structures for feature specifications are selected based on the AIN Basic Call Model. To exploit the rich structural concepts defined in the MSC language, we mapped the structures of the AIN Basic Call Model to structures present in MSC. Table 4-1 presents a summary of the mapping:

AIN Concept		*MSC Structure*
Behavior of a feature	$\Rightarrow$	MSC specifications
One call scenario	$\Rightarrow$	One MSC specification
Network elements and users	$\Rightarrow$	Top level instances
Events happening in between network elements and the users	$\Rightarrow$	Top level messages
Basic call state machines	$\Rightarrow$	Sub-MSC of the SSF (switch)
One basic call state machine	$\Rightarrow$	One instance in the sub-MSC
Internal events in the switch	$\Rightarrow$	Internal messages in the sub-MSC
PICs	$\Rightarrow$	States on the BCSM instance
Trigger	$\Rightarrow$	Action on the BCSM instance

Table 4-1: Mappings between AIN Concepts and MSC Structures.

In Table 4-1 "behavior of a feature" refers to the overall behavioral characteristics of the feature. This behavior can be represented using several characterizing call scenarios. Each call scenario represents a procedure which begins at call setup at the switch and ends at completion of the call.

4.3 Naming Convention

Although the structures for feature specifications present one-to-one mappings between feature concepts and MSC structures, the namings of these structural elements create another possibility of ambiguities. To uniquely identify a specification, we need to have some rules for the namings of structural elements and, as we have said, the naming conventions are simply sets of rules that help to remove these ambiguities. The naming conventions are created to help the feature specifiers communicate with each other and with computer programs. Similar to creating a new programming language on top of the specification language (in our case, MSC), the naming conventions define a set of reserved keywords and syntax for many structures. With the help of the EBNF formalism, we will illustrate the structures and their naming conventions one by one.

4.3.1 Network Elements

In an MSC, network elements currently involved in the Call can be represented as instances. There are generally two types of elements: "users" and "elements internal to the network". User names are represented as a single letter. For elements internal to the network, we adopt the functional entities naming conventions of the IN distributed functional plane and disregard the CCAF, SMF, SMAF, and SCEF. That is, the following functional entities are present: SSF/CCF, SCF, SRF, SDF. In EBNF format, we define the allowable top-level instance names as follows:

Top_Inst_Name ::= <letter> | 'SSF' | 'CCF' | 'SCF' | 'SDF' | 'SRF'

In EBNF, "|" represents the logic "OR" operator, "::=" represents the definition operator, "{ }" specifies that the elements inside it can be repeated for zero or more times, "[]" specifies an optional element that can only be repeated zero or one time, "<>" specifies a fixed element that is present once and only once.

In the above definition, <letter> is a single alpha character representing a user. For the same functional entities which reside in different nodes of the network, we will assume that they form a single functional entity. In other words, we will not consider physical distribution of functional entities.

In order to represent the BCSMs that are vital to our detection technique, we apply the submsc construct on the SSF. That is, we refine the SSF instance into a submsc consisting of O_BCSM and T_BCSM instances. The names of the O_ or T_BCSM instances also include an additional parameter specifying which party it represents. Optionally, a BCSM instance may also carry a connection number if it is necessary to distinguish between BCSMs from different connections maintained by the same feature. Therefore, we define an instance name in the submsc and names for all instances as follows.

Submsc_Inst_Name ::= 'O_BCSM' | 'T_BCSM' <letter> ['('<number>')']
Instance_Name ::= <Top_Inst_Name> | <Submsc_Inst_Name>

In Figure 4-1, we can see how we represent typical elements of the network in MSC.

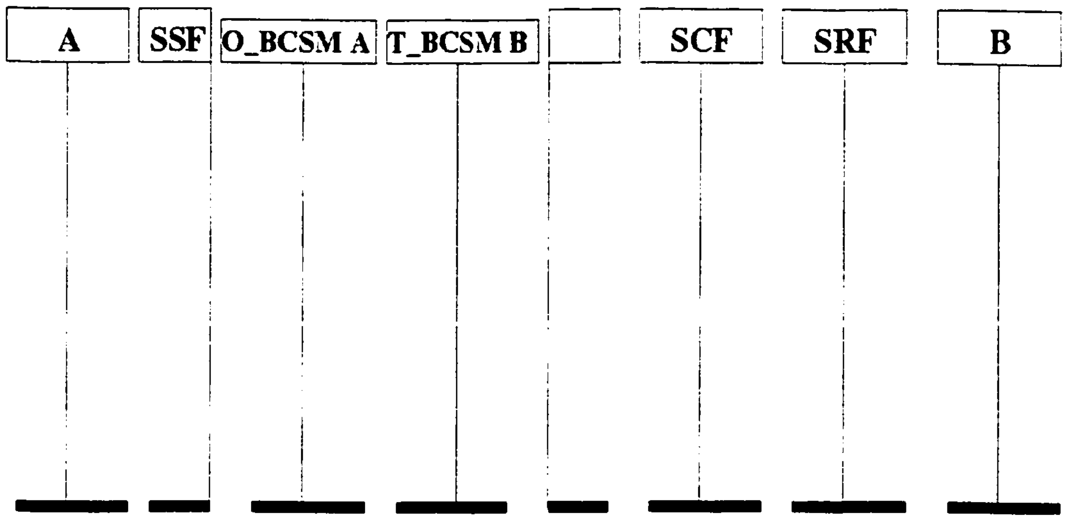


Figure 4-1: Network Elements Represented as Instances in MSC.

4.3.2 Format of Events

We can represent events that occur between the network elements as messages in MSC. Sometimes it becomes necessary to uniquely identify an event sent and received between network elements and this entails restrictions on the format of events.

Bellcore's Generic Requirements for Switching Systems [26] has defined the format of events as shown in Table 4-2. However, this format is redundant when applied to events specified using MSC messages, since every message in MSC may already have a parameter node attached to it, either as a receiver or a sender. For example, since an Off-hook message drawn from user A to the SSF effectively identifies the Off-hook A event, there is no need to specify the parameter A explicitly. Furthermore, the event format in Table 4-2, if applied to message names in MSC, may cause some inconsistencies between the actual participants of the event. For example, the Start Ringing A B event defines A as the recipient node of the event. However, mistakenly drawing the arrow of the message towards B will create confusion between the meaning of the event and the actual recipient node in the MSC specification. To avoid this and to make the specification more concise, we adjust the format in Table 4-2 as follows:

User to Switch	Switch to User
Off-hook A	DialTone A
On-hook A	Start AudibleRinging A B
Dial A B	Start Ringing A B
Flash A	Start CallWaitingTone A B
	Stop AudibleRinging A B
	Stop Ringing A B
	Stop CallWaitingTone A B
	LineBusyTone A
	Announce A R
	Disconnect A B

Table 4-2: POTS Events (based on Bellcore's Generic Requirements for Switching Systems [26]).

The format of a message name is:

Message_Name ::= <event> <Instance_Name> ['(' parameter_list ')']
parameter_list ::= <parameter> {',' <parameter>}

where <event> is a base event name (i.e. without the extra parameters) as presented in Table 4-2; <Instance_Name> is a name of the instance as defined by the naming conventions in section 4.3.1, **excluding** the sending and receiving parties of the message; and <parameter> is an additional information carried by the current event such as a recording. In other words, we show only parties that are not directly involved with the message and other information (such as recordings) that the event carries.

We will illustrate this format in a typical POTS specification as shown in Figure 4-2.

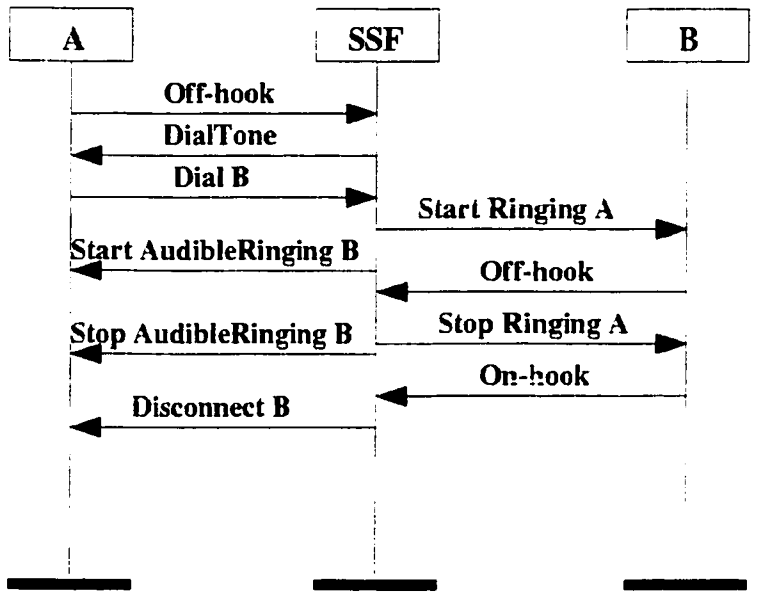


Figure 4-2: A POTS Specification Using the Event Format.

In this example, from the top to the bottom, the messages exchanged, when expressed with the format in Table 4-2, would be:

```
Off-hook A, DialTone A, Dial A B, Start Ringing B A,
Start Audible Ringing A B, Off-hook B, Stop Ringing B
A, Stop Audible Ringing A B, On-hook B, Disconnect A
B.
```

Our format is intuitive: the `Start Ringing A` message drawn to B, for example, indicates that the ringing happens at B's telephone, but A is indirectly involved in sending the message. Addressing A's role is particularly useful when we want to examine static requirements of the feature. We will elaborate on this in Section 4.3.4.

4.3.3 PICs, Initial Trigger and GOAL

In section 4.3.1 we have described the representation of BCSMs with MSC. We will now discuss the MSC representation of four other properties that are related to feature interaction detection: PICs, Initial Trigger, and GOAL.

Recalling that a PIC is defined as a set of states in the life of a call at a switch, we would naturally model PICs using MSC states attached to the BCSMs. The format of the state name is:

```
PIC_Name ::= 'PIC' <number> | 'O_Exception' | 'T_Exception'
                   { ',' <number> | 'O_Exception' | 'T_Exception' }.
```

where PIC is a reserved keyword that identifies that the state represents a PIC; <number> is the number of the PIC which we have listed in Appendix A. If the current PIC is O_Exception or T_Exception, they do not have numbers, so we will represent them using their full names, i.e., PIC O_Exception or PIC T_Exception where O_Exception and T_Exception are also reserved keywords.

As an example, we present in Figure 4-3 the MSC specification of the Originating Call Screening(OCS) feature. This feature prevents the caller from reaching certain directory numbers if the called directory number is on his/her screening list. Figure 4-3 presents the behavior of OCS in the scenario that the called party is not on the screening list of the calling party. Here, we can see how we model PICs of the OCS feature. Here `PIC 2,3` denotes that PICs 2 and 3 are passed one after another. Since there are no events sent in between these two PICs, this abbreviation will not create any ambiguity.

Since the Initial Trigger is associated with actions that do not belong to the normal processing of the call, we will model it using the action construct in MSC. The format of the action name of the Initial Trigger is :

IT_Name ::= 'IT' e<number> : <Trigger_Name> ['(' <Priority> ')']
Priority ::= <Natural_Number>

where IT is a reserved keyword that identifies that the action is an Initial Trigger; e<number> represents the TDP where <number> is a TDP number listed in Appendix A; <Trigger Name> is the full name of the trigger that is also taken from Appendix A; <Natural_Number> is a positive integer. The <Priority> is an optional entry to a trigger specification. As shown in Figure 4-3, the Initial Trigger for OCS is represented as `IT e3: OffhookDelay`.

We can now represent the GOAL of the feature. The GOAL is defined as the point in the BCSM where we can consider the objective of the feature as fulfilled. We will represent this by attaching a GOAL label to the PIC where the aim of the feature can be considered as realized. The following Definition defines the format of such a PIC.

GOAL_PIC :: = <PIC_Name> : 'GOAL'

In Figure 4-3, the GOAL of OCS is represented as `PIC4 : GOAL` which means that the GOAL of OCS is reached at PIC4, i.e. after the called party authorization process is complete.

Now that we have represented the initial trigger and the GOAL, the activation period (i.e. the period beginning at the initial trigger and ending at the GOAL) is obvious. Corresponding to two BCSMs, there are two traces for the OCS feature. If we use a "→" to represent state transition, "!" to represent sending of messages and "?" to represent the receiving of messages, then we can represent the traces of OCS in the activation periods in the following format:

$$O\_BCSMA : PIC_3 \xrightarrow{!DialTone.?DialB.!QwP(Offhkdel.callingDN.CalledDN).?Authorize\text{-}Origination} PIC_4$$
$$T\_BCSMB : PIC_{11}$$

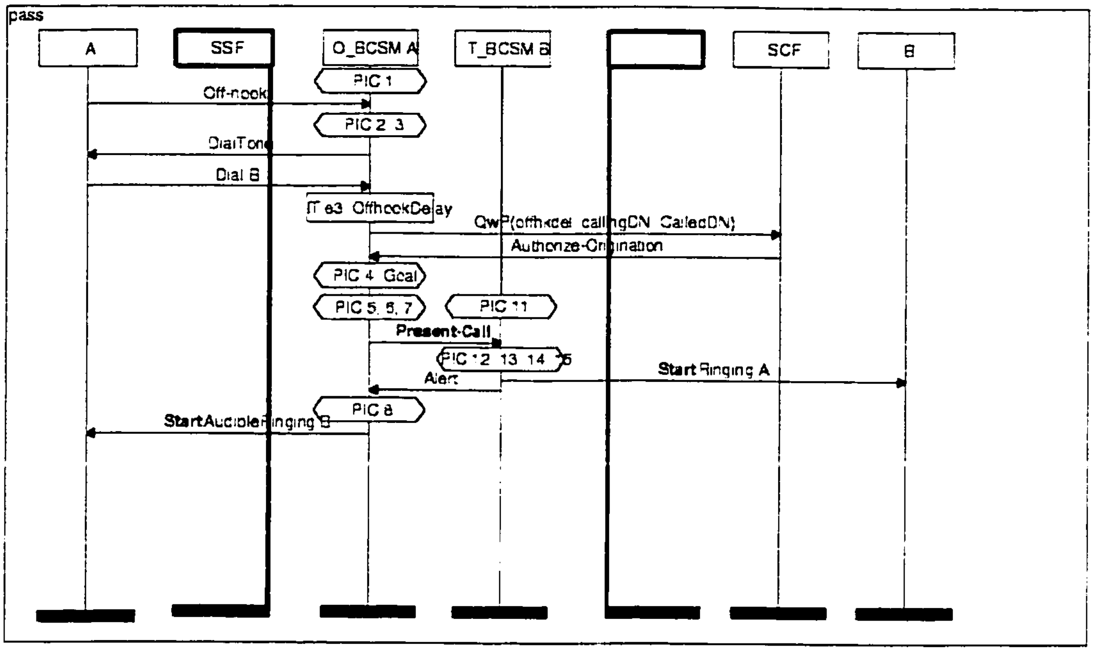


Figure 4-3: MSC Specification for the OCS Feature.

4.3.4 Static Requirements

Static requirements on a feature usually stipulate that some state of the call is not allowed. For example, a typical static requirement is that callers should not be allowed to connect to themselves. To specify such a requirement, we may choose to use temporal logic formulas or other textual requirement specification languages. By adding declarations of formulas or simple specifications to the MSC model, we can specify static requirements [28]. However, one has to define the format of the formulas and this is the equivalent to defining a new language. Because MSC itself is a good requirement specification language, we do not need to introduce other languages and can instead specify static requirements directly on a feature, using a separate MSC diagram. However, the format of the specification is such that it defines a pattern that will lead to violation of the static requirement. We shall see why it is beneficial to specify static requirements this way in Chapter 5.

Let us look at an example. A major requirement on OCS is that callers should not be allowed to connect to a directory number which is on their screening list if they call the number. Suppose A has OCS and C is on A's screening list, then the static requirement on the OCS of A can be represented as shown in Figure 4-4. This requirement says : the Start Ringing A message drawn toward C is prohibited. From this example, we can see that the format of the message name plays a key role. If the parameter A is left out, then it will not be clear who is indirectly involved in sending the "Start Ringing" message, especially when a third party (e.g. B) is present.

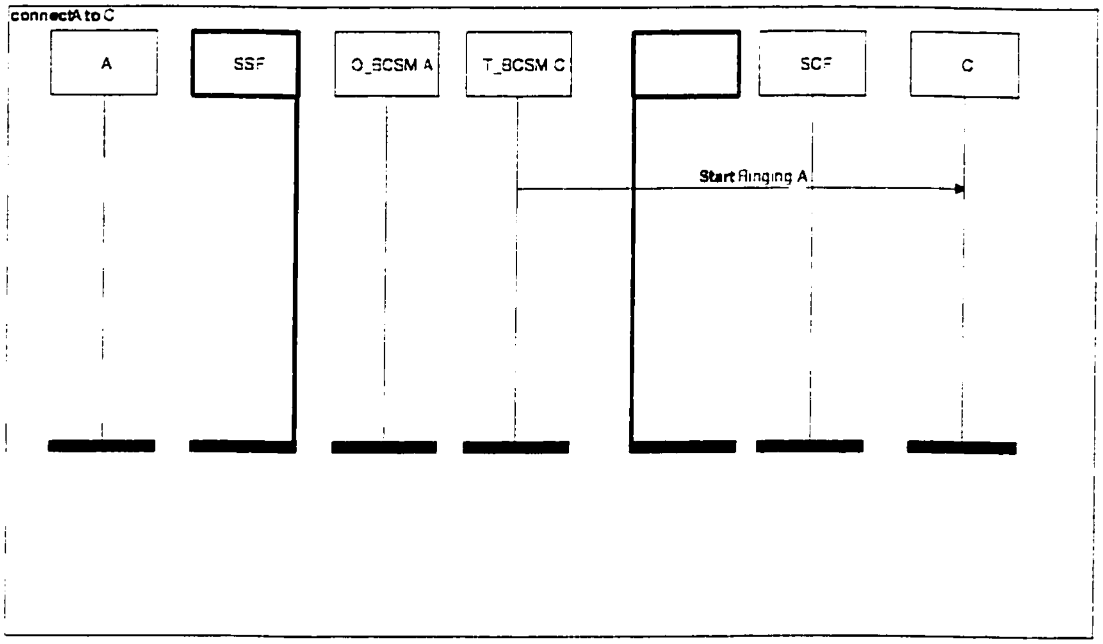


Figure 4-4: MSC to Specify Static Requirement of OCS.

4.4 The Specification Procedure

As a summary of the specification styles presented so far, we provide here general procedural guidelines on how to specify features using these styles:

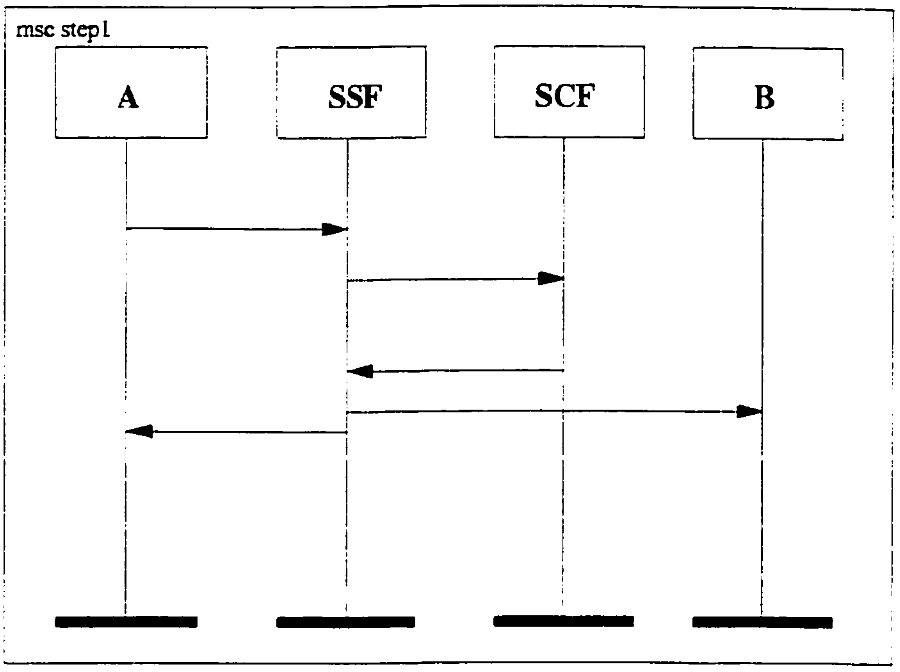


Figure 4-5: Specification Procedure. Step 1.

Step 1: Identify typical call scenarios for the features under study. In each scenario, identify network elements participating in the call. By applying the naming conventions presented in section 4.3.1, draw instances of the MSC models. Following the naming conventions for messages presented in section 4.3.2, identify the characterizing messages of the feature and draw top-level messages sent in between the network elements and users. We provide an illustration of this step in Figure 4-5.

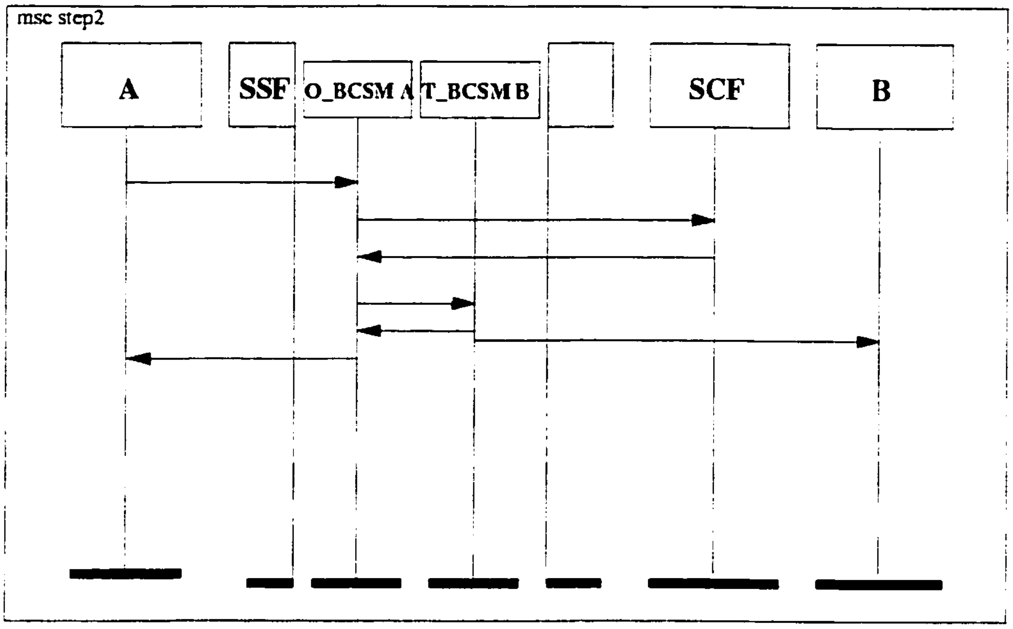


Figure 4-6: Specification Procedure, Step 2.

Step 2: Refine the SSF to a submsc containing O-BCSM and T-BCSM instances. Connect external messages to the SSF with the BCSMs. In between the O-BCSM and T-BCSM instances, draw messages to show the internal events. For illustration of this step, see Figure 4-6.

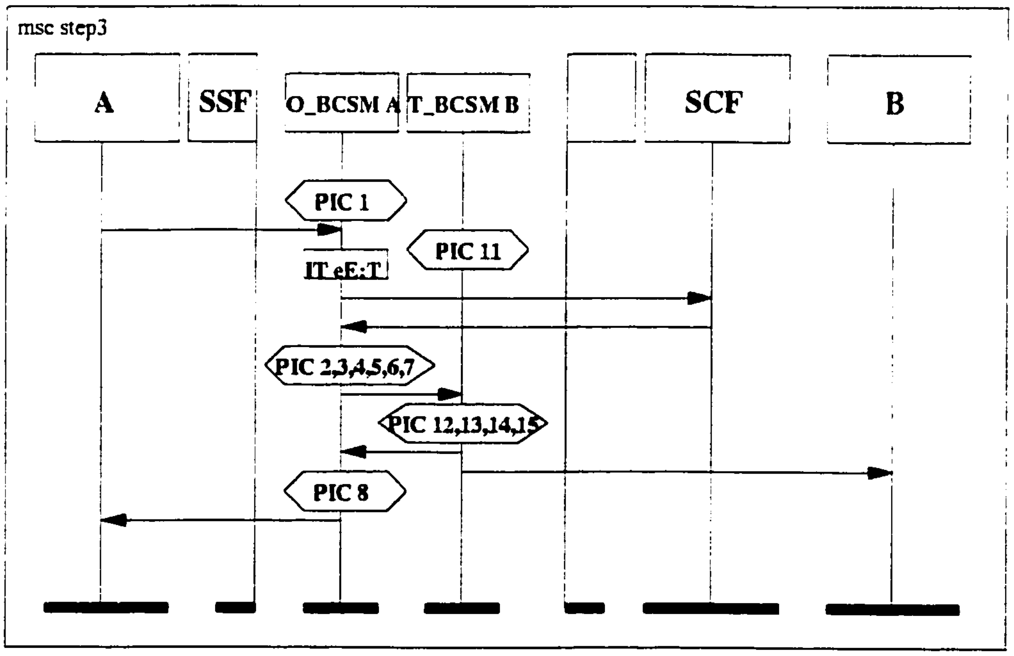


Figure 4-7: Specification Procedure, Step 3.

Step 3: Identify the initial trigger that activates a feature. Following the naming conventions presented in Section 4.3.3, draw relevant PICs to show the relative position of the initial trigger and to mark the traces of the feature. In Figure 4-7, the internal action with the `IT eE: T` denotes the initial trigger for this feature. Here, E refers to the TDP number and T is the name of the trigger. We show all PICs as states in the MSC model. Note that `PIC 2,3,4,5,6,7` denotes that PICs 2, 3, 4, 5, 6 and 7 are passed one by one. Since there are no external events sent in between these PICs, this abbreviation will not create any ambiguity.

