Modeling of Intelligent Networks Using SDL and an Approach for Feature Interaction Detection

Yuan Peng

A Thesis
in
The Department
of
Computer Science

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

Concordia University

Montreal, Quebec, Canada

March, 1998

0-612-39490-5

ABSTRACT

Modeling of Intelligent Networks Using SDL and an Approach for Feature Interaction Detection

Yuan Peng

Features are novel telecommunication functions that are provided to users as individual commercial offerings. Intelligent Network (IN) is a new network framework proposed by ITU-T (International Telecommunication Union) in order to enable fast and cost-effective introduction of a large number of useful features. However, as more and more features are developed, various kinds of unexpected interference emerge among multiple features. Such interference prevents the features from fulfilling their tasks correctly. This is the problem known as Feature Interaction (FI).

This thesis concentrates on the analysis and detection of Feature Interactions in Intelligent Network systems. First we present our work in modeling IN using the formal description language SDL (Specification & Description Language) based on the Distributed Functional Plane (DFP) of the Intelligent Network Conceptual Model (INCM) that is defined in the ITU-T Q.12xx series recommendations. Use of SDL in this model makes it precise, concise and free of ambiguity. This model serves as a platform to study IN entities, features and FIs. Secondly, a new detection approach is proposed in order to discover FIs efficiently. This approach attains strong detection ability and low computational complexity by focusing on scanning for major causes that lead to FIs rather than studying detailed feature behaviors. Most known FIs listed in Bellcore FI Benchmark as well as several new FIs were successfully detected using our method. Our IN model in SDL can also be used to simulate the call processing and feature running situations, which provides a potential way to verify the FI detection results of our approach, especially for new FIs.

Acknowledgments

First of all, my sincere gratitude is dedicated to both of my supervisors for their indispensable encouragement, guidance and support throughout the period of my thesis work. Supervisor 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 as well as providing financial support to me. Our project scheduling, result reviewing, experiment resource backing up, text presentation enhancement and outside coordination contributed by supervisor Dr. Peter Grogono are indeed essential and excellent as well.

In addition, I am very grateful to Dr. Bill Atwood for helping me improve the text presentation of my thesis. His careful review is highly appreciated.

Among the people who also deserve my special thanks are graduate secretary Ms. Halina Monkiewicz and coordinator Ms. Pauline Tung Dubois who have provided extensive miscellaneous help during my graduate study, and the analyst group of Computer Science Department that offers effective and prompt technical support, especially Mr. Stan Swierz who maintained the running environment of the software tool ObjectGeode.

Also worth to mention, technical communication and suggestions from members of our telecommunication software research group are helpful and appreciated.

Finally, I would like to extend my deepest thanks and respect to my wife Jie who has accompanied me overcoming numerous difficulties and hardships. It is her extraordinary understanding, support and unselfish love that have made the completion of this project and thesis possible.

Table of Contents

List of Figures

List of Tables

Chapter 1 Background and Introduction

The introduction of telecommunication techniques such as telephone, fax and computer networks has greatly enhanced our communication ability with regard to speed, quality and capacity. Nowadays, supported by rapid technology advances in Hi-Tech areas such as semiconductor and computer industry, telecommunication has become the dominant way of communication and is facing booming demands as well as much higher requirements. Users do not stop at normal two-party conversations with good quality and speed. More powerful call functions are needed to provide better services and address users' individual communication needs.

With increasing number of new call functions being included into the communication systems, it becomes very difficult to manage the sophisticated network control software using the conventional network architecture. The Intelligent Network (IN) proposed by ITU-T (International Telecommunication Union) represents a new type of network infrastructure under which novel call functions can be added to existing networks easily and quickly. This thesis describes our work related to IN systems including the introduction of IN architecture and entities, modeling of IN using the Specification & Description Language (SDL), and a new detection approach for discovering Feature Interactions (FI). Our study is based on a formal model that is essentially an abstraction of IN-structured telephony systems. However, the principles derived here can be applied to digital data communication networks as well.

1.1 Background

Real-time telecommunication is first introduced by the invention of telephone systems. Since then, telephony system has become the major communication tool in our daily life. In the past decades, switching technology has evolved through several stages: *dedicated wire connection*, *operator-performed switching*, automated *electromechanical switching* and *computer program-controlled switching*. Most networks today are running under the control of sophisticated computer programs.

After users are mostly satisfied with the speed and quality of telephone systems, they began to ask for more powerful call functions that allow them to communicate in a way that fits their individual needs. Network providers have developed many novel call functions to satisfy the user needs and remain competitive. Systems with no such new call functions are referred to as POTS (*Plain Old Telephone System*) to indicate that they are only capable of processing basic telephone calls between two parties. Newer systems are built based on POTS and able to deal with more complex call connections and tasks such as Three-Way Calling that allows three parties to be in the same call conversation simultaneously.

1.1.1 Telecommunication Features

For technical as well as commercial reasons, new call functions are included in the networks and offered to users in separate functional packages. These packages are not part of standard POTS systems and thus carry extra charges when being used.

Features in telecommunication systems are such packages of incrementally added functions providing advanced call services to subscribers or the telephone administration organizations [Bowen 89]. These packages, which are provided to users on a subscribe-and-use basis, are also called "*telecommunication services*", "*service features*", and "*services*" in different literature.