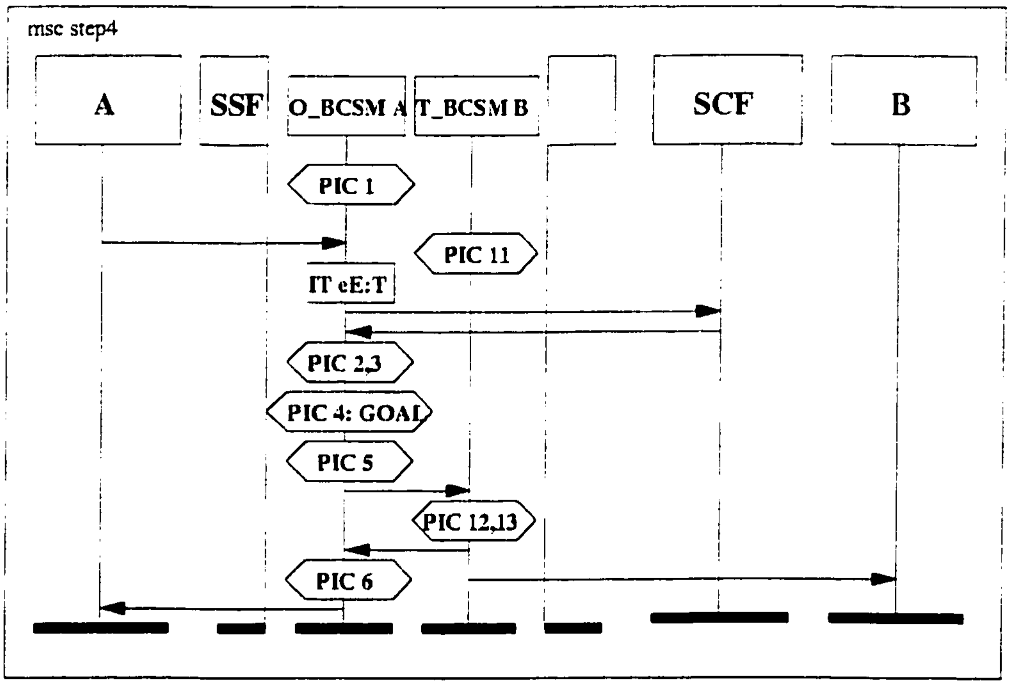


Figure 4-8: Specification Procedure, Step 4.

Step 4: By analyzing the characteristics of the feature, we identify the GOAL of the feature. We show this by attaching a GOAL label to the PIC where the objective of the feature can be thought of as fulfilled. The activation period of the feature is the period beginning at the initial trigger and ending at the GOAL. In figure 4-8, as we can see, the GOAL for this feature is reached on PIC 4. Note that in this instance, we should separate this PIC from others because, if we do not, we cannot attach the GOAL label to it.

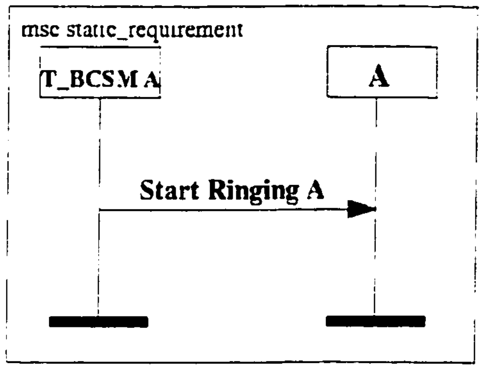


Figure 4-9: Specification Procedure, Step 5.

Step 5: We specify static requirements of the feature in separate MSC diagrams. Figure 4-9 specifies a typical requirement on most features, i.e. callers should not be allowed to connect to themselves. This is shown by a pattern, i.e. the sending of the `Start Ringing A` message drawn toward A, that violates the requirement.

In Chapter 6 of this thesis, we will present MSC specifications of many features described in the Bellcore Feature Interaction Benchmark [5]. We hope this will help readers quickly adapt to the way we specify real telecommunication features with our specification style.

4.5 Comments on the Feature Specification Style

As an evaluation of the specification style presented in this chapter, we summarize a few characteristics of this specification style:

1. **High-level specification of feature behavior**

   The specification style presented in this chapter represents a high-level specification of feature behavior. The viewpoint of the specification is very close to the user's viewpoint to telecommunication features. Although, contrary to many pure requirement specification methods. we have now reached the point where we can illustrate a few internal details of the network, such as the basic call models and triggers. These concepts are relatively easy to understand by users with basic programming knowledge. Other network details such as memory usage or process control are not included in our specification style, and thus, transparent to the feature specifier.

2. **Fast prototyping tool**

   The specification style introduced in this chapter can be used as a fast prototyping tool for telecommunication features. It can be considered as a template for feature specifications. The reader may consider the feature specification procedure we suggest as a standard way to fill up the template and produce specifications in an efficient manner.

3. **User friendly**

   As mentioned previously, since the specification style is designed to produce high-level specifications of feature behavior, users with basic programming knowledge will be able to understand the concepts involved. In addition, we have selected intuitive MSC structural mappings and naming conventions in order that the specification style is easier for normal users to learn.

4. Limitations

Some of the limitations of this specification style are the direct consequence of our design intention, i.e. to produce high-level specifications of feature behavior. This perspective provides us with all the benefits summarized above. However, since many low-level behaviors of features are ignored, feature interaction detection methods based on this specification style generally cannot detect interactions that happen between low-level behaviors. Our approach, to be presented in Chapter 5, in particular, cannot detect many low-level interactions. We will see this consequence clearly in the benchmark to be presented in Chapter 6.

For simplicity, our current specification style does not consider physical locations of network components. However, many interactions happen due to confusions between physical entities and their functional assignments. For example, in Chapter 6, we have two interactions that involve directory numbers and their assignments to physical lines. Because of our simplification, we are unable to detect these interactions.

Feature interactions that give rise to inconsistencies in charging constitute a considerable amount of interaction cases. Because of the time limitations for this project, our current specification style is unable to include the means to model charging aspects of features, and our approach, presented in Chapter 5, is unable to check behavioral preservations on charging aspects, and thus unable to detect these interactions. Again, in Chapter 6, the benchmark clearly shows this limitation.

Some feature interactions arise due to the synchronicity of feature behaviors. In Chapter 6, we will present one example that renders an interaction of this type. Due to the fact that MSC currently supports only modeling of asynchronous behavior, we are unable to address synchronicity in our specification style in MSC.

Chapter 5

Classification and Detection of Feature Interactions

This chapter describes the complete methodology, techniques and algorithms employed in the detection of feature interactions as part of our general framework. The chapter is organized as follows:

- Section 5.1 derives a definition of feature interactions based on our viewpoint of the feature interaction problem.
- Section 5.2 gives a classification of feature interactions.
- Section 5.3 emphasizes the importance for finding interaction-prone call scenarios.
- Section 5.4, 5.5 and 5.6 introduce the methodology and techniques for detecting various types of interactions.
- Section 5.7 describes the algorithms that implement the detection techniques.

5.1 Definition on Feature Interactions

To date, we do not have a standard definition of feature interactions. However, there are several formal and informal definitions in use today that partly describe the nature of the problem. In the proceedings of the Second Feature Interaction Workshop, a feature interaction is defined as follows [4]:

> *"A feature interaction occurs when the behavior of one feature is changed by the behavior of another. In many cases, this can lead to unexpected or undesired behavior which affects the quality of the service*

provided to telecommunications users."

The above not only defines an interaction, it also indicates that some interactions are expected and desirable while others may be unexpected or undesirable. From this definition, we can see that a necessary condition for feature interaction is that the behavior of any one of the features is changed. In other words, we can detect possible interactions by checking the preservation of feature behavior. Therefore, in this thesis, we use the following definition:

There is a feature interaction if the behavior of any one of the participating features is not preserved.

Sometimes, non-preservation of feature behavior does not necessarily mean there is harmful feature interaction. For example, in the case of the OCS feature presented in Chapter 4, blocking of unauthorized calls is desired, although it may change the behavior of many features. For simplicity, our definition does not distinguish between desired and harmful interactions. Therefore, additional actions will have to be undertaken to detect harmful interactions.

5.2 Classification of Feature Interactions

We have shown previously that, to detect a feature interaction, several important properties of features should be taken into consideration. These properties include the initial trigger, the traces and static requirements on the feature. In Chapter 4, we have presented our feature specification style which takes these properties into consideration. Here, we would also like to classify feature interactions according to these properties.

As mentioned previously, our methodology is to check for preservation of feature behavior. Since, when viewed from the high-level, the initial trigger, the traces and static requirements constitute the most important characteristics of a feature behavior, we can say that, at the high-level, the behavior of a feature is preserved if and only if :

a. its initial trigger can be reached (not blocked by other features)

b. its traces in the activation period are preserved (not modified by other features)

c. static requirements of the feature are respected.

Thus, our informal definition of feature interactions is equivalent to the following:

Feature Interaction: We define as a Feature Interaction any violation of **a**, **b**, or **c**.

In other words, we can classify feature interactions into 3 categories:

a. Blocking of Initial Triggers,

b. Interaction between Traces and

c. Violation of Static Requirements

In the following sections, we will illustrate our techniques for detecting each categories of interactions classified above.

5.3 Call Configurations and Scenarios

When analyzing interactions between features, it is necessary to consider the interactions in each call configuration and each call scenario. For example, to analyze the interactions between Call Waiting (CW) and Three Way Calling (TWC), we must consider on which parties these two features are enabled. And if the call scenario is changed, e.g. A calls B is changed to B calls A, it will also change the interaction situation. While the possible combinations between two features can be enormous, Keck [27] has presented a technique to filter through all call scenarios and identify those scenarios which are interaction-prone. In simple cases, however, one can simply take all combinations and skip the filtering process.

In this thesis, we will assume that all scenarios which are subject to further study have been identified. We will study feature interactions in each identified scenario. Based on our framework, we will study interactions related to the initial trigger, the dynamic behavior (represented by traces) and static requirements of the feature.

5.4 Blocking of Initial Triggers

Note that the prerequisite for two features to interact is that the trigger criteria of the features can coexist. In particular, the trigger criteria for O_Called_Party_Busy and O_Called_Party_No_Answer or T_Busy and T_No_Answer cannot coexist in a two-party call, since the terminating party is either busy or not busy; there is no alternative. The trigger criteria for different digit strings cannot coexist in a two-party call either. For example, 911 service and Freephone service do not interact in a two party call because their triggers are different digit strings, one is 3 digits and the other is 11 digits. The user can dial only one number to initialize a call.

Blocking of Initial Triggers: Let us suppose that the trigger criteria of two features can coexist in the current call configuration. If it is possible for one feature to block the other feature from reaching its trigger, then the two features are said to have a "Blocking of Initial Trigger" interaction.

Based on careful observation of trigger conditions and positions, we identified four situations that can arise due to trigger criteria and positions of triggers:

a. No interaction

b. Blocking of Initial Trigger

c. No Blocking of Initial Triggers and no possibility of interaction between traces; however, there may be violation of static requirements.

d. No Blocking of Initial Triggers although there may be any other type of interactions.

Table 5-1 gives a summary of these situations and their corresponding trigger conditions and positions.

Interaction Situation	**Cases**
No Interaction	Trigger criteria do not coexist
Blocking	1. Activation periods do not overlap and at least one feature does not pass by the trigger of the other feature 2. Two features share the same trigger in the same BCSM or two triggers reside in the same TDP of the same BCSM and there is no priority. At least one feature does not get back to the trigger of the other feature 3. The feature with earlier trigger or trigger with higher priority does not pass by the trigger of the other feature
No Blocking, No Trace Interaction	Activation periods do not overlap and two features pass by the trigger of one another. Check violation of Static Requirements.
No Blocking Only	Other situations. Continue to check all the other interactions

Table 5-1: Interaction Cases Related to Initial Trigger.

In Table 5-1, "pass by" or "get back" refer only to peer traces. If the trigger of one feature resides on a trace that the other feature does not have, then by default, the second feature also passes by this trigger. This is because, the call control will not prevent this trace from being instantiated. Once started, the first feature has total control and can ensure call processing to pass by the trigger.

Detecting Blocking of Initial Triggers is straightforward: simply look at each trigger criteria, the position of the two triggers and GOAL and the traces if necessary. Check the four interaction classes in Table 5-1. If it is possible for one initial trigger to be blocked, there is a Blocking of Initial Trigger interaction. Otherwise, we should refer to Table 5-1 to see what actions should be undertaken to detect other types of interactions.

In order to illustrate this technique, we present an example of Blocking of Initial Trigger type of interaction. In Chapter 3, we have shown a feature interaction example between the CFU and CFB features in the scenario that the called party is busy. We have explained that related approaches may not be able to detect this interaction. Here, we will show how our approach detects this interaction as a Blocking of Initial Trigger.

In Figure 5-1, we present the MSC specification of the CFU feature in the scenario that the called party is busy. Figure 5-2 shows the MSC specification of the CFB feature in the same scenario. In the two specifications, we can see that the two features have overlapping traces. The trigger for CFU, i.e. TermAtt, resides in TDP e20 which, as we can see, is reached after PIC 11. On the other hand, the trigger for CFB, i.e. T_Busy, resides in TDP e30 which is reached after PIC 13. From this, we can conclude that the trigger for CFU is reached first. However, after reaching the initial trigger, trace of CFU corresponding to T_BCSM B clearly do not pass by PIC 13 which contains the

CFB trigger. Therefore, the situation between CFU and CFB in this scenario corresponds to the third "Blocking" case classified in Table 5-1. Thus, we detect a Blocking of Initial Trigger interaction.

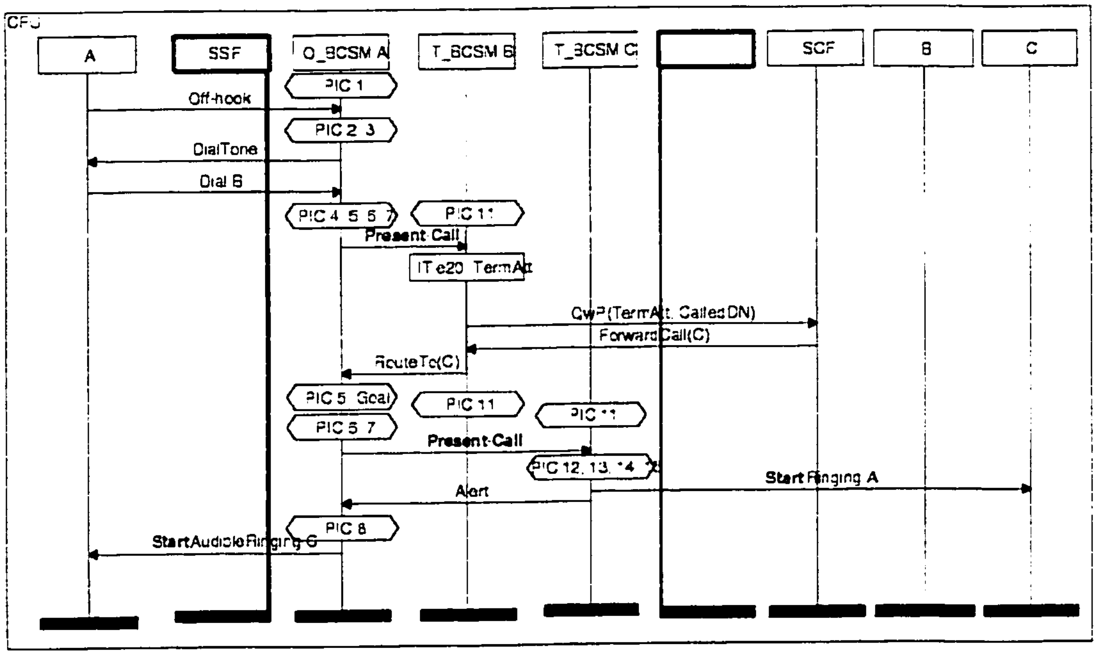


Figure 5-1: Call Forwarding Unconditional (CFU).

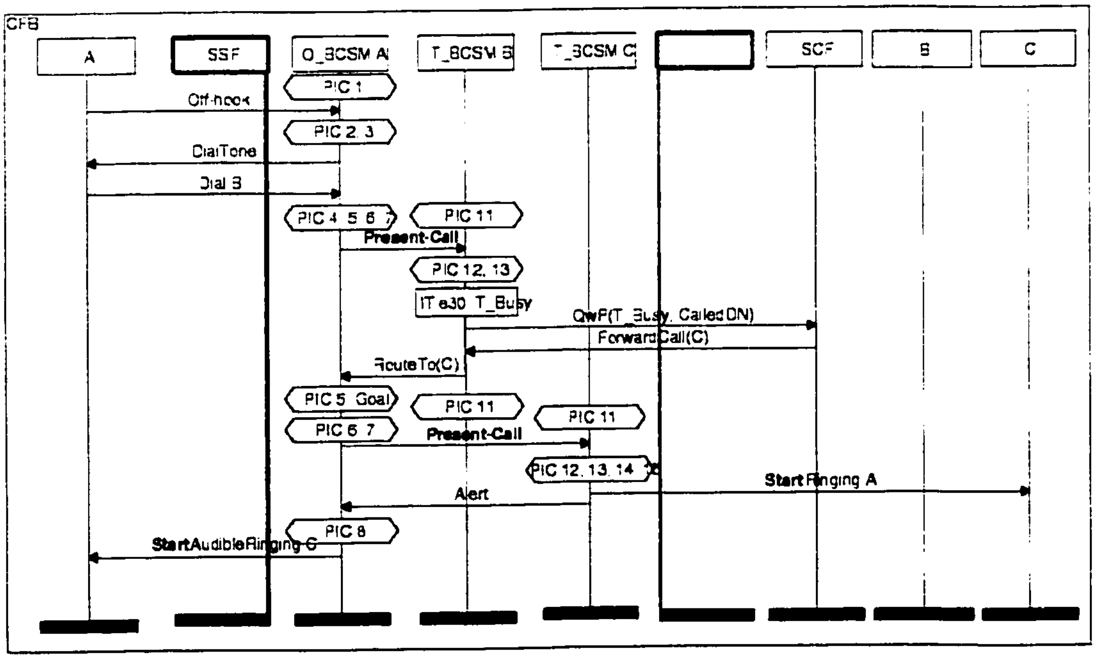


Figure 5-2: Call Forwarding on Busy (CFB).

5.5 Interaction Between Traces

In the fourth interaction situation classified in Table 5-1, there is no blocking of triggers. However, since activation periods overlap, traces of two features in the overlapping parts of the activation periods run in parallel. We must therefore examine the possibility of interaction between traces. In other words, in the overlapping parts of the activation periods, if at least one of the traces is not preserved, then there is an interaction between traces; if both can be preserved, then there is no trace interaction. Further examination shows that conflicts only happen on peer traces of two features (the traces that refer to the same BCSM) since the BCSM tries to execute both traces with possibly different behavior.

Since peer traces in the overlapping parts of the activation periods consist of transitions, we can examine the interactions at the transition level. According to finite state machine notation, every transition consists of three properties: a current state property, a next state property and actions taken on the transition. Note that peer traces in the overlapping activation periods always have a common starting PIC, because this is where two traces start to overlap.

Now, the question is: how can we determine if two corresponding transitions are conflicting? Here, we give a few examples to illustrate conflicting transitions. Suppose that "!" denotes the send action; "?" denotes the receive action; "→" denotes state change. Let us consider the following peer traces (which, as we can see, already synchronized in the first PIC states):

Trace 1: $PIC_a \rightarrow PIC_b \rightarrow PIC_c$
Trace 2: $PIC_a \rightarrow PIC_e \rightarrow PIC_f \rightarrow PIC_b \rightarrow PIC_c$

In this example, we did not consider the events sent and received in between the PICs. We only consider state changes of two traces. Suppose $e \neq b$. Obviously, these two traces have different destinations in the first transition:

trace 1 moves to PIC_b while trace 2 moves to PIC_e. Does this mean that the two traces are conflicting? The answer is: not necessarily. Suppose that the PIC indices take the order: e<f=a<b<c. In trace 2, the call control first goes to PIC_a, then it jumps to a previous state PIC_e (because, for instance, it needs to collect extra information), then passes through PIC_f, then PIC_b then PIC_c. We can see that all transitions in trace 1 are fulfilled in trace 2. In real-life situations, if two traces are running in parallel and take the above forms, then the BCSM will in all likelihood follow trace 2 since the "jump-back" action is active and the action to "pass on to next state" is passive. Now, we can see that although the transitions do not follow trace 1, all trace 1 transitions are preserved. In other words, there is no conflict.

However, the following pair of traces will not be able to run together because they conflict in respect to their transition destinations during the first transition. The BCSM will follow trace 2, since it has an active jump forward. After going to PIC_3, the BCSM will not be able to pass PIC_2. Trace 1 is thus suppressed.

Trace 1: $PIC_1 \rightarrow PIC_2 \rightarrow PIC_4$
Trace 2: $PIC_1 \rightarrow PIC_3 \rightarrow PIC_4$

From the above examples, we derive the following conclusion in regard to transition destinations:

Sometimes, a trace jumps to a previous state to collect extra information and comes back to the current state when this is accomplished. If this results in a different destination from that of a normal transition, we do not regard it as a conflict. However, in those cases where the trace does not come back, or jumps forward to some later PIC state, the transition conflicts with normal transitions.

In view of this conclusion, we cannot simply say there is conflict if two transitions have different destinations. To facilitate analysis of trace interactions, we propose a special form of trace ("SF trace"), wherein each transition is taken as the "giant step" for the current PIC to reach the next PIC that, according to AIN basic call model, has a number higher than the current PIC. We will consider transitions to previous states and all messages exchanged as actions on the giant step. In this SF trace, transitions do not conflict as to their destinations if and only if their destinations are the same. Bearing in mind that the first starting PICs are always the same, we can also say that peer traces in the overlapping parts of the activation periods do not conflict on any transition destinations if and only if their SF peer traces contain the same set of PIC states.

In the following example, the peer traces are transformed to SF traces. Trace 2 does not change since it does not contain transition to previous states. In this example, "$\rightarrow PIC_3$"denotes the action of going to a previous state PIC_3. We can see in this example that each of the resulting two traces consist of only one transition, and their destinations are the same. In other words, they do not conflict on their destinations.

$$\begin{aligned} Trace1: & \; PIC_4 \rightarrow PIC_3 \rightarrow PIC_4 \xrightarrow{!a.?b} PIC_5 \\ & \Rightarrow PIC_4 \xrightarrow{\rightarrow PIC_3.\rightarrow PIC_4.!a.?b} PIC_5 \end{aligned}$$

$$Trace2: PIC_4 \xrightarrow{!a.?c} PIC_5$$

We have represented conflicts between transition destinations. We do not need to examine conflicts between transition origins because the first starting PICs of overlapping peer traces are always the same, and we are always checking conflicts between destinations from the top to the bottom. Absence of conflicts in the previous transitions automatically guarantees that the origin PICs of the next peer transitions are the same.

Agreement of transition destinations: Two transitions in SF traces are said to disagree as to their destinations if they have conflicting (different) destination PICs. They are said to agree as to their destinations if their destination PICs are the same and this common destination PIC is said to be the agreement of transition destinations.

We now consider the conflicts between transition actions. As we can see from the previous example, actions on a transition form a sequence. If we consider each action as a state, then the sequence of actions looks similar to a trace, though with no further actions on each transition. Regarding them in this way, we can determine their conflicting situations just as we would for peer traces. However, unlike PICs, individual actions do not take any orders. In other words, there is no jumping forward or jumping backward between actions. For this reason we need to define conflicts between actions in a somewhat different way.

Now, suppose we have two transitions (already in the special form):

$$Transition1: PIC_4 \xrightarrow{!a.?b.!e} PIC_5$$

$$Transition2: PIC_4 \xrightarrow{!a.?c.\rightarrow PIC_4.!d.?b.!e} PIC_5$$

We call individual actions such as "!a" a node, and a pair of neighboring actions such as "!a.?b" an edge. We can see in this example that the edge !a.?b in Transition 1 has a corresponding sub-sequence !a.?c.→PIC_4.!d.?b in Transition 2. We call !a.?c.→PIC_4.!d.?b an extended version of the edge !a.?b, since besides !a and ?b, the former contains some add-on actions. The edges ?b.!e in both transitions are in the same format; thus we call them equal.

When these two transitions are running together, the BCSM will try to execute both transitions. Transition 2 has some add-on actions from the feature. these usually have higher priorities, so Transition 2 will be executed. Since the BCSM will also execute the sequence of actions that Transition 1 requires. we can conclude that Transition 1 is not influenced by the presence of Transition 2. Therefore. actions of Transition 2 appear to be the final action sequence to be executed. we call it the agreement of the two action sequences.

We can see from this example that each of the edges in the two action sequences belong to one of three types:

1. It has an extended version in the peer sequence. For example. !a.?b in Transition 1.
2. It participates in constructing an extended version for an edge in the peer sequence. For example. !a.?c in Transition 2.
3. It has an equal edge in the peer sequence. For example. ?b.!e in both Transitions.

And there is still another type of edge:

4. none of the above three.

When the fourth type of edge exists in either sequence. we can conclude that this edge is only recognized by one sequence and that it will not satisfy the needs of the other sequence. Thus. existence of this type of edges means the interests of two sequences are in conflict.

On the other hand, if all edges in two sequences are of the first three types, then they tend to reach an agreement. The agreement consists of extended versions and equal edges.

Agreement between action sequences: If two action sequences consist of only edges of the first three types, then they are said to agree, and their agreement consists of extended versions and equal edges. If any of them contain an edge of the fourth type, they are said to disagree.

A null sequence (that is, no actions) is said to agree with any other sequence and the agreement is the other sequence.

We follow with a procedure for calculating agreements of two transition action sequences. We will illustrate this procedure with an example provided in Figure 5-3.

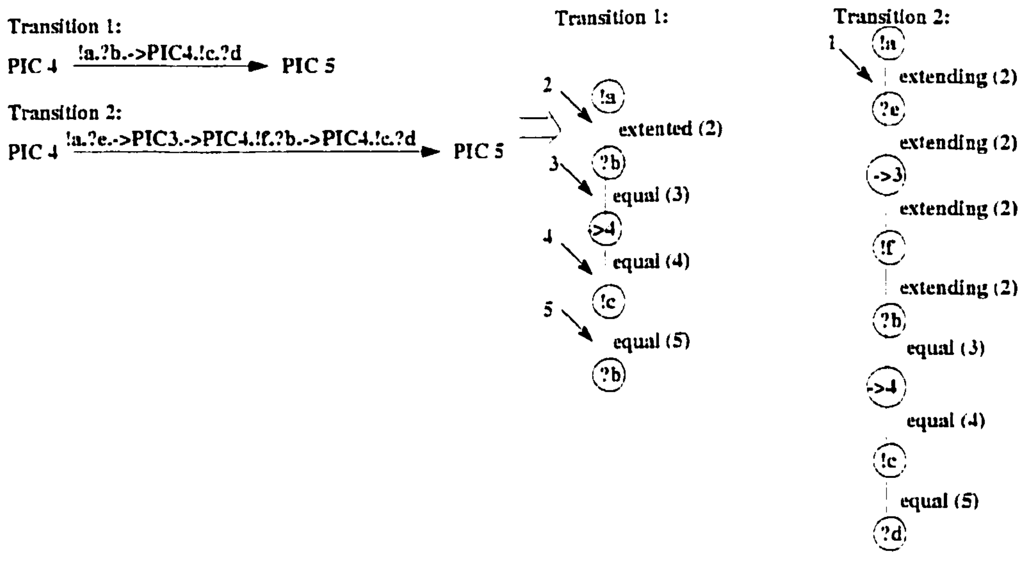


Figure 5-3: Transforming and Labeling Action Sequences.

Procedure for calculating agreement between action sequences:

Step 1. Transformation

Transform both action sequences into the nodes and edges graphical format illustrated in Figure 5-3. Here the nodes of the graph represent single actions which take effect in this transition phase. The edges in the graph denote the sequential relationship between actions. For the backward transition actions, only the action that takes effect in this phase, i.e. the first jump, is shown. In

Figure 5-3, we can see that the backward transition period $\rightarrow PIC_3.\rightarrow PIC_4$ of Transition 2 is shown in the graph by a $\rightarrow 3$ node.

Step 2. Labeling: Following is a labeling procedure on the edges of the graph which simulates the negotiation process between two action sequences:

1. From the beginning, all edges are assumed to be unlabeled. Take an unlabeled edge from the top of either graph and proceed to 2, below.

2. Compare the edge with the first unlabeled edge in the other graph. If they are equal, label both of them with an "equal" label, and repeat 2. If they are not, see if an extended version of this edge exists in the other graph. In this context, "extended version" means all edges taken before reaching a node that is equal to the next node of the current edge. If an extended version exists, the current edge is labeled "extended", while all edges in the extended version in the peer graph are labeled "extending". If a label has been successfully applied in this step, then a "failure counter" is reset to zero. If neither an equal edge nor an extended version of the current edge exists, then no label can be applied. Increase the failure counter by one and go to 3, below.

3. Check if the failure counter is equal to two. If so, this is the second consecutive time that the labeling process in 2 has failed. Therefore, the labeling process cannot proceed on any graph, meaning that the negotiation process had failed. The two action sequences cannot reach an agreement. In other words, their agreement is invalid. However, if the failure counter is equal to one, there is still hope that this edge belongs to an extended version of an edge in the peer. So, take the first unlabeled edge in the peer graph and go to 2, above.

4. If at any time during the process, either graph reaches the bottom node (meaning that all edges of this graph have been labeled), determine if there are still unlabeled edges in the peer graph. If so, all actions after the unlabeled edge are those that should be combined with the agreement sequence reached so far since these are add-on actions from the feature as well. Whatever is the case, we should continue to examine backward transitions in 5, below.

5. We do not need to examine a backward transition if the edge before it has an extending label, since the backward transition only exists in one action sequence. If it has an equal label, then go inside both backward transition periods and repeat the entire process for deciding agreements on all transitions in the backward transition periods. If the traces in the backward transition periods agree with each other, take their agreements and go to 6, below. If this not the case, the two action sequences disagree.

6. The agreement of the two action sequences consists of all edges that have an extending label and all edges that have an equal label.

In Figure 5-3, we can see this labeling procedure clearly. The steps taken are marked by numbers with an arrow pointing to the edge currently being examined. After the labeling procedure, the agreement of the two action sequences is equal to that of Transition 2.

Agreement between transitions: Two transitions in the special form are said to disagree if they either disagree on their transition destinations or they disagree on their transition action sequences. Otherwise, they are said to agree, and their agreement consists of the agreement of their transition destinations and agreement of their action sequences.

Based on the above analysis and definitions, we define trace interactions as follows,

Interaction between Traces: We say that two features have interaction between traces if any of their peer traces in the overlapping activation periods disagree on a specific transition.

According to the above definition of trace interaction, we can see that when two traces do not interact, then all their corresponding transitions must have agreements on transition actions and destinations. This agreement gives us some indication as to how to combine two traces: if two traces agree on all their transitions, then they can be combined to form one trace, which is made up of all agreements of the corresponding transitions.

The following example of trace interactions further illustrates this technique for detecting trace interactions:

Example:

This is an example in the Bellcore benchmark [5]. Suppose during a phone conversation between B and C, an incoming call from C has arrived at the switching element for B's line and triggered the Call Waiting feature that B subscribes to. However, before being alerted by the Call-Waiting tone, B has flashed the hook, intending to initialize a Three-way Call. The ambiguity here is whether the flash-hook signal should be considered as the response for Call Waiting, or as an initiation for Three-way Calling.

Figure 5-4 gives the behavior of the call when the flash-hook signal is considered as the response for Call Waiting and Call Waiting is launched.

Figure 5-5 specifies the behavior when the flash-hook is considered an initialization for Three-way Calling and Three-way Calling is launched. Because both features pass by the trigger of the other feature, there is no blocking of triggers. And when the trigger for Three-way Calling is launched, the two features are running in parallel. However, in the peer traces corresponding to O_BCSM B, after outputting the On-hold signal, the two features disagree on the next action. Call Waiting issues an accept signal and prepares to connect to A, while Three-way Calling activates another instance of O_BCSM and prepares for the next call initialization. As shown below, if we transform the peer traces into our special form of trace, we can see the disagreement clearly.

$$\text{Call waiting O\_BCSM B}: PIC_9 \xrightarrow{?flash.!On-hold.!Accept}$$

$$\text{Three - way calling O\_BCSM B}: PIC_9 \xrightarrow{?flash.!QwP().?TWC-Authorized!On-hold.!activate}$$

In the labeling procedure, when the labeling proceeds to the !On-hold node, it cannot label any more. The failure counter reaches 2 and the procedure aborts and gives information that a conflict has been found.

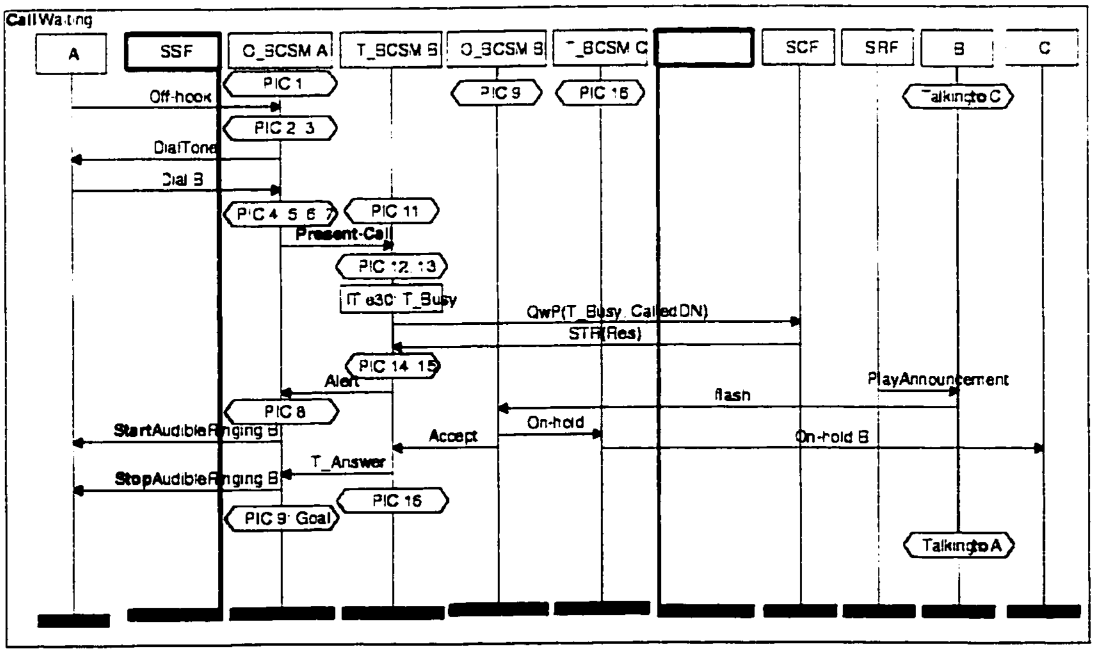


Figure 5-4: Call Waiting.

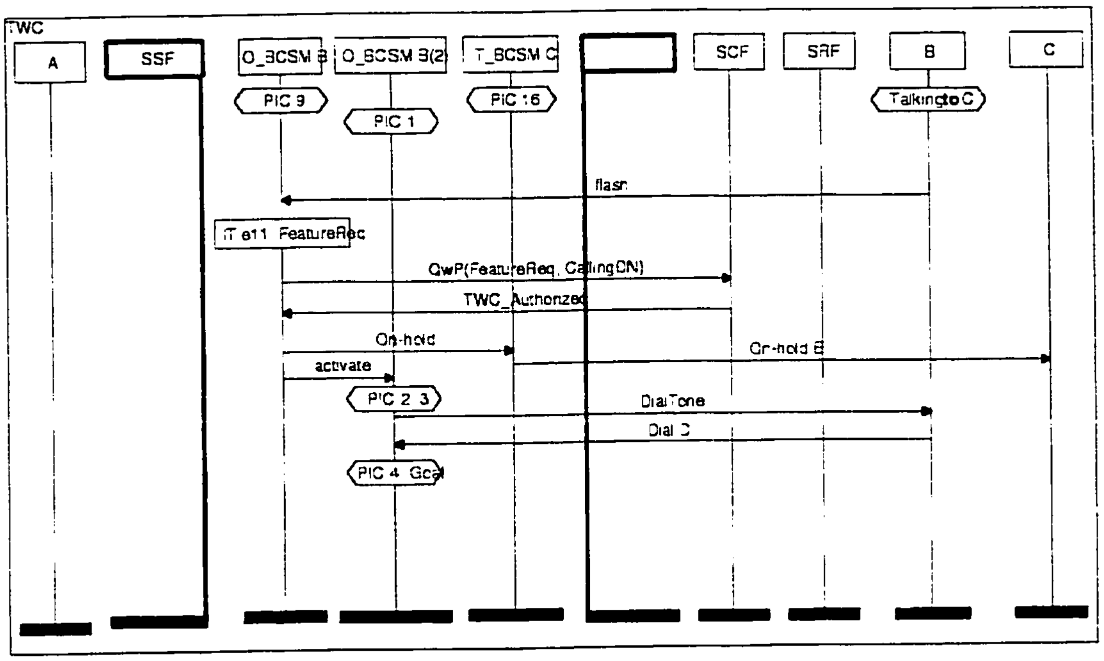


Figure 5-5: Three-Way Calling.

5.6 Violation of Static Requirements

When two features can coexist and there is no interaction between initial triggers and no interaction between traces, two features can be combined to form one trace. However, the combined behavior can sometimes violate some static requirements of one of the combining features.

Our specification style allows to use MSCs to specify violation patterns of static requirements. By performing a search algorithm on the combined trace, we can determine the presence of the violation pattern.

Violation of Static Requirements: In our framework, Violation of Static Requirement occurs when the violation pattern in the MSC specification of static requirements is found in the combined behavior.

In many situations, the violation pattern consists of only a single message, not a sequence of several messages. In this case, the search can instead be performed on each feature. This is particularly useful when the combined behavior is difficult to obtain.

Let us look at an example to illustrate violation of static requirements. The two features OCS (Figure 4-3) and CFNoA (Figure 5-6) do not have overlapping activation periods. Thus they do not interact on their initial triggers and their traces. However, their combined trace may violate a static requirement of OCS, which is that A should not connect to a number that is on the screening list. Suppose C is on the screening list of A. Since the violation pattern presented in Figure 4-4 contains only one message, instead of combining two traces and performing the search on the combined behavior, we can instead perform the search on each feature. In this example, we perform the search on OCS and CFNoA individually. We can see that the violation pattern, i.e. the Start Ringing A message connected to C, is present in the specification of CFNoA (Figure 5-6). So, we have detected a violation of static requirements of OCS.

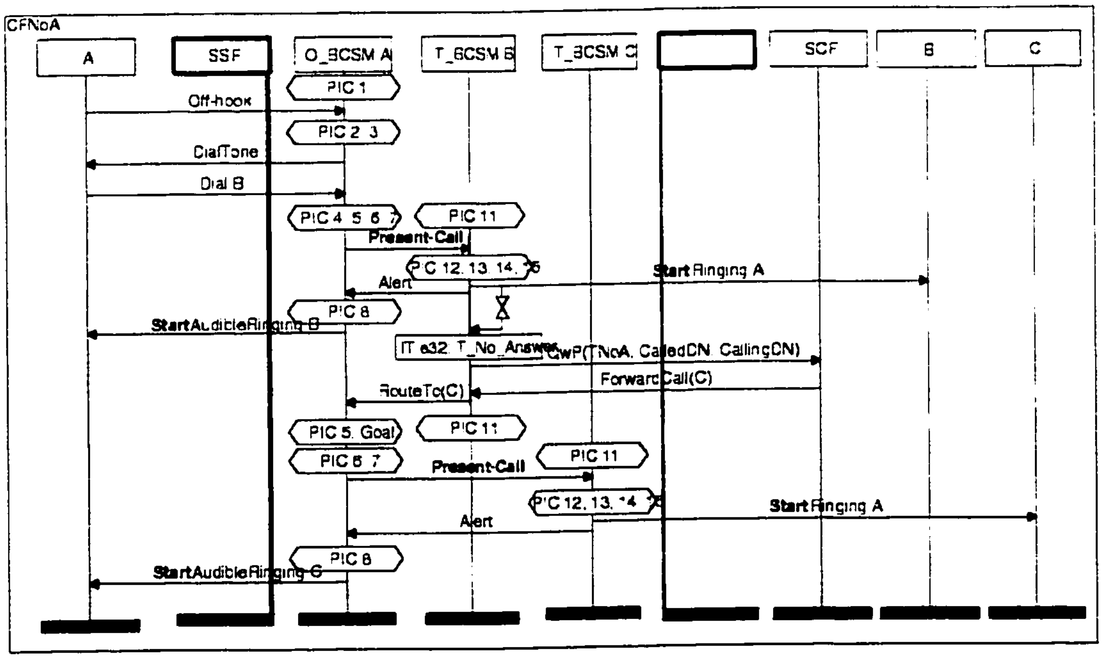
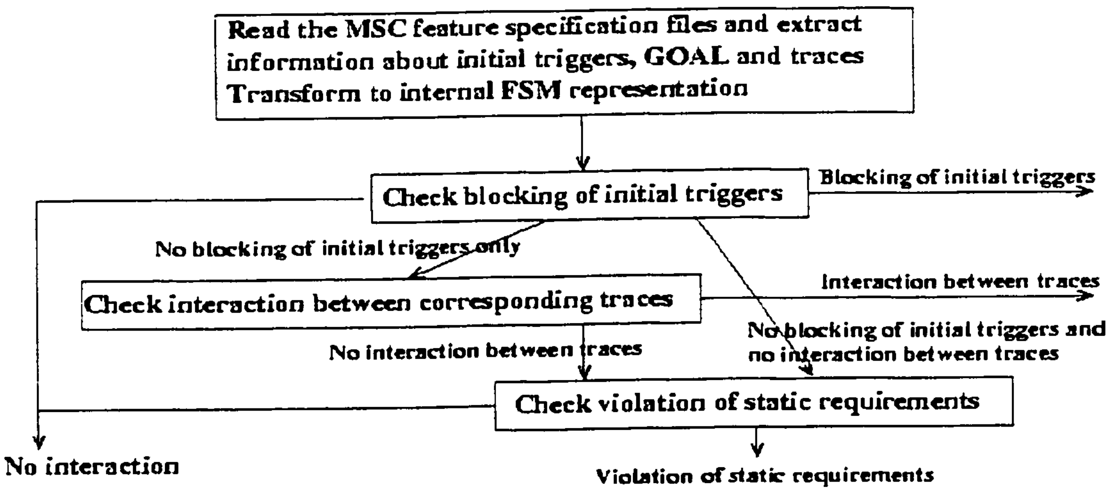


Figure 5-6: Call Forwarding on No Answer (CFNoA).

5.7 The Detection Algorithms

Let us emphasize again that our method is only applicable when the call scenario is specified. Thus one should always identify all possible scenarios first, then apply the method proposed in this thesis to each call scenario. For any given call scenario, assuming that two features are specified using our MSC specification style, Figure 5-7 shows the overall algorithm for detecting feature interactions.



Read the MSC feature specification files and extract information about initial triggers, GOAL and traces Transform to internal FSM representation
Check blocking of initial triggers
Blocking of initial triggers
No blocking of initial triggers only
Check interaction between corresponding traces
Interaction between traces
No interaction between traces
No blocking of initial triggers and no interaction between traces
Check violation of static requirements
No interaction
Violation of static requirements

Figure 5-7: The Main Algorithm.

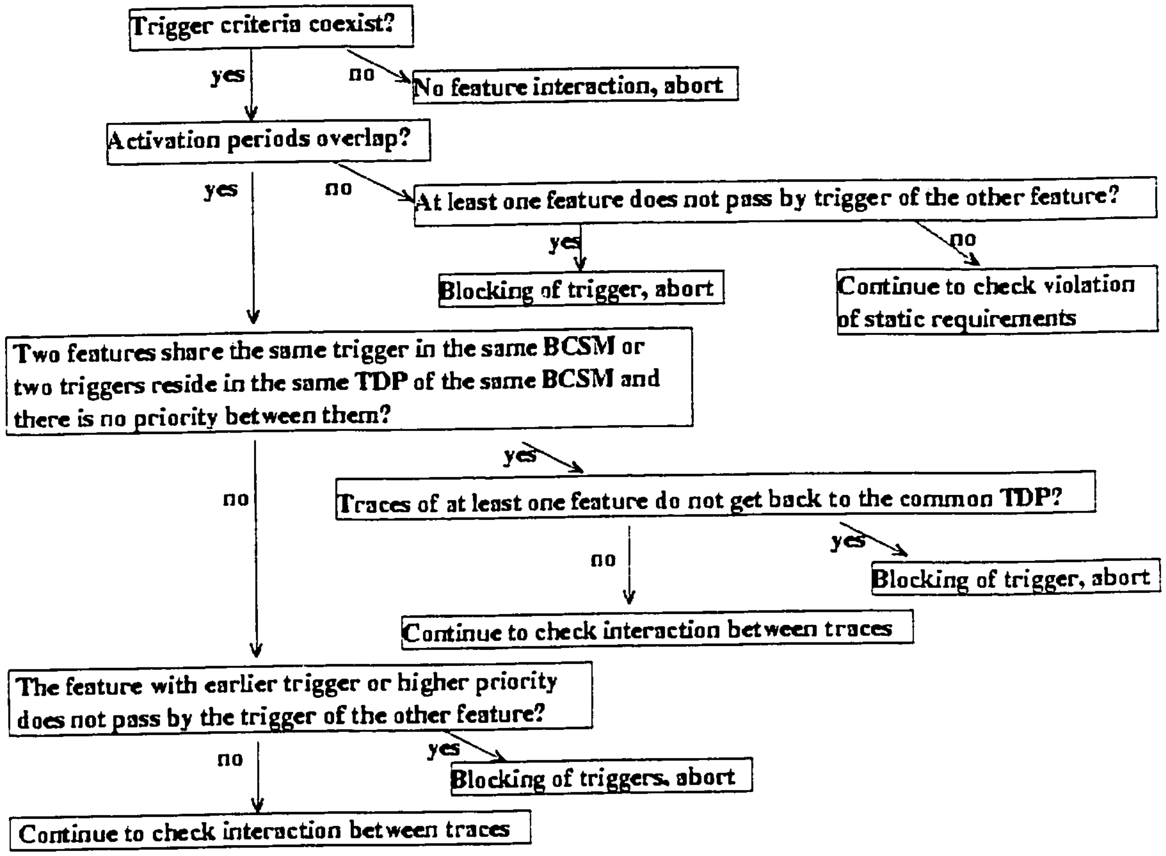

Trigger criteria coexist?
yes
no
No feature interaction, abort
Activation periods overlap?
yes
no
At least one feature does not pass by trigger of the other feature?
yes
no
Blocking of trigger, abort
Continue to check violation of static requirements
Two features share the same trigger in the same BCSM or two triggers reside in the same TDP of the same BCSM and there is no priority between them?
yes
no
Traces of at least one feature do not get back to the common TDP?
yes
no
Blocking of trigger, abort
Continue to check interaction between traces
The feature with earlier trigger or higher priority does not pass by the trigger of the other feature?
yes
no
Blocking of triggers, abort
Continue to check interaction between traces

Figure 5-8: Algorithm for Detecting Blocking of Initial Triggers.

The algorithm for detecting interaction between traces is given in Figure 5-9.

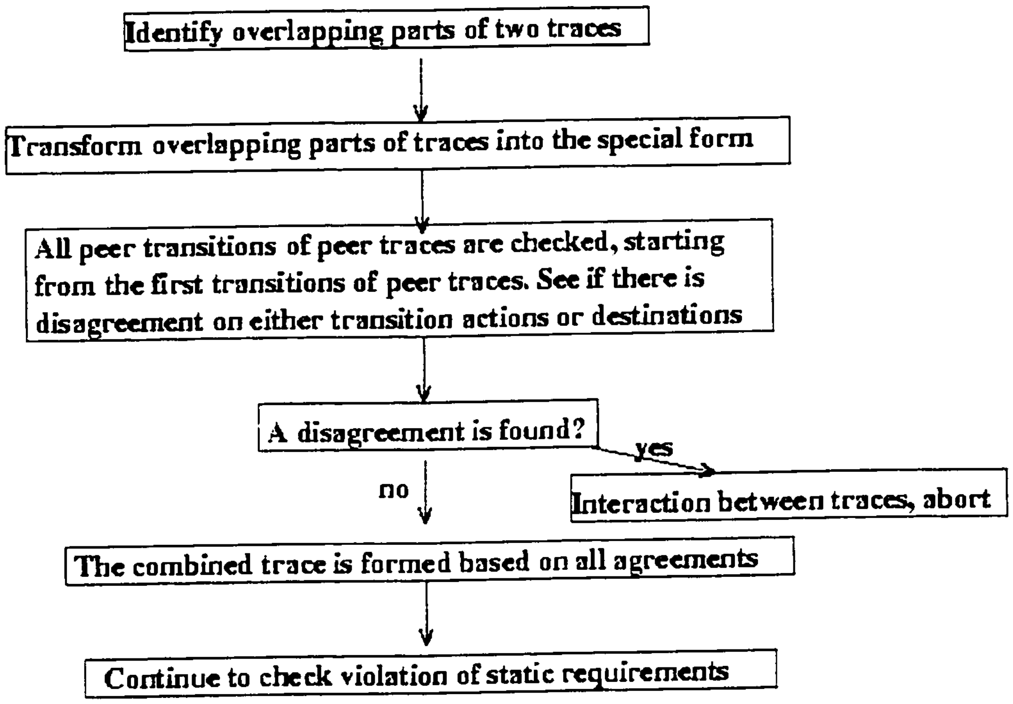

Identify overlapping parts of two traces
Transform overlapping parts of traces into the special form
All peer transitions of peer traces are checked, starting from the first transitions of peer traces. See if there is disagreement on either transition actions or destinations
A disagreement is found?
yes
Interaction between traces, abort
no
The combined trace is formed based on all agreements
Continue to check violation of static requirements

Figure 5-9: Algorithm for Detecting Interaction between Traces.

The labeling procedure for calculating agreements between action sequences is shown in Figure 5-10.

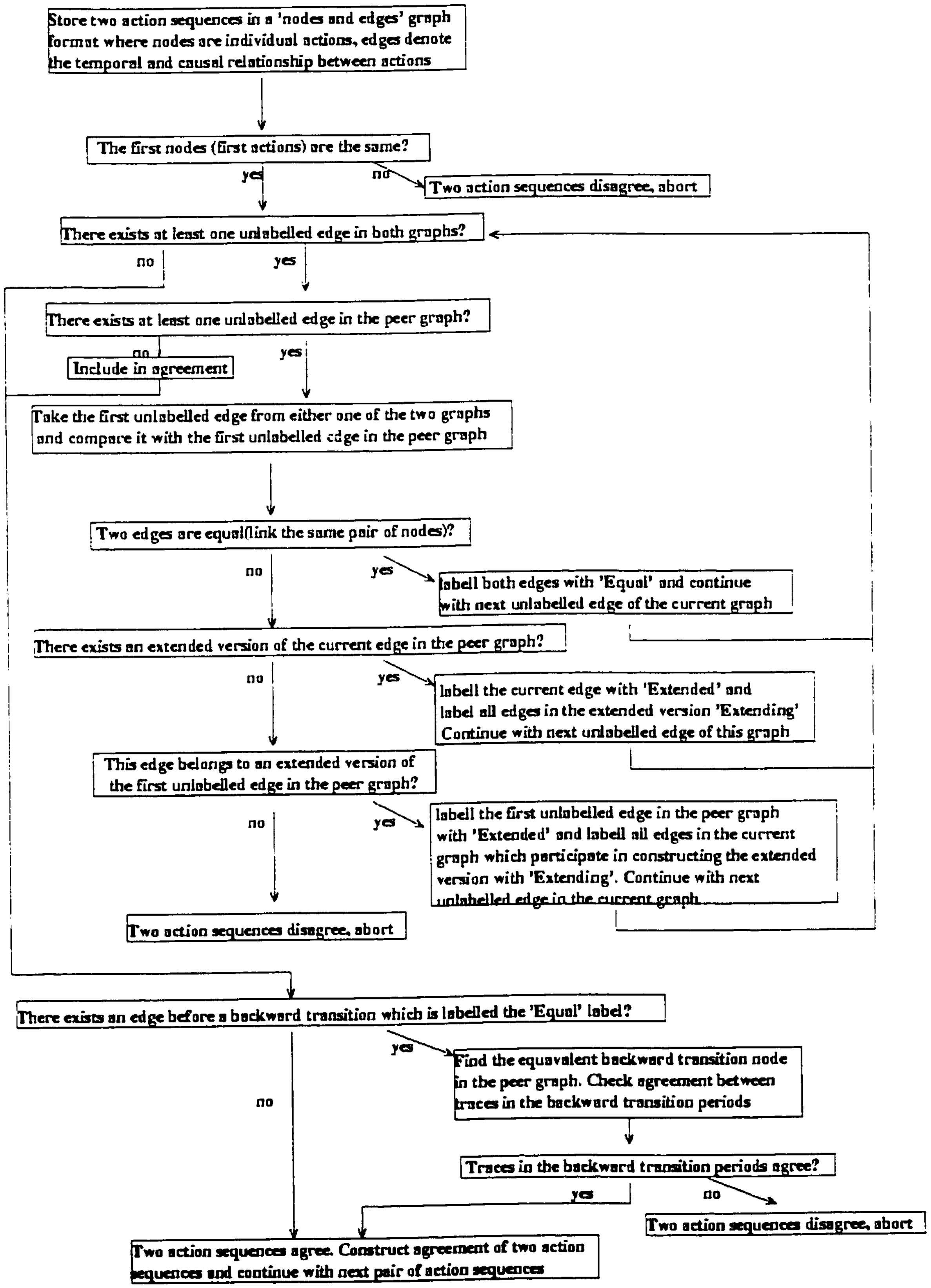

Store two action sequences in a 'nodes and edges' graph format where nodes are individual actions, edges denote the temporal and causal relationship between actions
The first nodes (first actions) are the same?
yes
no
Two action sequences disagree, abort
There exists at least one unlabelled edge in both graphs?
no
yes
There exists at least one unlabelled edge in the peer graph?
no
Include in agreement
yes
Take the first unlabelled edge from either one of the two graphs and compare it with the first unlabelled edge in the peer graph
Two edges are equal(link the same pair of nodes)?
no
yes
labell both edges with 'Equal' and continue with next unlabelled edge of the current graph
There exists an extended version of the current edge in the peer graph?
no
yes
labell the current edge with 'Extended' and label all edges in the extended version 'Extending' Continue with next unlabelled edge of this graph
This edge belongs to an extended version of the first unlabelled edge in the peer graph?
no
yes
labell the first unlabelled edge in the peer graph with 'Extended' and labell all edges in the current graph which participate in constructing the extended version with 'Extending'. Continue with next unlabelled edge in the current graph
Two action sequences disagree, abort
There exists an edge before a backward transition which is labelled the 'Equal' label?
yes
Find the equavalent backward transition node in the peer graph. Check agreement between traces in the backward transition periods
no
Traces in the backward transition periods agree?
yes
no
Two action sequences disagree, abort
Two action sequences agree. Construct agreement of two action sequences and continue with next pair of action sequences

Figure 5-10: Labeling Procedure for Calculating Agreement between Action Sequences.

Finally, the algorithm for detecting violation of static requirements is given in Figure 5-11.

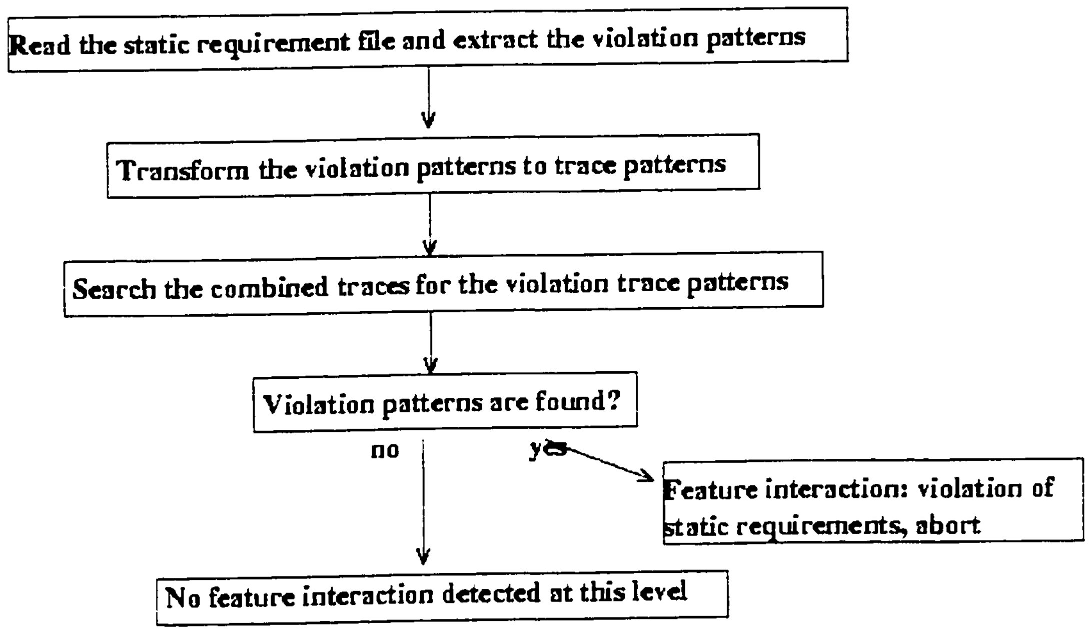

Read the static requirement file and extract the violation patterns
Transform the violation patterns to trace patterns
Search the combined traces for the violation trace patterns
Violation patterns are found?
no
yes
Feature interaction: violation of static requirements, abort
No feature interaction detected at this level

Figure 5-11: Algorithm for Detecting Violation of Static Requirements.

5.8 Comparison with Related Approaches

We believe that our new approach is more complete than Bellcore's approach. We have classified interactions into three types and developed algorithms for each one of them, while Bellcore deals with only one type, comparable only to our second type of interactions, i.e. interaction between traces. Due to the fact that Bellcore did not consider the first type of interactions, their approach encounters many problems. For example, they may fail to detect certain cases of interactions or, in other cases, detect something that is actually not an interaction. They may be able to integrate some cases of the third type of interaction by making a requirement violation a conflicting message. However, this does not apply to all the static requirements. If the specification does not reach the depth of explicitly specifying some internal messages, they still cannot create such a conflict.

Even though the Bellcore algorithm also deals with traces, their algorithm differs significantly from our algorithm for detecting interaction between traces. They use a highly theoretical approach, i.e. the trace equivalence criteria introduced in process algebra, according to which two features do not interact only if the projection of the union of their traces onto their own events results in a set of traces identical to their original sets. Our more practical approach defines agreement between traces as the collective agreements between each pair of transitions. For each pair of transitions, we define agreement as both the agreements between transition destinations and actions. At the lowest level, we define agreements between destinations or actions according to common sense: determination as to whether the destinations are the same and all actions can be acted. Then our algorithm simulates the negotiating process between two traces and outputs the results of the negotiation. There is no complex computation of the set of traces or events. Furthermore, our approach works better within our general framework because it uses the results obtained from detecting the first type of interactions and it generates a combined trace to be used for detecting the third type of interactions.

Besides Bellcore's approach, related approaches do not reach the depth of explicitly specifying the trigger of a feature. This means that these approaches cannot in general detect blocking of trigger type interactions.

In Johan Blom's paper *Formalization of Requirements with Emphasis in Feature Interaction Detection* [28], the author uses temporal logic to specify requirements on a feature. He also uses reachability analysis to search for violation of system requirements. This is similar to our techniques for finding violation of static requirements. However, other aspects of his approach are not comparable to ours because they use logic reasoning, not detailed analysis, to find interactions. In terms of performance, their approach will inevitably encounter the state explosion problem. Therefore, when detecting complex interaction problems, their approach will be slower than our approach.

Chapter 6

Tool and Application

In this chapter, we will first introduce a prototyped implementation of our algorithms as a feature interaction detection tool. Then, we use the feature interaction benchmark developed by Bellcore [5] and present a benchmark of our approach. Finally, we present a summary of the benchmark results.

6.1 Tool Architecture and User Interface

We have developed a prototyped tool to implement the detection algorithms presented in Chapter 5. This tool takes as input MSC feature specification files and requirement specification files produced by the ObjectGeode MSC editor, and gives information as to whether the features interact or not. Figure 6-1 illustrates the architecture of the tool.

In this diagram, there are three inputs to the tool. Each of the two MSC feature files contains MSC feature specifications for one feature. This specification must follow the specification style introduced in Chapter 4. MSC requirement files, containing MSC specifications of violation patterns, comprise the third input. The tool first extracts information from these files and transforms it into internal FSM representation. Then the algorithms are performed one by one, and the results are printed or saved in a file.

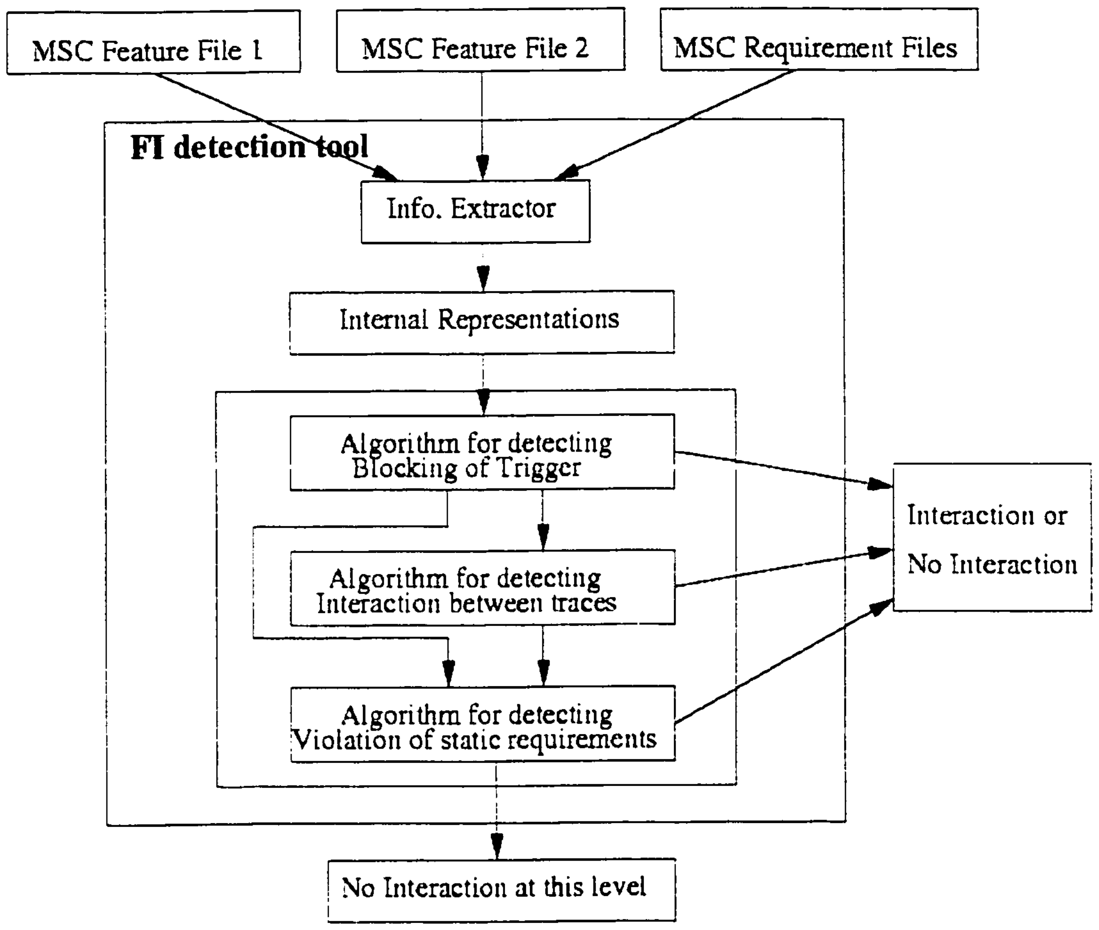
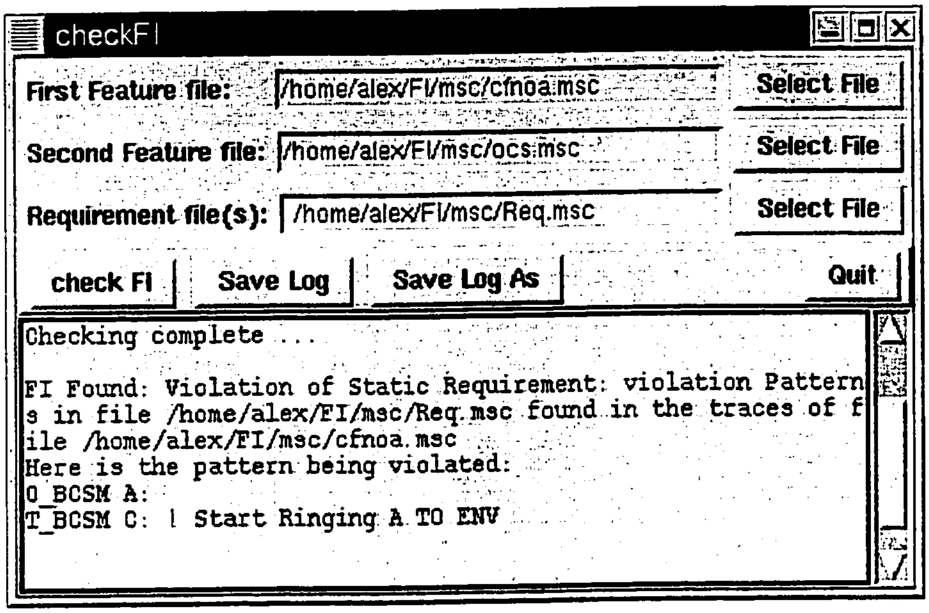

MSC Feature File 1
MSC Feature File 2
MSC Requirement Files
FI detection tool
Info. Extractor
Internal Representations
Algorithm for detecting Blocking of Trigger
Algorithm for detecting Interaction between traces
Algorithm for detecting Violation of static requirements
Interaction or No Interaction
No Interaction at this level

Figure 6-1: Architecture of the FI Detection Tool.

Figure 6-2 illustrates the snap shot of the tool GUI. The top part of the GUI is the part responsible for collecting the parameters (i.e. MSC files) for the tool. Every parameter has an entry field allowing the user to enter the parameter by hand, and a button which invokes a file selection widget if pressed. In the middle of the GUI, there are 4 command buttons. The "check FI" command button causes the tool to check for interaction between the features specified in the MSC files. The result is logged into the text field in the bottom part of the GUI. The "Save Log" and "Save Log As" command buttons save the results in a file whose name the user may specify through a file selection widget. The "quit" command button exits the program.

The GUI is so designed that the user is easily acquainted with the tool. We also introduce some measures to prevent the user from making mistakes. For example, if the user does not specify enough parameters, or if any file specified by the user as a parameter is not found, the GUI will warn the user of this via a message and refuse to launch the tool.



First Feature file: /home/alex/FI/msc/cfnoa.msc

Second Feature file: /home/alex/FI/msc/ocs.msc

Requirement file(s): /home/alex/FI/msc/Req.msc

```
Checking complete ...

FI Found: Violation of Static Requirement: violation Pattern
s in file /home/alex/FI/msc/Req.msc found in the traces of f
ile /home/alex/FI/msc/cfnoa.msc
Here is the pattern being violated:
O_BCSM A:
T_BCSM C: ! Start Ringing A TO ENV
```

Figure 6-2: GUI of the FI Detection Tool.

6.2 Bellcore's Benchmark

We applied the tool to known interaction cases described in the Bellcore benchmark [5]. In the following subsections, we illustrate the benchmark results in detail.

6.2.1 Call Waiting and Answer Call

Problem Description: *A caller attempts to reach a busy line. The Call Waiting feature generates a call-waiting tone to alert the called party. The Answer Call feature connects the calling party to an answering service. In a case where caller **B** is a subscriber to both features, what happens if **B** is already on the line when the second call comes in? Does **B** receive a call-waiting tone or is the second call directed to the answering service?*

Feature Specifications: Based on the call scenario described above, we constructed MSC feature specifications for Call Waiting and Answer Call as shown in Figure 6-3 and Figure 6-4.

Benchmark Results: Below are extracts from the log file saved after running the tool. We can see that the tool identifies the interaction as a blocking of trigger interaction.

```
********** checkFI version 0.1 **********

Feature Files:
/home/alex/FI/MSC/callwaiting.msc
/home/alex/FI/MSC/answercall.msc

Requirement Files:

Running ...

Checking complete ...

FI    Found:    Blocking    of    Trigger:    Trigger    for
"answercall.msc" may be blocked!

***********************************
```

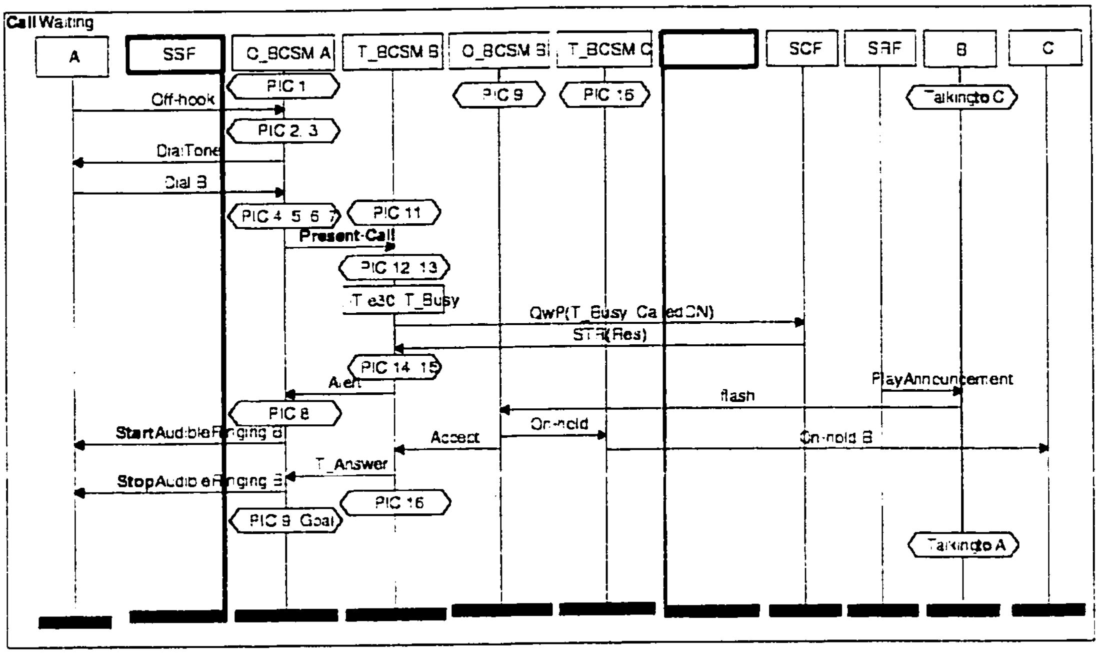


Figure 6-3: Call Waiting.

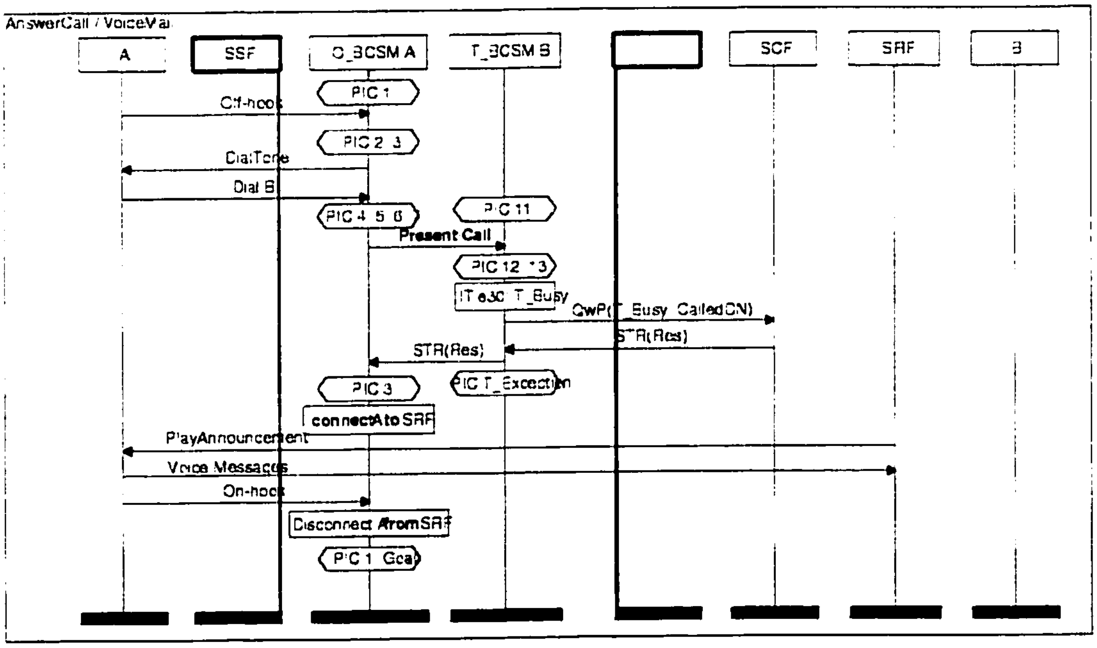


Figure 6-4: Answer Call.

6.2.2 Call Waiting and Three-Way Calling

Problem Description: *During a phone conversation between **A** and **B**, an incoming call from **C** arrives at the switching element for **B**'s line and triggers the Call Waiting feature that **B** subscribes to. However, before being alerted by the call-waiting tone, **B** has flashed the hook, intending to initiate a three-way call. Does the network* *interpret the flash-hook as the response for Call Waiting, or an initiation signal for Three-Way Calling?*

The problem here is that, given the limited signaling capability of customer premise equipment (CPE), the same signal is designed to mean different things, depending on context. In the context of the Three-Way Calling feature, a flash-hook signal (generated by hanging up briefly or depressing a 'tap' button) issued by a busy party signals the network to start adding a third party to the established call. In the context of the Call Waiting feature, the same signal tells the system to accept a connection attempt from a new caller while putting the current conversation on hold.

Feature Specifications: In Chapter 5, we have provided the MSC feature specifications (Figure 5-2 and Figure 5-3) for Call Waiting and Three Way Calling in this scenario.

Benchmark Results: As we have seen in Chapter 5, the interaction between Call Waiting and Three Way Calling in this scenario belongs to the category of interaction between traces. The following is part of the tool's output. We can see the result is in accordance with our analysis in Chapter 5.

```
In the two traces:
O_BCSM B: PIC 9(? flash FROM, ! On-hold TO, ! Accept TO)
T_BCSM C: PIC 16(? On-hold FROM, ! On-hold B TO)

and
O_BCSM  B:  PIC  9(?  flash  FROM,  !  QwP(FeatureReq,
CallingDN) TO, ? TWC_Authorized FROM, ! On-hold TO, !
activate TO)
T_BCSM C: PIC 16(? On-hold FROM, ! On-hold B TO)

FI Found: Interaction Between Traces!
```

6.2.3 "911" and Three-Way Calling

Problem Description: *Suppose that **A** wishes to aid a distressed friend **C** by connecting **C** to a 911 operator (**B**) using the Three-Way Calling service. If **A** and **C** are in conversation and **A** calls 911, **A** can establish the three-way call, since **A** maintains control of putting **C** on hold before calling 911. In the unlikely event, however, that **A** calls 911 first and then tries to bring in **C**, **A** cannot make the three-way call.*

The problem here is that, before bringing a third party into the conversation, a Three-Way Calling subscriber must put the second party on hold. However, the 911 feature prevents users from putting a 911 operator on hold.

Feature Specifications: A basic requirement on 911 services is that one cannot put a 911 operator on hold. This requirement can be specified using MSC as shown in Figure 6-7. Based on the call scenario described above, we also constructed MSC feature specifications for 911 and Three Way Calling as shown in Figure 6-5 and Figure 6-6.

Benchmark Results: The following shows the output of running the tool with the two feature specifications and the requirement specification on 911.

```
FI Found: Violation of Static Requirement: violation
Patterns in file "911Req.msc" found in the traces of file
"TWC_scene1.msc"

Here is the pattern being violated:
O_BCSM A:
T_BCSM B: ! On-hold A TO
```

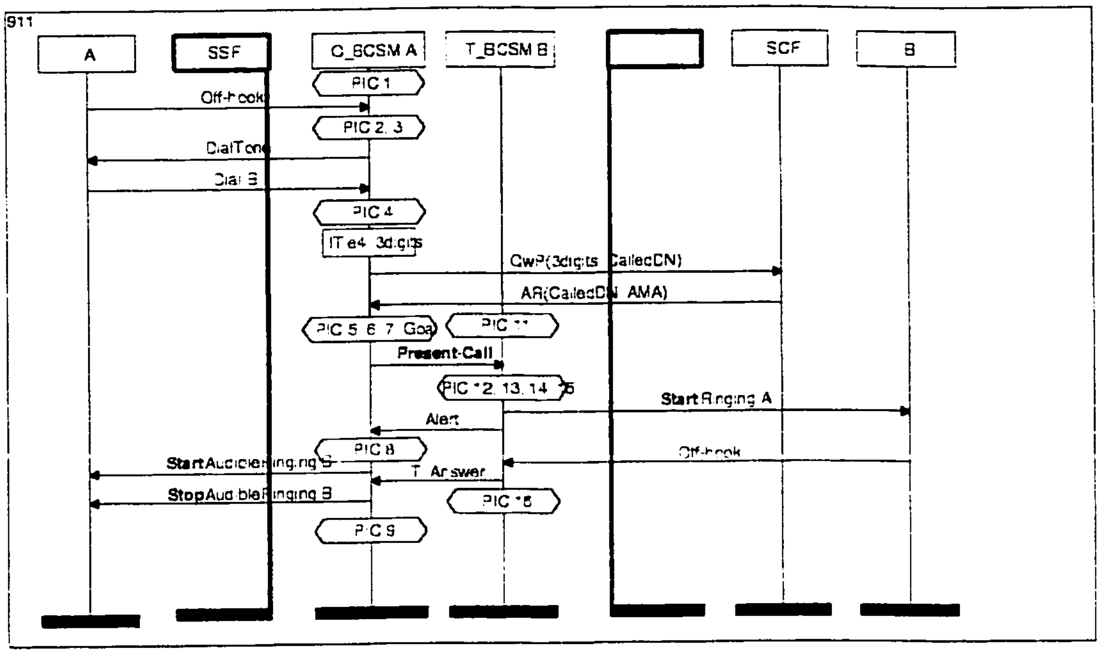


Figure 6-5: "911".

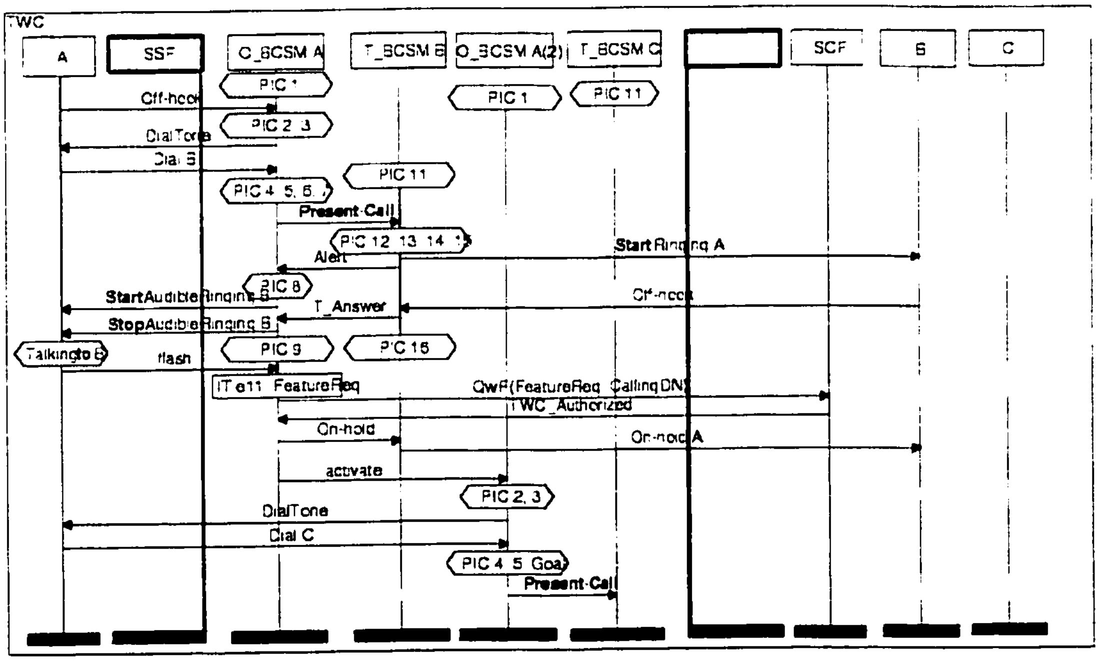


Figure 6-6: Three-Way Calling.

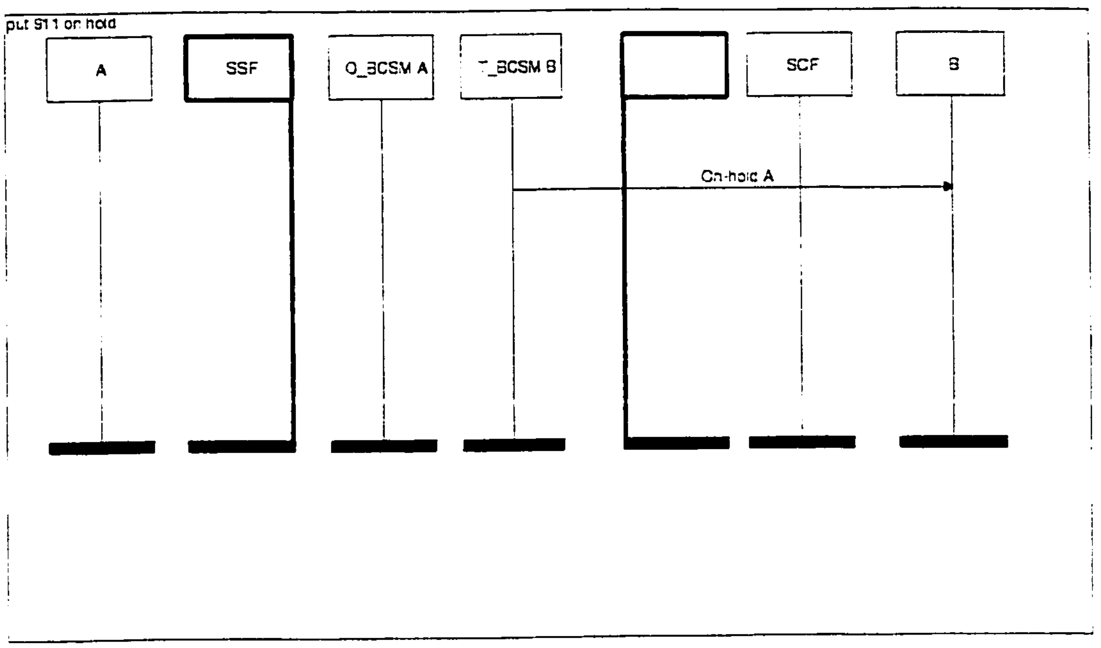


Figure 6-7: Static Requirement on "911".

6.2.4 Terminating Call Screening and Automatic ReCall

Problem Description: *Suppose that* ***B*** *has "no bother" features in response to phone harrassment, and A's number is among those that* ***B*** *refuses to accept via the Terminating Call Screening feature. If A calls* ***B*** *while* ***B****'s line is busy, then when* ***B****'s line becomes idle the Automatic ReCall feature will initiate a connection attempt back to A.*

The problem here is that the Terminating Call Screening feature screens every incoming call against the incoming call screening list. The Automatic ReCall feature automatically returns the last incoming call. If however, the user is on the line when another call comes in, and the Automatic ReCall feature is processed before Terminating Call Screening, Automatic ReCall could register the incoming call number without having the number screened and initiate connection attemps. This would contradict the purpose of the Terminating Call Screening feature.

Feature Specifications: Based on the call scenario described above, we constructed MSC feature specifications for Terminating Call Screening and

Automatic ReCall as shown in Figure 6-8 and Figure 6-9.

Benchmark Results: After careful examination of the specifications of these two features, we can see that the situation described above cannot actually take place, since the TermAtt trigger for Terminating Call Screening is always reached before the trigger T_Busy for Automatic ReCall. This is probably the current implementation of the T_BCSM that avoids this problem. If the implementation is different, for example, if the T_BCSM checks the busy condition before termination attempt, then the situation could actually arise. When we run the tool to analyse the current configuration, it will give us output indicating that there is a blocking of trigger type interaction. However, this is a desired interaction. Below is part of the output from the tool

```
FI Found: Blocking of Trigger: Trigger for "AutomaticReCall.msc" may be blocked!
```

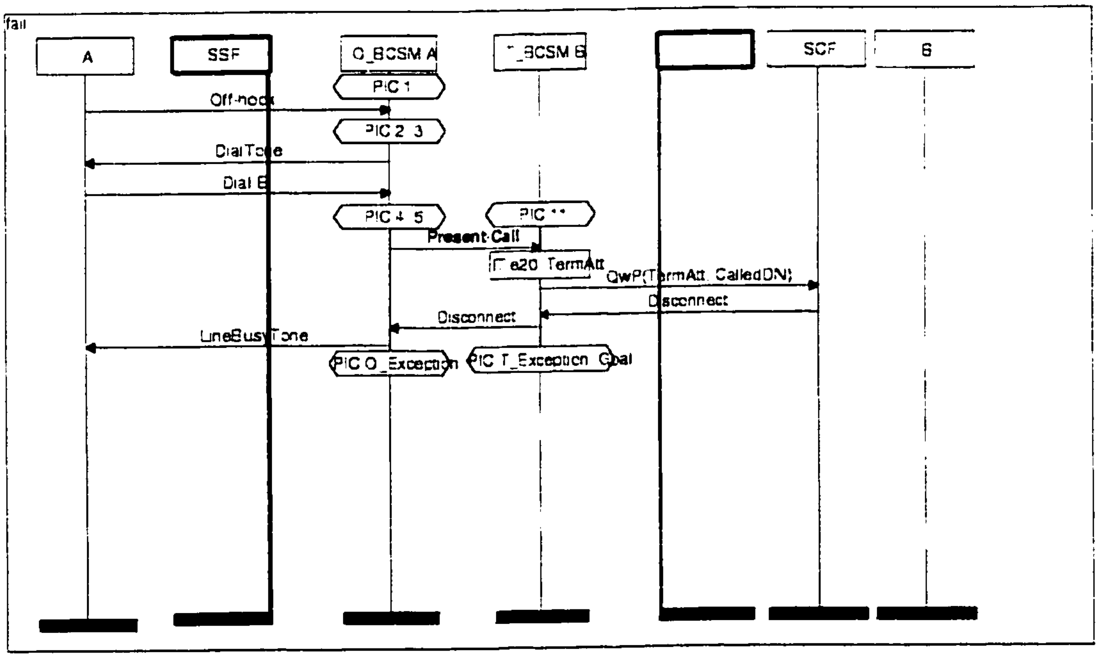


Figure 6-8: Terminating Call Screening.

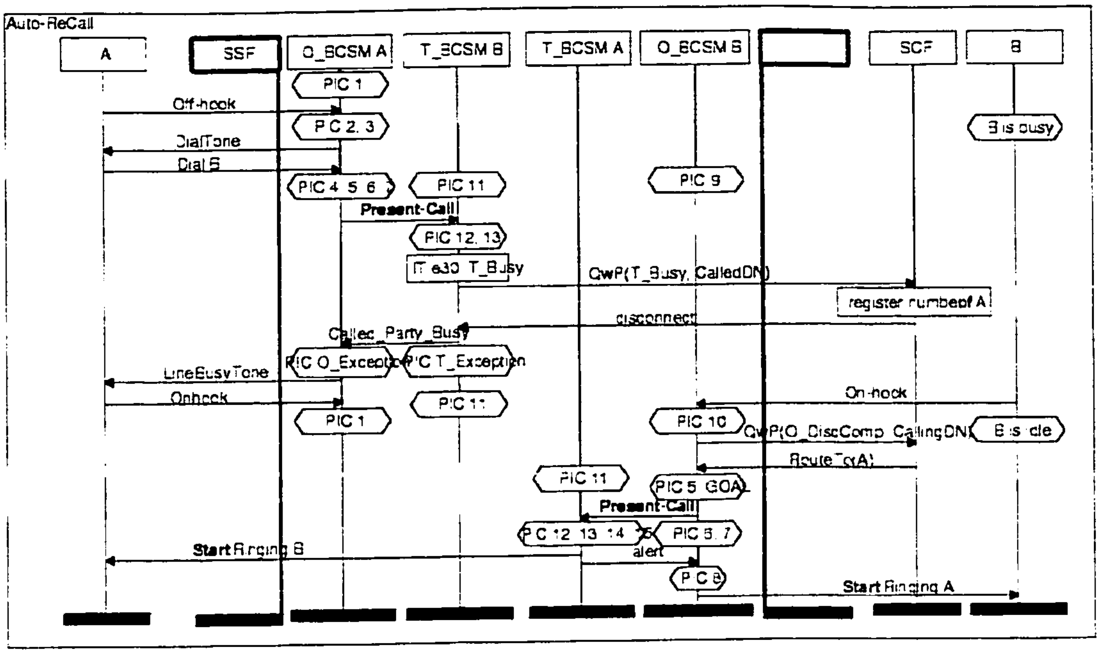


Figure 6-9: Automatic ReCall.

6.2.5 Originating Call Screening and Area Number Calling

Problem Description: *A, who is subscriber of the AIN Originating Call Screening feature, calls Domino Pizza's area number to order a pizza. Domino's Pizza pays for Area Number Calling feature to serve its customers by directing calls to the nearest location. The call cannot go through.*

The problem here is that the Originating Call Screening feature aborts attempts to connect the subscriber to directory numbers in the screening list, whereas Area Number Calling decides the actual terminating number based upon the originating number and the dialed number. Each of them needs to launch a query for call origination treatment during call set up. Switching elements may impose restrictions on the number of queries per call and, after automatically launching a query for the Originating Call Screening feature on behalf of the caller, the component will not be able to launch another query for the Area Number Calling feature.

Feature Specifications: In Chapter 4, we have constructed the MSC specification for Originating Call Screening (Figure 4-3). In Figure 6-10, we present the MSC specification for the Area Number Calling Feature.

Benchmark Results: After running the tool on the specifications, we got the message shown below, as expected.

```
FI   Found:   Blocking   of   Trigger:   Trigger   for
"AreaNumberCalling.msc" may be blocked!
```

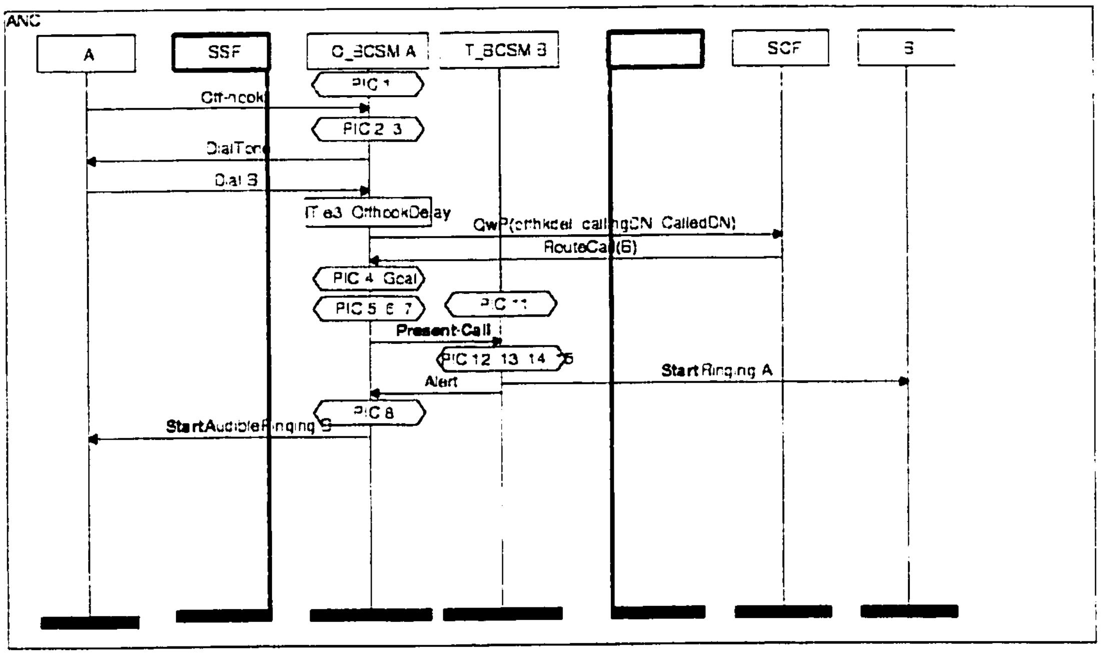


Figure 6-10: Area Number Calling.

6.2.6 Operator Services and Originating Call Screening

Problem Description: *A has subscribed to the Originating Call Screening feature deployed in a local switching element, hoping to screen outgoing calls made to any number in a screening list. However, anybody can make an operator assisted 0+ or 0- call to a screened number using* ***A****'s line.*

The problem here is that Operator Services may be handled in a remote switching element that does not have access to the feature subscription profile of every customer who wishes to use the services. Therefore, every call made through Operator Services acts like an outgoing POTS call, except that it is operator-assisted.

Feature Specifications: In this example, we need to use the static requirement on OCS presented in Figure 4-4. Figure 6-11 illustrates the MSC specification of the IN Operator service feature. We can still use Figure 4-3 as the specification for OCS.

Benchmark Results: As shown below, the tool detected the violation of static requirement on OCS.

```
FI Found: Violation of Static Requirement: violation
Patterns in file "Req.msc" found in the traces of file
"OperatorService.msc"
Here is the pattern being violated:
O_BCSM A:
T_BCSM C: ! Start Ringing A TO
```

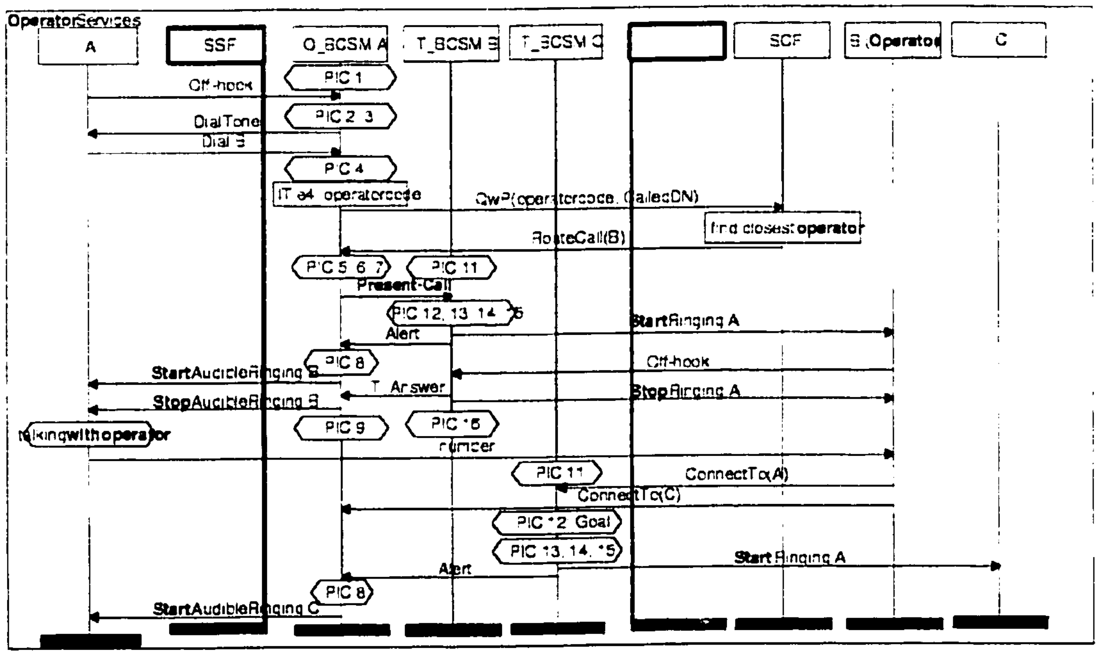


Figure 6-11: Operator Services.

6.2.7 Credit-Card Calling and Voice-Mail Service

Problem Description: *A is a credit-card customer who frequently calls Aspen and is familiar with Aspen calling procedures. A places a credit-card call to Aspen and hits "#" immediately to access his Aspen Voice-Mail without waiting for Aspen's* *introductory prompt. However, the "#" signal is intercepted by the credit-card call feature and interpreted as an attempt to make a second call.*

The problem here is that, for the convenience of customers, many credit-card calling services instruct callers to press "#" to place another credit-card call, instead of requiring them to hang up and dial the long distance access code again. But to access Voice-Mail messages from phones other than his/her own, some Voice-Mail services such as Aspen allow users to (1) dial the Aspen service number, (2) listen to introductory prompt (instruction), (3) press "#" followed by the mailbox number and passcode to indicate that the caller is a subscriber, and then (4) proceed to check their messages. When a customer places a credit-card call to Aspen, the customer does not know exactly at what point the Credit-Card Calling feature stops interpreting signals and starts passing them to a called party.

Feature Specifications: Figure 6-12 and Figure 6-13 show the MSC specifications for the Credit Card Calling feature and Voice Mail Service features in the scenario described above.

Benchmark Results: As shown below, the tool identifies a blocking of trigger type interaction, which is exactly what we would expect.

```
FI Found: Blocking of Trigger: Trigger for
"VoiceMail.msc" may be blocked!
```

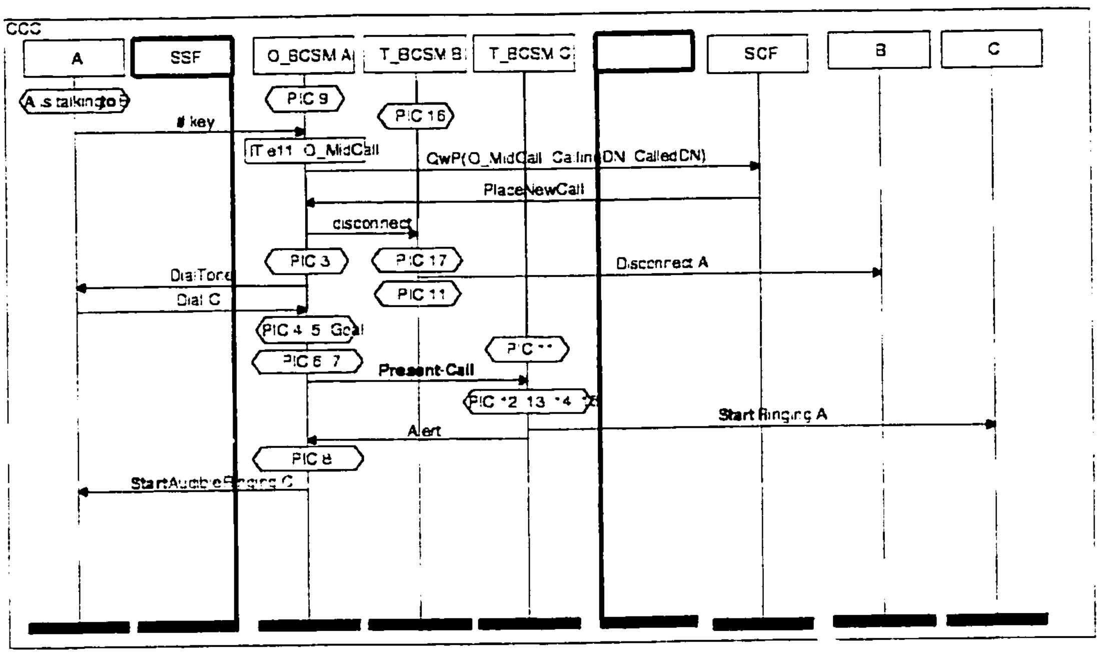


Figure 6-12: Credit Card Calling.

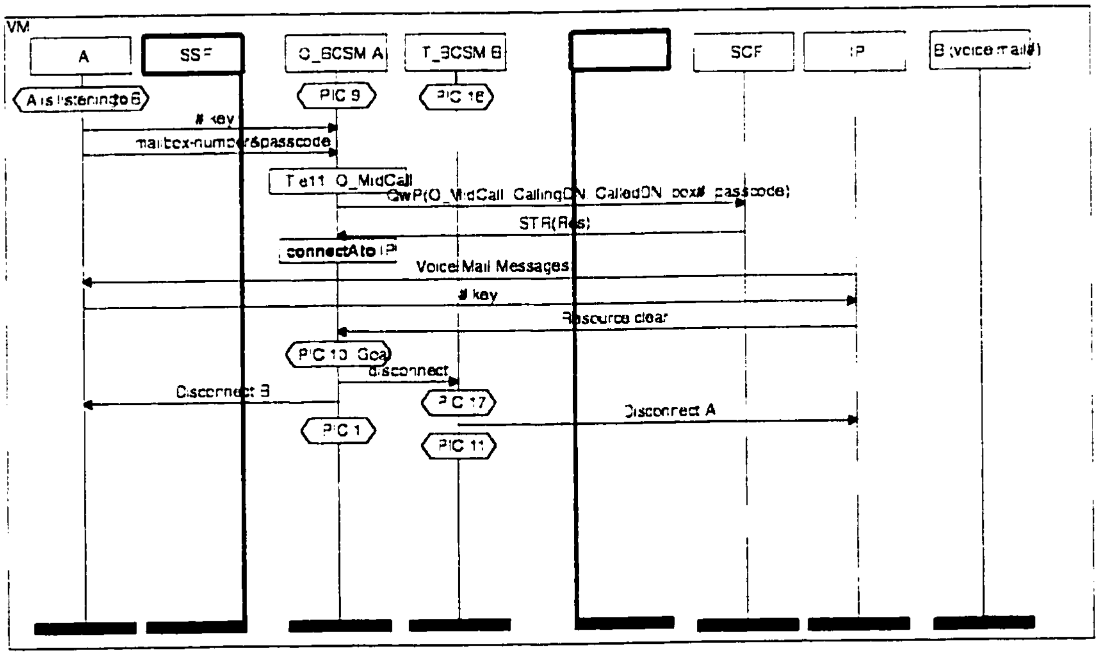


Figure 6-13: Voice Mail.

6.2.8 MBS-ED and CENTREX

Problem Description: *The Bellcore Benchmark poses the following problem:*

"The AIN Release 0 Multi-location Business Service-Extension Dialing (MBS-ED) feature allows a customer to extend a 4 or 5 digit extension dialing plan to

locations served by different switching elements. This is accomplished by querying an SCP or an adjunct to translate digit combinations to directory numbers. CENTREX features are also based upon a 4-digit extension dialing plan, but are served by a single switch. Thus, CENTREX features do not query an SCP or adjunct, as the corresponding directory numbers must be local to that swich. When a 4-digit number is received in a switch that supports both AIN Release 0 MBS-ED and CENTREX features, it is not clear whether an SCP should be queried or not. Assignment of disjoint subsets of the 4-digit numbers to the MBS-ED features and the CENTREX features could be used by the switch to detect what type of features in effect whenever a 4-digit number is received."

Feature Specifications: Figure 6-14 and Figure 6-15 show the MSC specifications of the MBS-ED and CENTREX features in the scenario described above.

Benchmark Results: After running the tool with the two MSC files as input, the following information is output from the tool:

```
FI Found: Blocking of Trigger: Trigger for
"CENTREX.msc" may be blocked!
```

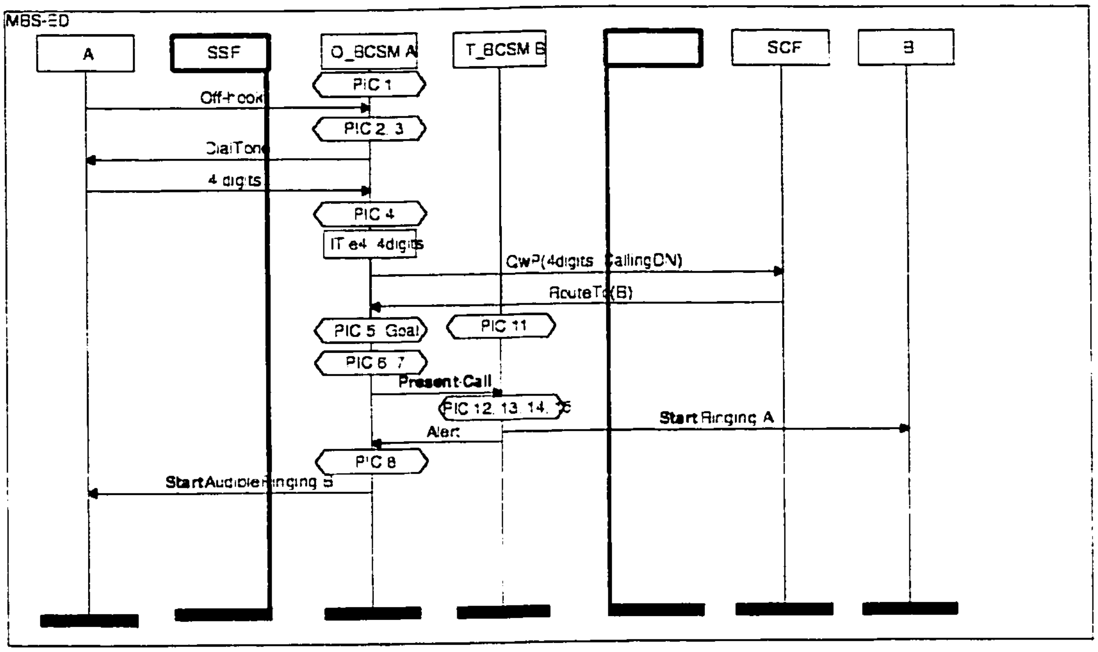


Figure 6-14: MBS-ED.

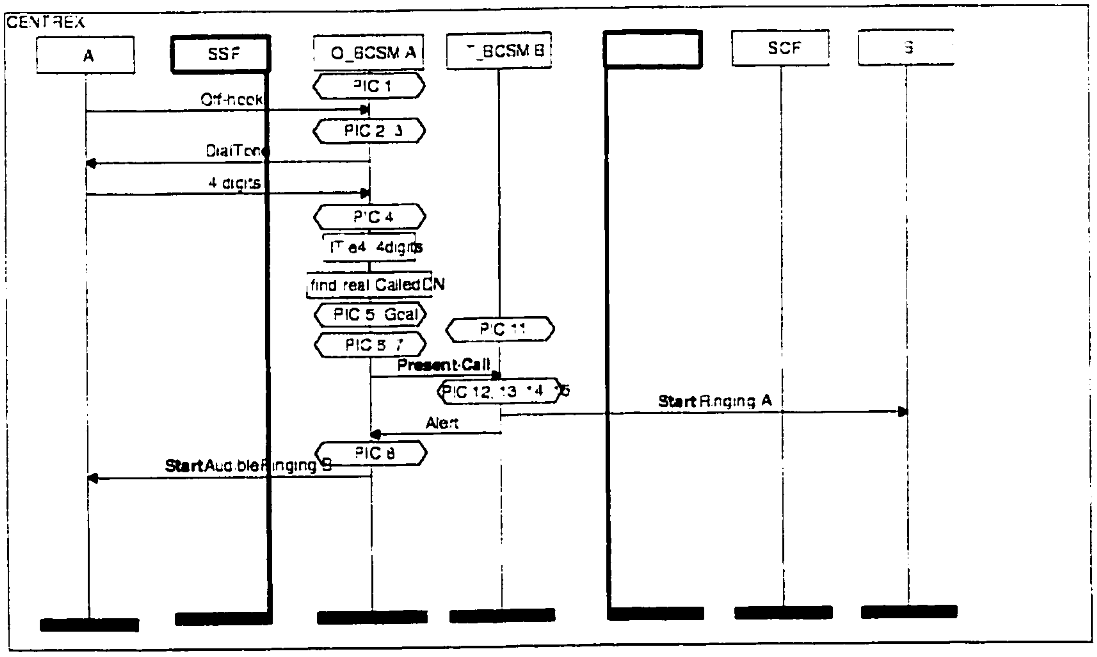


Figure 6-15: CENTREX.

6.2.9 Call Forwarding and Originating Call Screening

Problem Description: *An adolescent knows that his parents have used Originating Call Screening to block all calls to a dial-porno number* ***C*** *from their home line* ***A****. Using the Call Forwarding feature, he instructs the switching element to forward all* *calls terminated at **A** to **C**, and then call himself to get the effect of Call Forwarding. This seeming loophole was not anticipated by his parents.*

The problem is that Call Forwarding allows incoming calls to be redirected to another directory number, while Originating Call Screening aborts attempts to connect the subscriber to some other directory number and then forward calls to the same number, but the above situation can arise if the forwarding number and the call screening number were supplied by two different people sharing the same physical line. Whether the forwarding number is considered a dialed number (to be checked against the screening list) becomes an issue. If Call Forwarding takes precedence over Originating Call Screening, calls can be forwarded to the forwarding number despite the fact that the number is also on the screening list.

Feature Specifications: In this example, the scenario for Originating Call Screening is that a caller calls him/herself. Therefore, we can no longer use the specification presented in Chapter 5. The new specification for OCS is presented in Figure 6-16. The specification for Call Forwarding in this scenario is presented in Figure 6-17. The static requirement for OCS is shown in Figure 4-4.

Benchmark Results: After running the tool with the two feature specification files together with the requirement specification file for OCS, the output shows a violation of static requirement type interaction, exactly as expected.

```
FI Found: Violation of Static Requirement: violation
Patterns in file "Req.msc" found in the traces of file
"cfu_self.msc"

Here is the pattern being violated:
O_BCSM A:
T_BCSM C: ! Start Ringing A TO
```

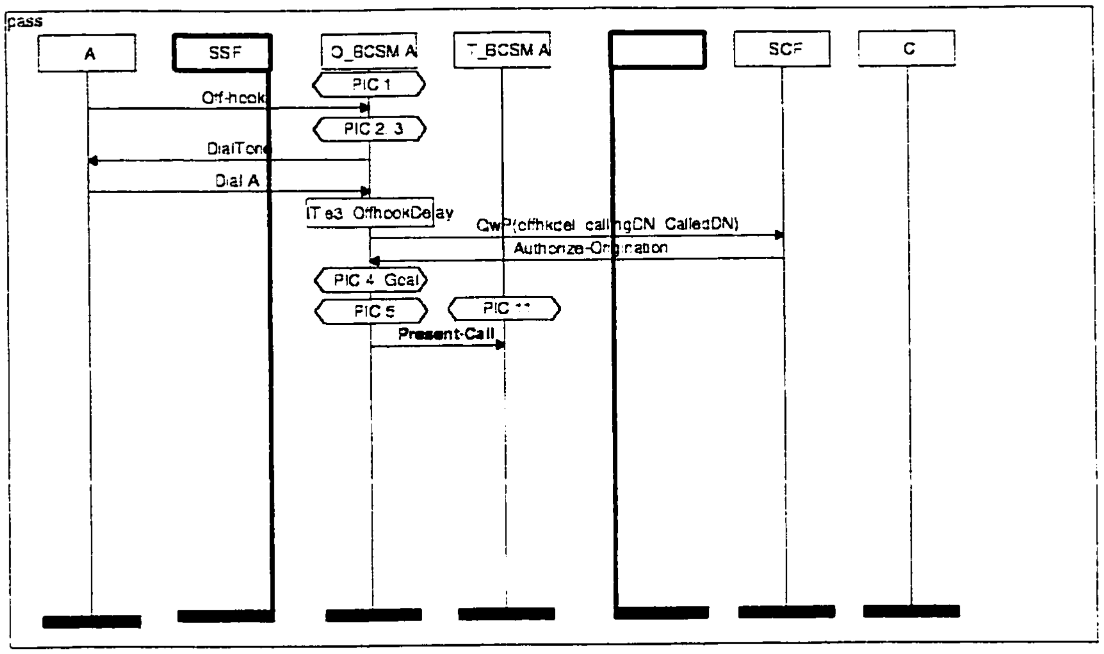


Figure 6-16: OCS, Calling to Oneself.

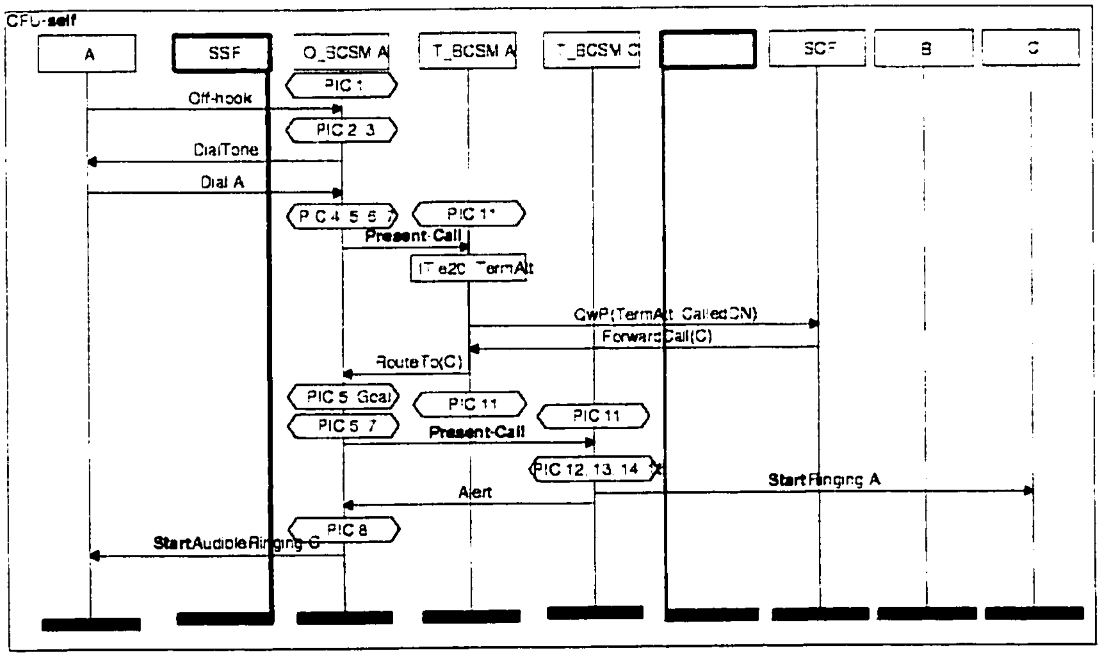


Figure 6-17: Call Forwarding, Calling to Oneself.

6.2.10 Call Waiting and Personal Communication Services (PCS)

Problem Description: *X and Y are both PCS customers currently registered with the same CPE; X has Call Waiting but Y does not. Y is on the phone when somebody calls X. Since X has Call Waiting and is registered on the line, the new call triggers* *the Call Waiting feature of **X**. But is it legitimate to send the call-waiting alert through the line to interrupt **Y**'s call? If not, then **X**'s Call Waiting feature is ignored. The problem here is that Call Waiting is a feature assigned to a directory number, but Call Waiting uses the status of the line with which the number is associated to determine whether the feature should be activated and PCS feature assigns a directory number to the line which may or may not have Call Waiting associated.*

Benchmark Results: In our specification style, since we are not concerned with the physical location of network elements, we are unable to model the Personal Communications Feature, which attaches a directory number to any physical line. Therefore, we were unable to detect this interaction. Future improvements on the specification style are needed.

6.2.11 OCS and MDNL-DR

Problem Description: ***B*** *is a subscriber of the MDNL-DR service with two numbers* ***X*** *and* ***Y****, and* ***A*** *has the Originating Call Screening service with the number* ***X*** *in the outgoing call screening list.* ***A*** *can still make calls to* ***B****'s line, if* ***A*** *dials* ***Y****.*

The problem here is that Originating Call Screening is a feature based on directory numbers and any call placed to a directory number on the screened list will be blocked. Disallowing calls to a directory number prevents connections to the identified line, provided the line is associated with only that directory number. However, services for Multiple Directory Number Line with Distinctive Ringing (MDNL-DR) allow more than one directory number to be associated with a single line.

Benchmark Results: Same as PCS, we are currently not able to model MDNL-DR with our specification style. Therefore, we are unable to detect this interaction.

6.2.12 OCS and Call Forwarding (revisited)

Problem Description: *A has C on the Outgoing Call Screening list, but, when A calls B, B, who has Call forwarding, will forward all incoming calls to the number C, and connections from A to the line identified as C will be established.*

The problem here is that Originating Call Screening blocks calls based on the number dialed; thus, calls to a particular line are blocked only if the dialed number is associated with that line. However, Call Forwarding connects to a line other than the one associated with the dialed number.

Feature Specifications: We use Figure 4-3 as the specification for OCS in this example. Figure 6-18 shows the Call Forwarding feature in this scenario.

Benchmark Results: The nature of this interaction is exactly the same as the interaction between OCS and CFNoA which we discussed in Chapter 5. After launching the tool with OCS. Call Forwarding and the requirement specification of OCS as input. we obtain the output as shown below.

```
FI Found: Violation of Static Requirement: violation
Patterns in file "Req.msc" found in the traces of file
"cfu.msc"

Here is the pattern being violated:
O_BCSM A:
T_BCSM C: ! Start Ringing A TO
```

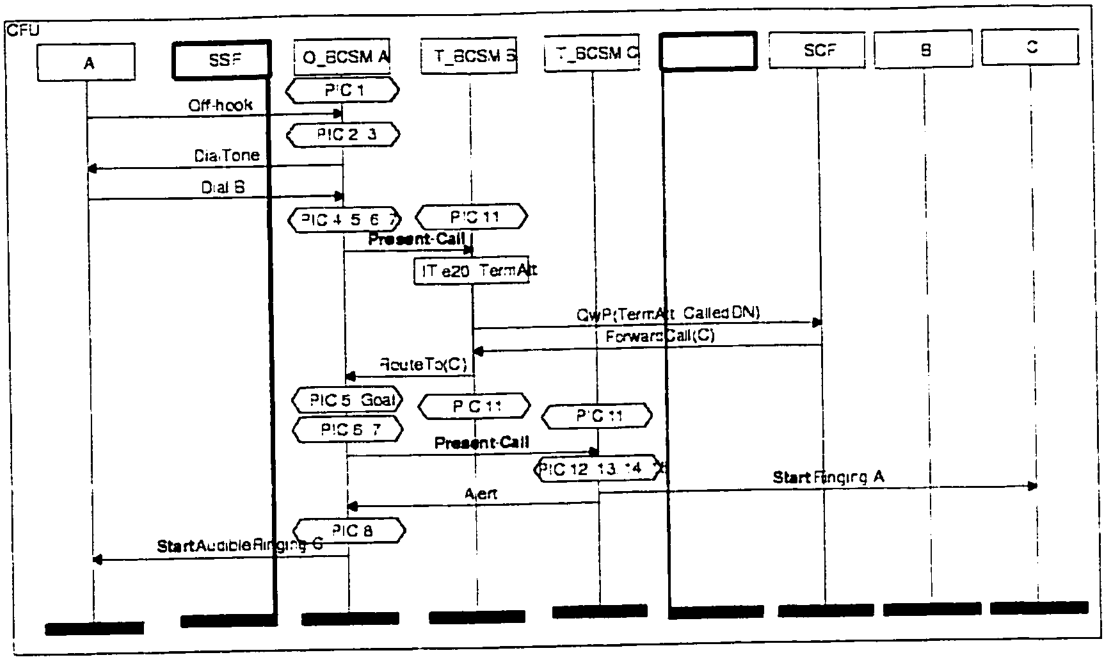


Figure 6-18: Call Forwarding.

6.2.13 Call Waiting and Automatic CallBack

Problem Description*: A calls B. B is a Call Waiting subscriber and A has activated the Automatic CallBack feature. Will Automatic CallBack work for A, if B is calling to C when A calls B? Note that because of the Call Waiting feature that B has, A will receive a ring-back signal from B, instead of the busy signal.*

The problem here is that Automatic CallBack is triggered if the called party is busy, but a line with Call Waiting appears to be idle to a caller, although it is actually busy.

Feature Specifications: In Figure 6-3, we already have the specification for the Call Waiting feature. In Figure 6-19, we present the specification for Automatic CallBack.

Benchmark Results: After launching the tool, the output indicates that the trigger O_Called_Party_Busy of Automatic CallBack is blocked. This is in accordance with the problem described above.

```
FI Found: Blocking of Trigger: Trigger for
"ACB_part1.msc" may be blocked!
```

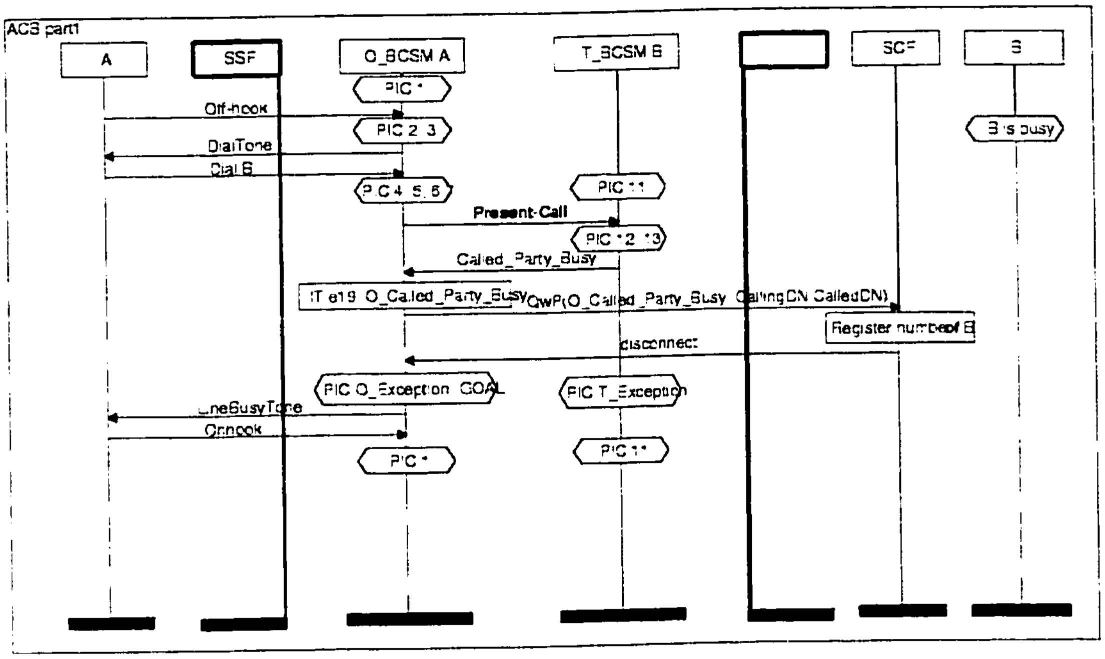


Figure 6-19: Automatic CallBack. Part 1.

6.2.14 Call Waiting and Call Waiting

Problem Description: *The Bellcore Benchmark describes an interesting conflict situation that often comes up:*

*Subscriber **A** calls subscriber **B**. Both have Call Waiting. **A** now puts **B** on hold to talk to **C**. While on hold, **B** decides to flash the hook to answer an incoming call from **D**, which puts **A** on hold as well. If **A** then flashes the hook expecting to get back to the conversation with **B**, **A** will be on hold instead, unless either **B** also flashes the hook to return to a conversation with **A** or **D** hangs up automatically returning **B** to a conversation with **A**.*

*Ambiguity arises when **B**, instead of flashing to return to conversation with **A**, actually hangs up on the conversation with **D** while **A** is still talking to **C**. There are two separate contexts in which to interpret **B**'s action. Assume that CW1 refers to the Call Waiting call among **C-A-B** and CW2 refers to the one among **A-B-D**. According to the specification of Call Waiting, in the context of CW2 **B** will be rung back (because **A** is still on hold) and, upon answering, become the held party in the CW1* *context and hear nothing. But, in the context of CW1, **B**'s call termination will be interpreted as simply a disconnection: **A** and **C** will be placed in a normal two-way conversation and **B** will be idled. The question is: Should **B** be rung back or should **B** be idled?*

The problem here is that Call Waiting allows a subscriber to put the other party on hold. However, it does not protect the subscriber from being put on hold. Confusion can arise when two parties exercise this type of control concurrently.

Feature Specifications: We model the two Call Waiting features for CW1 and CW2 with the two MSC specifications Shown in Figure 6-20 and Figure 6-21.

Benchmark Results: After launching the tool with CW1 and CW2, we detect an interaction between traces. The following shows the extracts from the output:

```
In the two traces:
O_BCSM A: PIC 9(? flash FROM, ! On-hold TO, ! Accept TO,
? Disconnect FROM, ! held_party disconnected TO) -> PIC
10(-> PIC 1)
T_BCSM B: PIC 16(? On-hold FROM, ! On-hold A TO, ? On-
hook FROM) -> PIC 17(! Disconnect TO, ->PIC 11)


and
O_BCSM A: PIC 9(? On-hold FROM, ! On-hold B TO)
T_BCSM B: PIC 16(? flash FROM, ! On-hold TO, ? get-back-
to held_party FROM, -> PIC 15, ! Start Ringing A TO)


FI Found: Interaction Between Traces!
```

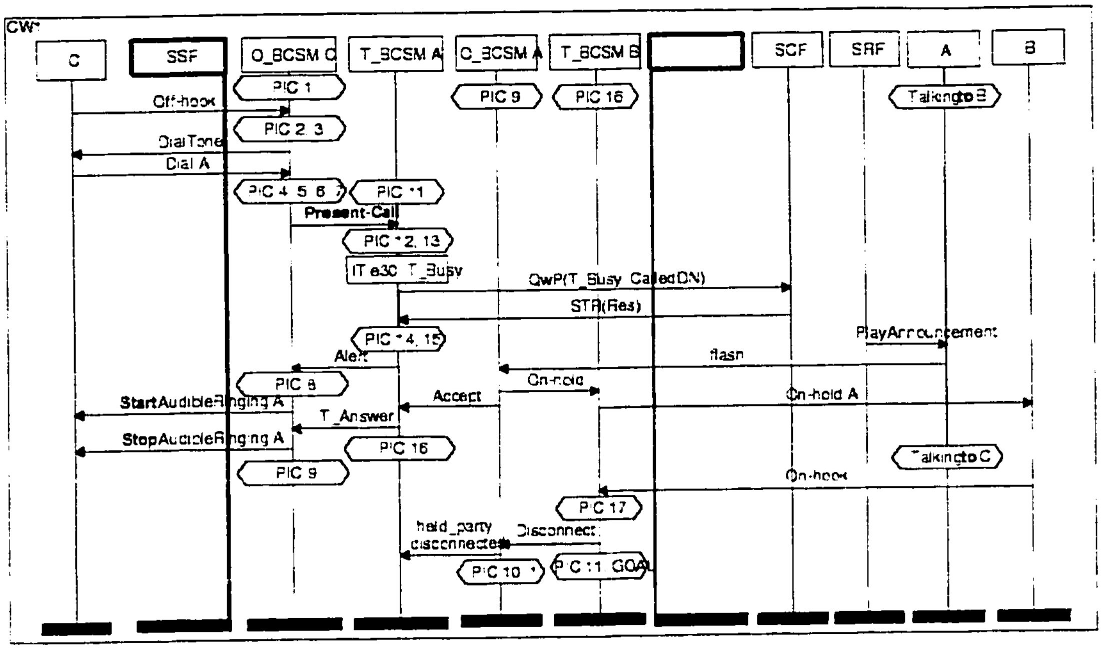


Figure 6-20: Call Waiting CW1.

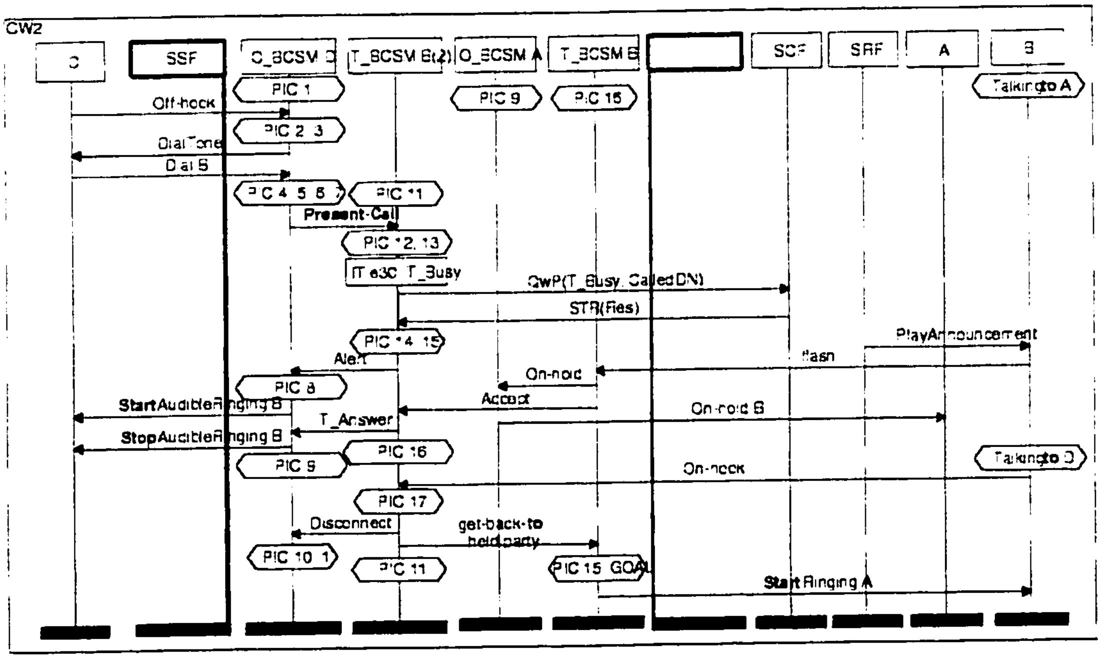


Figure 6-21: Call Waiting CW2.

6.2.15 Call Waiting and Three-Way Calling (revisited)

Problem Description: *A has both Call Waiting and Three-Way Calling.* ***B*** *has Call Waiting.* ***A*** *puts* ***B*** *on hold and talks to* ***C***. ***A*** *decides to have* ***B*** *join his conversation with* ***C***, *so he puts* ***C*** *on hold, makes a* *second* *call to* ***B***. ***B*** *hears the Call Waiting signal and answers the call using the Call Waiting feature.* ***A*** *brings* ***C*** *back into the* *conversation to establish a three-way call.*

There are now three contexts in this establishment: a Call Waiting call and a Three-Way Calling call, both established by ***A*** *among* ***B-A-C****, and a Call Waiting call established by* ***B*** *as* ***A-B-A****.*

Now, ***B*** *hangs up. According to the contexts established by* ***A****, the session becomes a two-way call between* ***A*** *and* ***C****; according to the contexts established by* ***B****, however,* ***B*** *should get a ring-back because* ***B*** *still has* ***A*** *on hold.*

The problem here is that a user can exercise both features simultaneously and the call control relationship is now quite complicated.

Benchmark Results: In this example, there are three features involved in the interaction. The Call Waiting and Three-Way Calling features of A and the Call Waiting feature of B. The three features form one scenario and separating any feature from them will make the scenario different. Our current detection methods and algorithms cannot handle interactions between three features. Since the scenario is not separable, we cannot check interaction between one feature and the combination of the other two features either. Therefore, we cannot detect this interaction.

6.2.16 Calling Number Delivery and Unlisted Number

Problem Description**:** *Customer* ***A*** *with an Unlisted Number places a call to another customer* ***B*** *with Calling Number Delivery. If the network allows* ***A****'s number to be delivered to* ***B****, then* ***A*** *loses privacy; if it doesn't, then* ***B*** *gets no information. Either way, one of the features doesn't perform its intended function. Currently, delivery of the number can be blocked by the Calling Number Delivery Blocking call-processing feature, which nullifies the Calling Number Delivery feature by delivering a number consisting of only 1's.*

The problem here is that Calling Number Delivery is a call-processing feature that delivers the directory number of the calling party to the customer's premises during the ringing cycle and assumes that information such as the subscriber's number will be released. Unlisted Number, on the other hand, is a directory-service *feature designed to allow a subscriber to keep the number private.*

Feature Specifications: In Figure 6-22, we provide the MSC specification of the Calling Number Delivery Feature. The Unlisted Number feature does not have much dynamic behavior by itself. Therefore, we do not need to model Unlisted number using MSC. In Figure 6-23, we present the static requirement on Unlisted Number, i.e. The sending of the calling number is prohibited. This is shown in Figure 6-23 by a violation pattern, that is the CallingDN(A) message which violates the static requirement.

Benchmark Results: As we would expect, the static requirement on Unlisted Number will be violated by Calling Number Delivery. Note, here we include only one feature file in the input. Here is the results obtained after running the tool:

```
FI Found: Violation of Static Requirement: violation
Patterns in file "UNReq.msc" found in the traces of file
"CND.msc"

Here is the pattern being violated:
O_BCSM A: OUT CallingDN(A) TO
T_BCSM B: IN CallingDN(A) FROM
```

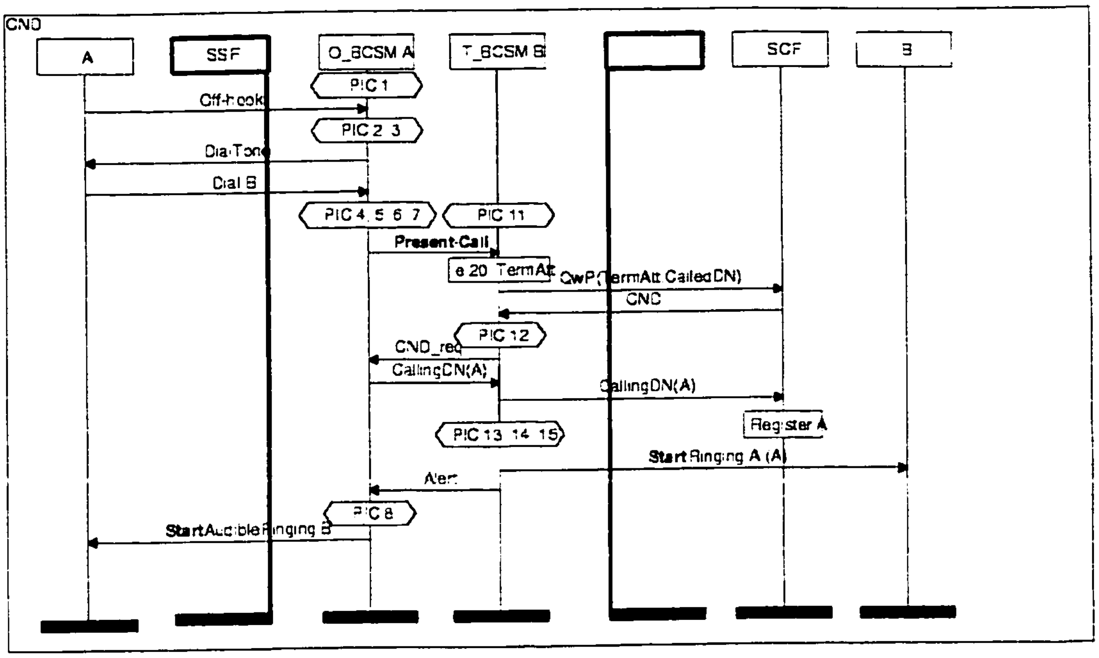


Figure 6-22: Calling Number Delivery.

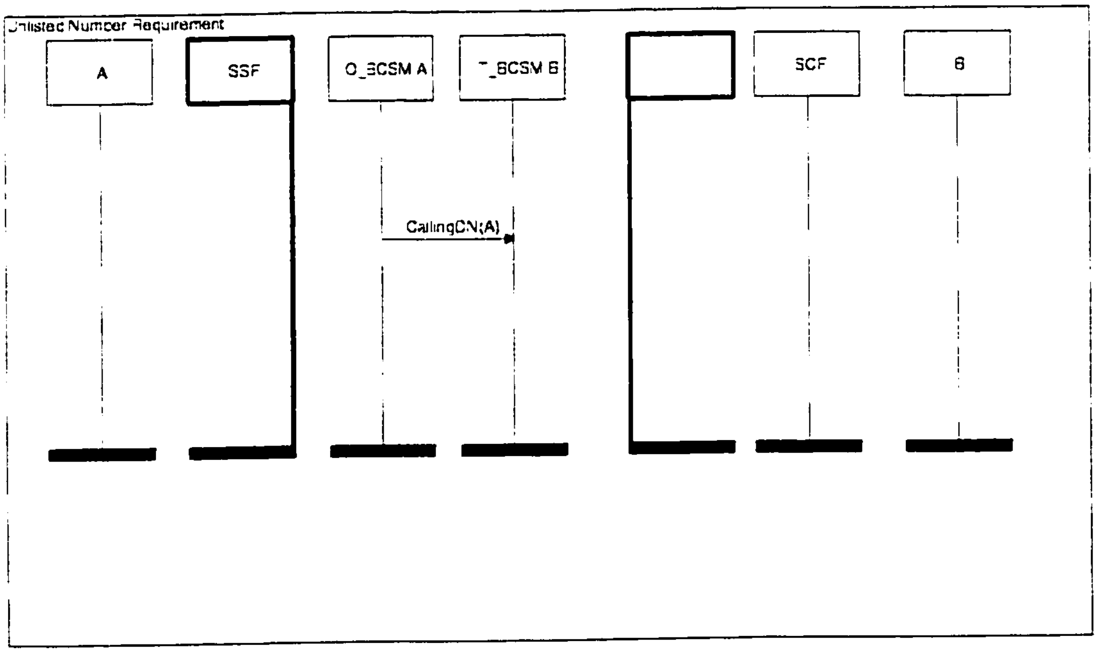


Figure 6-23: Requirement on Unlisted Number.

6.2.17 Call Forwarding and Call Forwarding

Problem Description: *Customer A with Call Forwarding redirects calls to his/her own number and creates an infinite loop. Or calls for A are forwarded to **B**, only to be forwarded back to A by **B** (i.e., a two-number loop); or to **B**, then to **C**, then back to **A** (i.e., a three-number loop); and so on.*

Detecting the loop during call processing is difficult if the amount of information (e.g., the numbers having been reached) passed is limited. Currently, SS7 uses a counter, aborting calls that are forwarded more than 5 times.

Feature Specifications: In this example, instead of modeling two fowarding features in two MSCs, it seems more natural to model the combined behavior in one MSC (Figure 6-24). The loop can be detected using a static requirement, i.e., if a call is presented twice to one number, then a loop exists. Figure 6-25 presents an MSC specification of such a static requirement.

Benchmark Results: As we might expect, in Figure 6-24, the call is presented twice to B and a loop is successfully detected. Note that, no matter how many parties are involved in forming the loop, we are able to detect the loop with the static requirement presented in Figure 6-25. Here are the results obtained after running the tool with the feature file and the static requirement file.

```
FI Found: Violation of Static Requirement: violation
Patterns in file "UNReq.msc" found in the traces of file
"CND.msc"

Here is the pattern being violated:
O_BCSM A: OUT CalledDN(B) TO. OUT CalledDN(B) TO
```

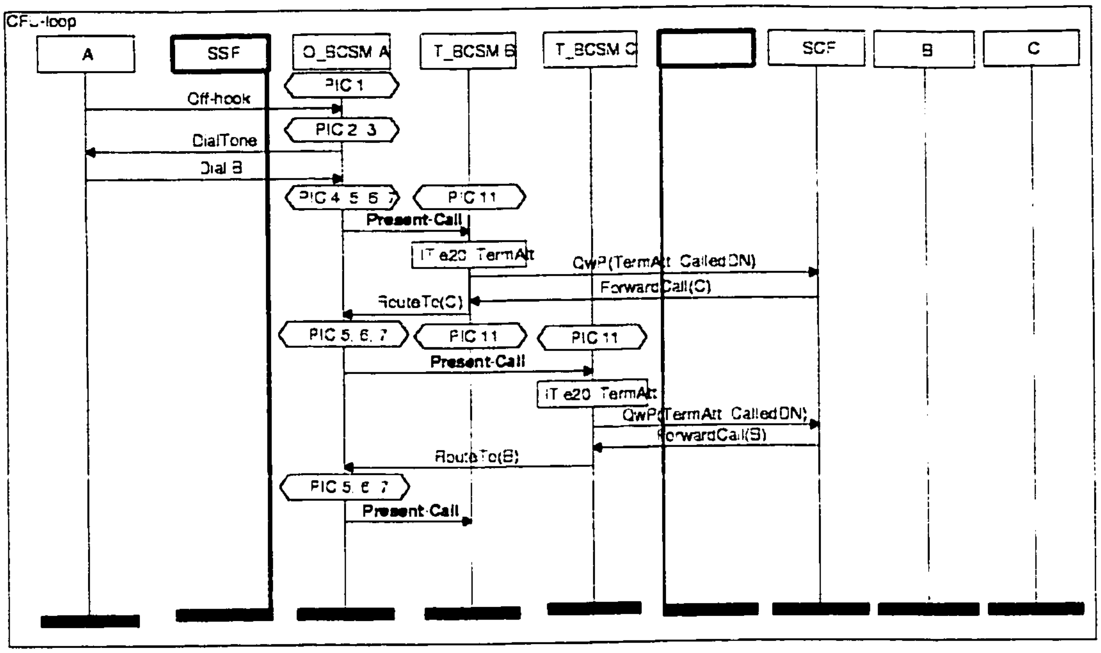


Figure 6-24: Two Call Forwarding in a Loop.

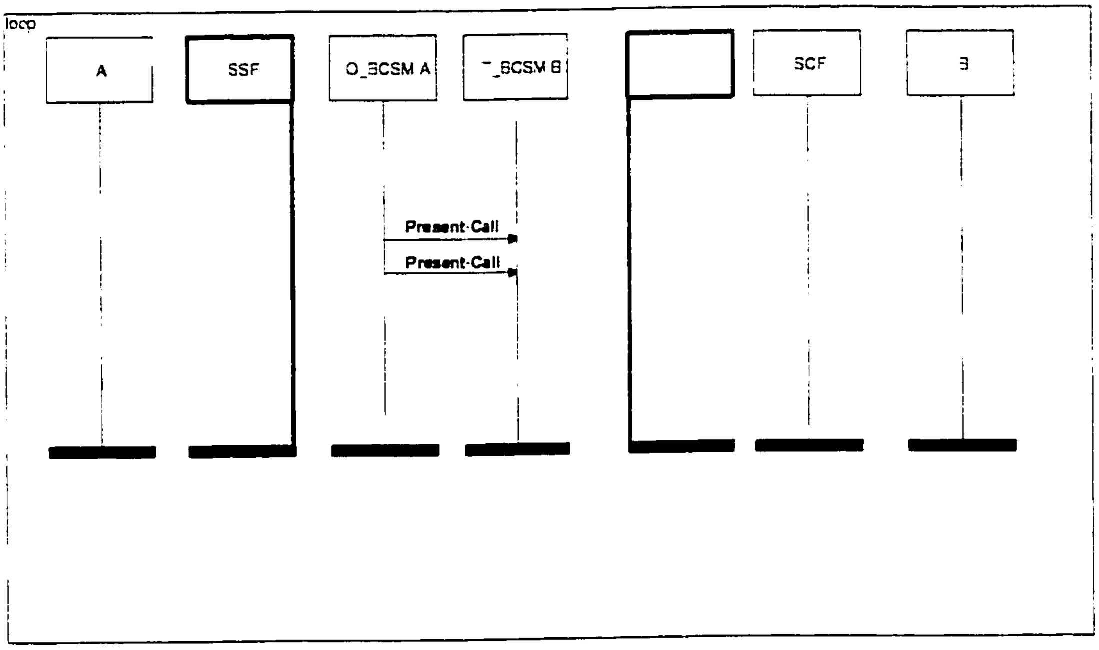


Figure 6-25: Specifying Loop as a Static Requirement.

6.2.18 Automatic CallBack and Automatic ReCall

Problem Description: *Customer **A** has the Automatic CallBack feature that continues to dial a busy number until it is answered. Customer **B** has the Automatic ReCall that returns the latest incoming call at the subscriber's request. **B**'s line is busy when **A** calls **B**, and the switching element serving **A**'s line keeps initiating new calls to **B** while the switching element serving **B**'s line stays busy returning every call* *made by A's AC feature. What's happening, according to Bellcore_Benchmark, is this:*

"When a subscriber dials the number X of a line that turns out to be busy, the feature Automatic CallBack (AC) can be activated. Automatic Call Back automatically redials X when X becomes idle. If X is idle, the call is considered completed, and the AC request is removed immediately. On the other hand, if X is busy, the AC request is placed on a queue of AC requests in the central office. Call set-up will be attempted again when X becomes idle, the subscriber's line is idle, and the subscriber has answered a special ringback. On the other hand, Automatic ReCall (AR) automatically returns the latest incoming call for the subscriber, whether the call from some line Y was answered or not. If Y is idle, the call completes and the AR request is removed. If Y is busy, the AR request is placed on a queue of AR requests in the central office. Call set-up will be attempted again when Y becomes idle, the subscriber's line is idle, and the subscriber has answered a special ringback, very much like the way Automatic CallBack operates."

Present configurations assume that the two features are simply unlikely to be executed at exactly the same time. No special provisions are made to disallow such behavior.

Benchmark Results: This problem happens due to synchronicity of some feature behavir. Our MSC specification style currently does not allow for the modeling of synchronous behavior and we cannot detect this interaction. In addition, MSC is a language for modeling asynchronous communications. Therefore, we cannot use MSC to specify synchronous feature behavior.

6.2.19 Long Distance Calls and Message Rate Charge Services

Problem Description: A Message Rate Charge Service allows subscribers a total number of local units per month. Since there is a local segment in every long distance call (which consists of at least three segments — two local accesses at each end and one provided by an interexchange carrier in between) should a customer be charged for the segments that have been completed even if the call did not go through successfully to its destination?

Benchmark Results: With our specification style, we are currently unable to model charging aspects of a feature. Therefore, we cannot detect this interaction.

6.2.20 Calling from Hotel Rooms

Problem Description: *A makes a short long-distance call from a hotel room phone and is not billed for the call. B makes a local call and allows the system to ring for three minutes. The call is not completed. He receives a bill from the hotel.*

The problem here is that "many hotels contract with independent vendors to collect access charges for calls originated from phones in their premises. Without being able to access to 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."

Benchmark Results: Same as the last example, we are unable to model charging aspects of a feature. Therefore, we cannot detect this interaction.

6.2.21 Billing in AIN Release 0

Problem Description: *The SSP assigns Area Number Calling a call-type code of 271 and assigns Originating Call Screening a call-type code of 275. The SCP must then tell the SSP to generate an AMA record that reflects the fact that both features are being used. How can it do this?*

The problem here is that, in AIN Release 0, the SCP instructs the SSP to generate a billing (AMA) record by sending a call-type code (3 numerics) to the switching element, but there is only room for one call-type code in the TCAP response. It becomes problematic to accomodate multiple services invoked by one

SCP query.

Benchmark Results: This example shows a low-level resouce contention problem. Our specification style currently does not support modeling of low-level system behavior such as the "room for call-type code" described in this example. Therefore, we cannot detect this interaction.

6.2.22 AIN-Based Services and POTS

Problem Description: *The Bellcore Benchmark describes a final problem type. A is a new subscriber to AIN-based services and the provisioning of these services for A is in progress. Once the trigger is set, the SSP will query the SCP database on all calls and trigger is set for A. However, the customer record is not updated in SCP and A has started to make a POTS call. IThe query triggered by A's POTS call results in a " customer record not found" response from the SCP and consequently, even the POTS call cannot go through.*

The problem here is that in the provisioning of some AIN-based services deployed in SCPs, updates must normally be made to two network components: the SSP must now know that a trigger should be established to query the SCP databases; and the SCP should have an updated record of customer information.

Benchmark Results: This interaction case describes the situation of the network which are undergoing some changes but the changes are not yet finalized. Our specification style currently cannot model such a situation, and we are unable to detect this interaction.

6.3 Benchmark Summary

Of the 22 interaction cases presented in the Bellcore feature interaction benchmark, we are currently able to detect 14 interactions using our approach. Our detection technique has maintained its consistency and accuracy in detecting the 14 detected interaction cases. Those we fail to detect belong to the following categories:

1. **Interactions that involve physical distribution of network components.** Examples 6.2.10 and 6.2.11 belong to this category. Both interactions involve directory numbers and their assignments to physical lines. For simplicity, our current version of the specification style has ignored modeling of physical locations of network components. For example, an "A" user instance identifies both the directory number of the user and the physical line of the user. Therefore, we are unable to detect this type of interactions.

2. **Inseparable scenarios that make detection difficult**. Example 6.2.15 belongs to this category. Our detection algorithm currently can only detect either interaction between two features or violation of static requirements on one feature. It cannot detect interactions with only one combined behavior as input. Features in an inseparable scenario can only be presented as a single combined behavior and we are therefore unable to detect their interactions.

3. **Interactions that happen because of synchronicity issues**. Example 6.2.18 belongs to this category. Since our specification style does not support modeling of synchronous behavior, we are unable to detect this type of interactions.

4. **Inconsistencies in charging**. Examples 6.2.19, 6.2.20 belong to this category. In our specification style, we haven't included mechanisms for modeling charging aspects of features. Therefore, we are unable to detect such interactions.

5. **Low-level interactions**. Example 6.2.21 shows a low-level resource contention problem. Our approach is not designed to handle low-level interactions.

6. **Unstable system states**. Example 6.2.22 belongs to this category. Our approach is designed to be used on a system that is relatively stable. We do not address interactions that happen during changing system states.

Of the above 6 categories of undetected interactions, some categories may be detected when we improve on the feature specification style to support modeling of certain detailed system behaviors. However, if we are not to lose many of the benefits of the feature specification style, such improvements must be controlled in a way that the overall style of the specification is maintained and kept simple.

Chapter 7

Conclusion

Nowadays, the society is heavily dependent on telecommunications and distributed systems, making the problem of feature interactions an issue of growing importance. In developing telecommunication systems, adding new functionalities costs huge numbers of person-hours on software modifications and testing. Much of this time is spent on detecting and resolving feature interactions. We anticipate that, as more services and features are offered in the future, the feature interaction problem will become more of a bottleneck in developing telecommunication systems. This situation has stimulated our research in the detection of feature interactions. This thesis presents our work in the course of this research.

7.1 Summary of the Contributions

We have addressed two objectives in this thesis: one is to present a framework for specifying telecommunication features using the formal language MSC; the other is to present our approach on the detection of feature interactions. Although not a direct objective of this thesis, we have shown the classification of different interactions to be helpful in understanding the nature of the interactions and to point towards the possible resolution or avoidance techniques.

In order to provide an evaluation that our approach meets the above objectives, we provide a comprehensive benchmark which shows the applicability of our approach and help us identify some limitations.

7.1.1 A Framework for Specifying Telecommunication Features

For the first objective, Chapter 4 presents the feature specification style for the formal language MSC. This style consists of two parts: the structures for components involved in the specification and the naming conventions for these stuctures. For the specification stuctures, we make an intuitive mapping between network components and available MSC language constructs. This gives us the means to represent all components involved in the specification. To eliminate further ambiguities that may arise due to namings of stuctures, we develop the naming conventions for all structures with the help of the EBNF definition formalism. The specification structures and their naming conventions together, define a subset of the overall MSC language dedicated to feature specifications.

We have shown in many examples that this specification style is high-level, user-friendly and can be used as a fast prototyping tool. In terms of limitations of the specification style, some of the limitations are the direct consequence of our design intention, i.e. to produce high-level specification of feature behavior. For example, we are currently not able to specify low-level system behavior such as memory usage, signaling capabilities etc. Some limitations are due to time constraints of the project. For example, charging aspects of features are not yet included in the current feature specification style; Also, physical locations of network components are not yet taken into account. Other limitations are due to limitations of the MSC language. For example, MSC currently lack support for specification of synchronous behavior. Due to all these limitations, feature interaction detection techniques developed on the basis of this specification style will be unable to detect certain interactions.

7.1.2 Detection of Feature Interactions

For the second objective of the thesis, we described our approach for detection of feature interactions in Chapter 5. This approach includes definition, classification of feature interactions, and detection techniques for each categories of interactions. Our methodology to detect feature interactions is to examine preservations of feature behavior. This methodology is reflected in the definition, the classification and detection of feature interactions.

Since the initial trigger, traces and static requirements appear to be the most important properties of a high-level feature behavior, we classified feature interactions into three categories corresponding to the three properties: Blocking of Initial Trigger, Interaction between Traces and Violation of Static Requirements.

For block of initial trigger interaction, we identified several situations that may arise due to trigger criteria, positions of triggers and positions that traces of features pass by. The detection algorithm simulates this identification process to find blocking of initial trigger, and if no blocking is found, decide what steps should be undertaken to detect other types of interactions.

For interaction between traces, we first identify overlapping parts between peer traces. For every pair of peer traces, we define their agreement as the collective agreements between their peer transitions. For every pair of peer transitions, we determine their agreements according to two properties: the next state and actions taken to reach the next state. At the lowest level, we propose a special form of trace that help to identify agreement between next states; We develop a labeling procedure that decide agreement between action sequences. Our algorithm checks peer transitions one by one, at the same time, the agreement trace is generated based on agreements between each pair of peer transitions. If any time during the process, a disagreement is found, we detect an interaction between traces. Otherwise, the agreement traces are generated and forwarded to the algorithm for detecting violation of static requirements.

Our technique for detecting violation of static requirements is based on our specification style for static requirements. Since our static requirements are specified using violation patterns, we simply perform a search on the combined behavior for the violation pattern. If the pattern is found, we detect a violation of static requirements.

7.1.3 Feature Interaction Benchmark

In Chapter 6, we have presented a comprehensive benchmark of our approach based on the Bellcore feature interactions benchmark. The benchmark is performed on a prototyped implementation of our approach as a feature interaction detection tool. The results of the benchmark prove that our approach is applicable. Of the 22 feature interactions, we are able to detect 14, and our detection technique has maintained its consistency and accuracy in detecting the 14 interactions cases. The benchmark also reveals some of the limitations of our approach. For example, we are currently unable to detect many low-level interactions, all interactions that involve charging aspects of features, some interactions that arise due to confusion on physical location of network components, some synchonicity problems, inseparable scenario problems and problems that happen in changing system states.

7.2 Directions for Future Work

We have seen that the approach presented in this thesis is simple, efficient, but has limitations. There is also a lot of work to do to expand on the existing framework to produce a general framework for detection, resolution or avoidance of feature interactions. In addition, adaptation measures are needed for addressing feature interactions that happen in other network architectures.

7.2.1 Addressing Current Limitations

Because of the limitations identified in the application to the benchmark, future research should be directed towards improvements on our approach:

1. For interactions that arise due to confusion on physical locations of network components, future work would be to take into account physical locations of network components in the specification style. Corresponding adjustments to the detection techniques should be made to include consideration of this type of interactions.

2. For inseparable scenario problems, fundamental changes should be made to the detection algorithms to not only check interactions between two separate features, but also be able to check interactions that happen between features in an inseparable scenario.

3. For synchronicity problems, progress in standardization in MSC will eventually include support for synchronous behavior. Up to that point, we should develop specification styles to support modeling of synchronous behavior and include corresponding detection techniques for detecting synchonicity problems. However, using other languages that support modeling of synchronous behavior, e.g. interworkings, might be another solution.

4. Some charging aspects of features cannot be directly specified using our MSC specification style as it has been developed so far and our approach does not as yet support detection of interactions involving charging aspects

of features. We recommend further work be directed towards adding these aspects to the specification style and developing algorithms that detect this type of interactions.

5. Currently, our techniques cannot detect some low-level interactions such as resource contention problems. Although we have developed our methods to be applied at a high level, it is always beneficial to incorporate low-level detection techniques as well. This may be done by modification of the feature specification style to enable modeling of low-level properties of features, although the specification may in this case become too complex and difficult to maintain. A more promising approach that may result in a general framework for detecting all levels of interactions would be to integrate our techniques with existing low-level detection techniques.

6. For problems that happen only in unstable system states, since our technique is an off-line detection technique and is supposed to be used only in system states that are relatively stable, it is not rational to include support for detection of such interactions. However, it is always desirable to integrate our approach with existing techniques that address these interactions.

7.2.2 Towards Resolution or Avoidance

Although not a direct objective of this thesis, we have shown that the classification of different interactions to be helpful in understanding the nature of the interactions and to point towards possible techniques for resolution or avoidance. Therefore, possible work at this stage may be to develop resolution and avoidance techniques based on our general framework.

7.2.3 Adapting to Other Network Architectures

The specification style we have presented is based on the Bellcore AIN architecture. This limits the applicability of our approach to other network architectures. However, the underlying detection methodologies remain valid. It appears desirable at this stage to develop specification styles for other architectures and apply the present method to features of other networks.

Nowadays, hybrid services consist of a considerable amount of new services offered to customers. Hybrid services are services that span over several different kinds of network architectures. For example, an IP phone service can use both an IN and an Internet network. This makes feature interaction detection in hybrid services more difficult than in traditional single-network services. A feasible solution may be to develop an abstract feature specification style that can represent features of all or most of the network architectures. This work will not only facilitate interaction detection in hybrid systems, but help to build an approach for detecting interactions in all or most network architectures.

References

[1] Bell Atlantic, "Intelligent Network (IN) Tutorial", available at URL: http://www.webproforum.com/in/index.html, International Engineering Consortium, June 1998.

[2] The SCORE Project, "Report on Methods and Tools for Service Creation. Service Interaction Analysis", Deliverable D206-Vol. 1. R2017/SCO/WP2/DS/P/031/b1, RACE Project 2017, December 1995.

[3] T. F. Bowen, F. S. Dworak, C. -H. Chow, N. D. Griffeth, G. E. Herman, and Y. -J. Lin, "Views on the feature interaction problem", In: *Proceedings of the 7th International Conference on Software Engineering for Telecommunications Switching Systems*, pp. 59-62, Bournemouth, 1989.

[4] L. Bouma and H. Velthuijsen (eds.), *"Feature Interactions in Telecommunication Systems", Proceedings of the Second International Workshop on Feature Interactions in Telecommunications Systems*, IOS Press, Amsterdam, May 1994.

[5] E. J. Cameron, N. Griffeth, Y. -J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen, "A Feature Interaction Benchmark for IN and Beyond", In: [4, pp. 1-23].

[6] J. Muller, H. Blanchard, P. Combes, A. M. Daniel, and J. M. Pageot, "Perfection is not of this world: debating a user-driven approach of interaction in an advanced intelligent network", In: *Proceedings of TINA*, 1993.

[7] Edmund M. Clarke and Jeannette M. Wing, "Formal Methods: State of the Art and Future Directions", *ACM Computing Surveys*, vol. 28, number 4, pp. 626-643, December 1996.

[8] ITU-T, "Recommendation Z.120: (1996) Message Sequence Chart (MSC)", Geneva, April 1996.

[9] J. M. Duran and J Visser, "International Standards for Intelligent Networks", In: *IEEE Communications Magazine*, pp. 34-42, February 1992.

[10] Uyless Black, "The Intelligent Network", Prentice Hall PTR, 1998.

[11] CCITT (now ITU-T), "Recommendations Q1200 series: Intelligent Network Recommendation", Geneva, 1992.

[12] CCITT (now ITU-T), "Recommendation Z.100: Specification and Description Language (SDL)" and Annex A to the recommendation, Geneva, March 1992.

[13] CCITT (now ITU-T), "Recommendation Z.100: Specification and Description Language (SDL) – Annex D: SDL User Guidelines", 1988.

[14] J. Grabowski, E. Rudolph, "Putting Extended Sequence Charts to Practice", In: *SDL'89: The Language at Work*, O. Faergemand and M. M. Marques (eds.), North Holland, 1989.

[15] P. A. J. Tilanus, "A formalisation of Message Sequence Charts", In: *SDL'91: Evolving Methods*, O. Faergemand and R. Reed (eds.), North Holland, 1991.

[16] CCITT (now ITU-T), "Recommendation Z.120: (1992) Message Sequence Chart (MSC)", Geneva, 1992.

[17] ISO, ISO 8807:1989, "Information processing systems -- Open Systems Interconnection – LOTOS -- A formal description technique based on the temporal ordering of observational behavior", 1989.

[18] ISO, ISO 9074: 1997, "Information processing systems --- Open systems interconnection --- Estelle --- A formal description technique based on an extended state transition model", Geneva, 1997.

[19] G. Booch, J. Rumbaugh, "The Unified Software Development Process", Addison-Wesley, 1999.

[20] I. Jacobson, "Object-Oriented Software Engineering – A Use Case Driven Approach", Addison-Wesley, 1992.

[21] M. Anderson, J. Bergstand, "Formalizing Use Cases with Message Sequence Charts", Master Thesis, Lund Institute of Technology, 1995.

[22] VERILOG SA, "User's Guide – ObjectGeode MSC Editor", version 4.0, D/GEMS/UA/400/908, VERILOG, April 1999.

[23] Wenwei Weng, "Use HMSCs for IN Services Specification", M. Eng. Project Report, Department of Electrical and Computer Engineering, Concordia University, December 1998.

[24] J. Bergstra and W. Bouma, "Models for Feature Description and Interaction", In *Feature Interactions in Telecommunications Networks*, vol. 4, P. Dini, R. Bouma, and L. Logrippo (eds.), pp 31-45, IOS Press, Amsterdam, 1997.

[25] Alfred Aho, Sean Gallagher, Nancy Griffeth, Cynthia Schell, and Deborah Swayne, "Sculptor with Chisel: Requirements Engineering for Communications Services", In: *Feature Interactions in Telecommunications Networks*, vol.5, K. Kimbler, L. G. Bouma (eds.), IOS Press, Amsterdam, 1998.

[26] Bellcore, "LSSGR: Signaling for Analog Interfaces, Generic Requirement", GR-506-CORE, Issue 1, June 1996.

[27] D. Keck, "Identification of Call Scenarios with Potential Feature Interactions", In: *International Workshop on Advanced Intelligent Networks (AIN'96)*, Passau, March 1996.

[28] J. Blom, "Formalization of Requirements with Emphasis on Feature Interaction Detection", In: *Feature Interactions in Telecommunication Networks*, vol.4, P. Dini, R. Bouma, and L. Logrippo (eds.), pp. 61-77. IOS Press, Amsterdam, June 1997.

[29] Bellcore GR-1298-CORE, Issue 4, September 1997.

[30] S. Mauw, "The formalization of Message Sequence Charts ", In: *Computer Networks and ISDN Systems – SDL and MSC*, vol. 28, Number 12, June 1996.

[31] E. Rudolph, J Grabowski, and P. Graubmann, "Tutorial on Message Sequence Charts (MSC'96)", Tutorial of the *FORTE/PSTV'96 conference in Kaiserslautern, Germany*, Kaiserslautern, October 1996.

[32] Axel van Lamsweede, Robert Darimont and Emmanuel Letier, "Managing Conflicts in Goal-Driven Requirements Engineering", In: *IEEE Transactions on Software Engineering*, vol. 24, Number 11, November 1998.

[33] James J. Garrahan, Peter A Russo, Kenichi Kitami, and Roberto Kung, "Intelligent Network Overview", In: *IEEE Communications Magazine*, pp. 30-36, March 1993.

[34] K. Kimbler, C. Capellmann, and H. Velthuijsen, "Comprehensive Approach to Service Interaction Handling", In: *Computer Networks and ISDN Systems*, vol. 30, September 1998.

[35] C. Capellmann, P. Combes, J. Pettersson, B. Renard, and J.L. Rutz, "Consistent Interaction Detection - A Comprehensive Approach Integrated with Service Creation", In: *Feature Interactions in Telecommunication Networks IV*, pp.183-197, IOS Press, Amsterdam, 1997.

[36] I. Aggoun, and P. Combes, "Observers in the SCE and SEE to Detect and Resolve Service Interactions", In: *Feature Interactions in Telecommunication Networks IV*, pp. 198-212, IOS Press, Amsterdam, 1997.

[37] H. Velthuijsen, "Issues of Non-Monotonicity in Feature Interaction Detection", In: *Feature Interactions in Telecommunication Networks III*, pp. 31-42, IOS Press, Amsterdam, October 1995.

[38] D. Keck, and P. Kuehn, "The Feature and Service Interaction Problem in Telecommunications Systems: A Survey", In: *IEEE Transactions on Software Engineering,* vol. 24, Number 10, October 1998.

[39] K. J. Turner, "An architectural description of intelligent network features and their interactions", In: *Computer Networks and ISDN Systems,* vol. 30, September 1998.

Appendix A

AIN 0.1 PICs TDPs and Triggers

This appendix attempts to provide a detailed description of the PICs, TDPs and triggers defined in AIN 0.1. The purpose of this appendix is to provide a reference to the actual design purposes, operations, and relationships of these PICs, TDPs and triggers. We draw extensively from Black's book [10] on Intelligent Networks. For an even more detailed analysis of the concepts discussed in this appendix, please refer to Black's book.

In chapter 2, Figure 2-10 has depicted the originating and terminating basic call state machines of AIN 0.1. Here, we explain the PICs, TDPs and Triggers that are defined in this model.

1. PICs

- **PIC1 0_Null:** Line or trunk interface is idle. Supervision is being provided.
- **PIC2 Authorized Origination Attempt:** Authority of user to place outgoing call with certain properties (e.g., line restrictions) is being verified.
- **PIC3 Collect Information:** SSP collects initial information (e.g., dialed address digits) from the user.
- **PIC4 Analyze Information:** SSP interprets and translates information according to the specified numbering plan. Routing address and type of calls (local, long distance) are determined.
- **PIC5 Select Route:** SSP is interprets the routing address and type of call to select the next route.

- **PIC6 Authorized Call Setup:** SSP verifies the authority of the calling party to place this particular call.
- **PC7 Send Call:** SSP sends an indication of desire to set up a call to the specified Called Party ID to the terminating call portion.
- **PIC8 O_Alerting:** SSP waits for the terminating party to answer.
- **PIC9 O_Active:** Connection established between calling and called party. Call supervision is provided.
- **PIC10 O_Disconnect:** Connection is left intact and appropriate timing may be started depending on the indication received, the access type and the position of the SSP in the connection.
- **PIC11 T_Null:** For a non-ISDN line, a conventional or SS7 trunk, or a private facility trunk, the line or trunk interface is idle (i.e., no calls exists), and the SSP provides supervision on the line or trunk. For an ISDN interface although the call reference of the cleared call has been released, other calls may still exist.
- **PIC12 Authorize Termination:** SSP verifies the authority to route this call to the terminating access (e.g., check business group restrictions, restricted incoming access to line etc.).
- **PIC13 Select Facility:** The busy/idle status of the terminating access is determined.
- **PIC14 Present Call:** The SSP informs the terminating access of the call (e.g., line serizure with power ringing, or Q931 SETUP message).
- **PIC15 T_Alerting:** SSP alerts the terminating resource and waits for the terminating party to answer.
- **PIC16 T_Active:** Connection established between calling and called party. Call supervision is provided.
- **PIC17 T_Disconnect:** Connection is left intact and the appropriate timing may be started, depending on the access type and the position of the SSP in the connection.

2. TDPs

- **e1 Origination Attempt:** SSP considers an Origination Attempt event to have occurred when it receives an off-hook indication from an idle non-ISDN line, a SETUP message from an ISDN interface, an Initial Address Message (IAM) from an SS7 trunk or when a trunk within a Trunk Group (TG) supporting conventional signaling is seized, or a seizure signal is received on a private facility.
- **e2 Origination Attempt Authorized:** SSP detects an originated event when the authority to place an outgoing call is verified.
- **e3 Information Collected:** SSP detects an information collected event when the complete initial information from the caller is available.
- **e4 Information Analyzed:** SSP detects an information analyzed event when the CallPartyID, the TypeOfCall, and when appropriate, the PrimaryTrunkGroup and PrimaryCarrierID parameters are determined by the SSP. The trigger analyses includes Carrier Identification Code (CICs), 3/6 digits and 7/10 digit DN North American Numbering Plan (NANP) addresses, feature codes, carrier access codes, prefixes, customized dialing plan addresses.
- **e5 Route Selected:** SSP detects a route selected event when the originating routing information fro the call has been determined.
- **e6 Origination Authorized:** SSP detects an origination authorized event when the authority to place the call is verified. For an SS7 trunk interface, if the received IAM indicates that a continuity check is being performed on the call connection and the call terminates to an analog line or ISDN interface subtending the SSP, this event occurs when a continuity message (COT) with a successful indication is received from the originating access.

- **e7 O_Term. Seized:** The SSP detects a O_Term. Seized event when an indication of a Call Accepted event is received from the terminating call portion or when certain abnormal cases occur in ISDN when the call is offered to an ISDN interface (or EKTS group) and no user equipment has responded.
- **e8 O_Answer:** The SSP detects an O_Answer event when an indication of a connected event is received from the terminating call portion.
- **e9 O_Abandon and O_Calling Party Disc.:** SSP detects a O_Abandon and O_Calling Party Disc. Event when the calling party disconnects, which can be the result of one of the following when a SSP receives:
  - An on-hook from a caller served by a non line(following flash timing), or
  - A call clearing message from a caller served by an ISDN interface, or
  - A disconnect indication for a conventional trunk or a private facility, or
  - A Release (REL) message for an SS7 trunk.
- **e10 Feature Requested:** SSP detects a Feature Requested event from PICs 1,3,4,5, and 6 when one of the following has occurred:
  - From an interface supporting ISDN Class I equipment, this corresponds to the ISDN user sending a feature activator.

    From PIC1, is expected in an INFORMATION message with a null call reference value. From PICs 3-6, this is expected in a SETUP or an Information message containing the call reference of the specific call.
  - This event is detected from PICs 3-6 when a switch-hook flash is entered from an analog line or private facility in the special situation where the user is initiating another two-party call in addition to an existing two-party call. Otherwise, this event cannot occur from a non-ISDN line, conventional or SS7 trunk, or from an interface supporting ISDN Class II EQUIPMENT.

    SSP detects Feature Requested events for Feature Requested trigger at the NULL, COLLECTING INFORMATION, ANALYZING INFORMATION, SELECTING ROUTE, AND AUTHORIZING CALL SETUP PICs.

- **e11 O_Mid_Call:** SSP detects a Feature Requested event when one of the following has occurred:
  - From a non-ISDN line or private facility, this event occurs when the user executes a switch-hook flash.
  - From an interface supporting ISDN Class I equipment, this corresponds to the ISDN user sending a feature activator, an ISDN HOLE message, or an ISDN RETRIEVE message. When the controlling leg applies to an EKTS group, a HOLD message can only be used to trigger AIN (or be treated as an event), when no other user in the EKTS group is connected to the call. If there are other users connected to the call, the HOLD message causes the user to be separated from the call, as defined user EKTS and is transparent to AIN . In addition, when the controlling led apples to an EKTS group, a RETRIEVE message can only be used to trigger AIN (or be treated as an event) if no user is connected to the call. If one or more users are already connected to the call, the RETRIEVE message causes the user to be connected to the call, as defined by EKTS, and is transparent to AIN.
  - This event cannot occur from a conventional or SS7 trunk, or from an interface supporting ISDN Class II equipment.

  SSP detects Feature Requested events for mid-call triggers at the CALL PROCEEDING, WAITING FOR ANSWER, ACTIVE, and RELEASE PENDING PICs.
- **e12 Cleared:** SSP detects a Cleared event when an indication that the called party disconnected is received from the terminating call portion, which can result from the following:
  - For an intra-switch call to a non-ISDN line, the SSP receives an on-hook indication from the terminating line (and the SSP has provided switch-hook flash timing).
  - For an intra-switch call to an ISDN interface, the SSP receives a call clearing message from the terminating user. In addition, this occurs when a

Call Rejected event is received from the terminating call portion that does not indicate that the user is busy. For an intra-switch call to an EKTS group, only one user in the terminating EKTS group is connected to the call and that user sends a call clearing message (if ISDN) or an on-hook indication (if an analog user, and switch-hook flash timing ahs been provided). Moreover, the Cleared event occurs when an EKTS user (in the terminating EKTS group) disconnects and under EKTS the switch determines that the call should be cleared.

- For a conventional trunk or private facility, the SSP receives a disconnect signal.
- For an SS7 trunk, a Release (REL) or a suspend (SUS) message is received.

- **e13 O_Disc.** Complete: SSP detects a O_Disc. Complete event that occurs when the timed release disconnect timer expires (at this SSP). For a terminating access of an interface supporting ISDN Class I and Class II equipment, this event is equivalent to the O_Disconnect event.
- **e14 Orig. Denied:** SSP detects an Orig. Denied event when the authority to place an outgoing call is denied. This event does not apply for private facilities or an interface supporting ISDN Class II equipment. For convention and SS7, this event occurs when glare is detected and the other call is given precedence.
- **e15 Collect Timeout and Collect Information Failure:** 1. SSP detects a Collect Timeout event when complete initial information was not received before a normal inter-digit timer expires. For an SS7 trunk the Collect Timeout event corresponds to the IAM not containing the information necessary to process the call. There is no timing involved. 2. SSP detects a Collect Information Failure event when it is unable to perform the information collection function because of a lack of switch resources (e.g., a digit collector is unavailable). This event can only be a requested event; it cannot be a trigger.
- **e16 Invalid Info:** SSP detects an Invalid Info event when the collected information is invalid.

- **e17 Network Busy:** SSP detects the Network Busy event when all routes are busy.
- **e18 Auth. Failure:** SSP detects the Auth. Failure event when the authority to place the call is denied. For an SS7 trunk interface, the Auth. Failure event occurs when a COT with a failure indication is received.
- **e19 O_Called party Busy:** SSP detects the O_Called party Busy event indication when one of the following occurs:
  - An O_Called party Busy event specifying user busy is received from the terminating portion of the call (i.e., network-determined-user busy or user-determined-user busy, or
  - A Presentation Failure event specifying user busy is received from the terminating call portion.
- **e20 Termination Attempt:** SSP detects a Termination Attempt event when it receives an indication of a desire to deliver a call from the originating portion of the call.
- **e21 Term. Authorized:** SSP detects a Term. Authorized event when the SSP determines that a call is authorized to be routed to a terminating user.
- **e22 Term. Resource Available:** SSP detects a Term. Resource Available event when the SSP determines that an idle facility (e.g., a non-ISDN line, ISDN interface, trunk or private facility) or a position in a queue is available.
- **e23 T_Term. Seized:** SSP detects a T_Term. Seized event when the called party is being alerted of the call. The Call Accepted event can be the result when the SSP:
  1. Provides power ringing to the terminating non-ISDN line; or
  2. Receives an ALERTING message or PROGESS message from an ISDN user; or
  3. Seizes a conventional trunk and no continuity check is required in the previous connection or, if a continuity check was required, a COT with a successful indication has been received; or

4. Receives an ACM for an SS7 trunk in response to an IAM; or
5. Seizes a private facility.

- **e24 T_Answer:** SSP detects a T_Answer event when the called party answers the call. A connected event can be the result when the SSP receives:
  - An indication that the terminating party served by an non-ISDN line when off-hook; or
  - A Connect message from an ISDN user; or
  - An answer indication on a conventional trunk or private facility; or
  - An ANM for an SS7 trunk in response to an IAM.
- **e25 T_Disconnect:** SSP considers a T_Disconnect event to be the result when the SSP:
  1. Receives an on-hook indication from a non-ISDN line; or
  2. Receives a call clearing message from an ISDN user (or a call terminating to an EKTS group, a user in the EKTS has sent a call clearing message and under EKTS, the call is to be cleared); or
  3. The SSP receives a disconnect signal from a configuration trunk or private facility; or
  4. Receives RELEASE (REL) or SUSPEND (SUS) message for an SS7 trunk.
  5. The T_Disconnect event may also result from error situations such as when an ISDN interface goes down (this may be detected at the data link layer of the protocol.
- **e26 T_Disconnect Complete:** SSP detects a T_Disconnect Complete event when the release timer expires (at this SSP). For a terminating access of an interface supporting ISDN Class I and Class II equipment, this event is equivalent to the Disconnected event.
- **e27 T_Mid-Call:** SSP detects Feature Requested events for mid-call triggers at the ALERTING and ACTIVE PICs of the TBCM. The SSP detects a Feature Requested event when one of the following has occurred:

1. From the ALERTING PIC, the Feature Requested event corresponds to an ISDN Class I equipment user sending a feature activator.
2. From the ACTIVE PIC, the Feature Request corresponds to one of the following:
    - From a non-ISDN line or private facility this event occurs when the user executes a switch-hook flash.
    - From a user served by ISDN Class I equipment, this corresponds to the ISDN user sending a feature activator, an ISDN HOLD message, or an ISDN RETRIEVE message.
    - This event cannot occur from a conventional or SS7 trunk, or from an interface supporting ISDN Class II equipment.

- **e28 T_Abandon & T_Calling Party Disc.:** SSP detects a T_Abandon & T_Calling Party Disc. Event when the terminating portion of the call receives indication that the originating portion is clearing.
- **e29 Term. Denied:** SSP detects the Term. Denied event when the authority to route the call to the called party is denied.
- **e30 T_Busy:** SSP detects a T_Busy event when the terminating access is found to be busy. The T_Busy event may also be detected as a result of an analog line being out of order, marked as busy by a customer make-busy key, or as a result of certain maintenance actions.
- **e31 Presentation Failure:** SSP detects the Presentation Failure event when the call is rejected by the call party, or an expected response is not received for a call terminating to a trunk. A Presentation Failure event can be the result of one of the following occurring:

1. A call cannot be rejected from a non-ISDN line.
2. In ISDN (and EKTS), a call is rejected when one or more users send a call clearing message in response to the terminating call, or one or more users send any message other than an ALERTING or CONNECT message an call setup timers expire.

3. For a conventional trunk or for a private facility, the call rejected event is detected when either of the following occur:
   - The SSP does not receive the proper acknowledgment signals (e.g., an expected wink is not returned).
   - The SSP receives a disconnect indication before an answer indication has been received.
4. For an SS7 trunk, the call rejected event is detected when any of the following occur:
   - The second attempt continuity check on the outgoing circuit fails.
   - The SSP receives a REL in response to an IAM.
   - The SSP does not receive a CRA after sending a CRM for the second time.
   - The SSP does not receive an ACM or ANM in response to the IAM in the allotted time (if timing is applied at the SSP).
   - The SSP receives a REL after an ACM has been received.

- **e32 T_No Answer:** SSP detects the T_No Answer event when the called party does not answer before the switch-based ringing timer expires. This timer is used for services such as Call Forwarding on No Answer.

3. Triggers

AIN 0.1 supports several triggers. AIN 0.2 supports four new triggers. As a summary, Table A-1 providers the triggers that are defined in AIN 0.2.

TDP	*Triggers*
Origination Attempt (e1)	Off-Hook Immediate
Information Collected (e3)	Off-Hook Delay Channel Setup PRI (Primary Rate Interface) Shared Inter-office Trunk
Information Analyzed (e4)	BRI(Basic Rate Interface) Feature Activation Indicator Public Feature Code Customized Dialing plan 3/6/10 Digit N11
Network Busy (e17)	AFR (Automatic Flexible Routing)
O_Called_Party_Busy (e19)	O_Called_Party_Busy
Cleared (e12)	O_No_Answer
T_Busy (e30)	T_Busy
T_No_Answer (e32)	T_No_Answer

Table A-1 Triggers Defined in AIN 0.2.