Strictly speaking, a telecommunication *service* represents a relatively independent and complete commercial call-function package while *feature*s are those atomic new call-functions that constitute the *service*s (as defined in [Q.1211]). However, in this thesis, we use the term "*feature*" instead of "*service*" and "*sub-feature*" instead of "*feature*". The reason is to avoid possible confusion between the telecommunication *service* and a SDL concept that carries the same name "service" but has completely different meaning.

As an example of telecommunication *feature*, Call Waiting (CW) is defined as: *"This feature allows the called party to receive a notification that another party is trying to reach his line while he is busy talking to another calling party*" in [Q.1211] by ITU-T. Intuitively speaking, if you subscribe CW, the CW will monitor your telephone line whenever you have a call, no matter you called out or you received the call. Should someone else call you at this time, CW works for you by sending a ringing tone to the caller so that it seems to the caller you are idle, and CW also sends you a special call-waiting tone indicating that another in-coming call is waiting. You may ignore it if you do not want to interrupt the current conversation and CW will self-mute after sending you a few reminding tones, in this case the caller feels as if nobody were available to answer the phone. Also if you do not want to miss the second call, you can "Hook-Flash" (depress and release the hook tongue quickly) to accept talking to the new caller and put the currently talking party on hold. After that, you can use "Hook-Flash" to switch back and forth between the two calls as you wish until the end of one of them, then you have a normal call.

Figure 1-1 shows the difference between the situations of with and without the feature CW. In the first case, Busy-Tone is sent to the party who tried to call you when you are already on the phone. While in the second case, CW allows the second call to be connected to your telephone line. CW works as a two-way switch and maintains both calls for you at the same time.

Telephone companies quickly realized the importance and profitability of such novel features in their networks. Developing more powerful features into their networks has become a major competition tactic among telephony system providers. Quite a few good features were introduced such as "1-800 free call", "Televoting", "Centrex", "Call Waiting", "Caller Number Delivery", etc.

1.1.2 Intelligent Networks (IN)

The use of computer program controlled switching technique means all the call switching and processing tasks inside a telecommunication network is done by computer software residing in the switches. Whenever the rules of processing calls are changed, the software inside the switches also needs to be updated.

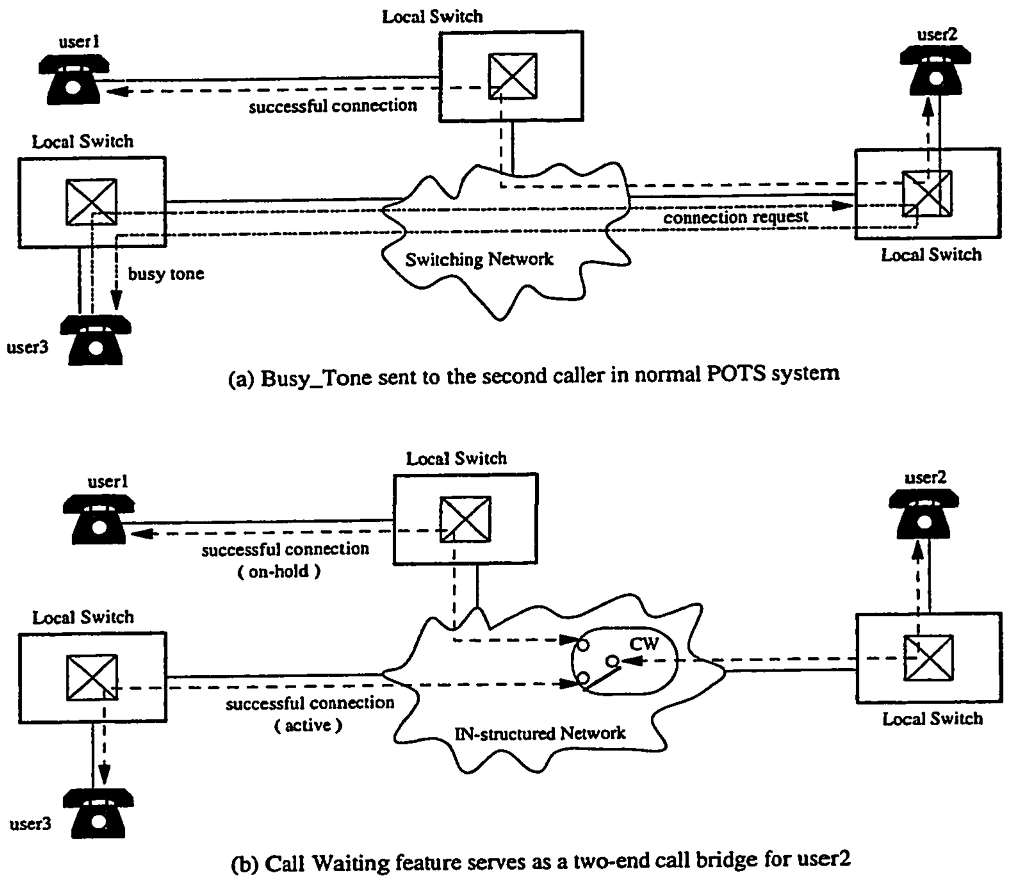


Figure 1-1: Illustration of feature Call Waiting

As we mentioned, features are novel functions added to the existing networks that modify the behavior of POTS in order to realize advanced or user-customized communications. That is why during the early stage of feature development, features are mainly "patches" of software programs that attached to existing Basic Call Processing (BCP) software in the switching-and-processing devices (also simply called "switching devices" and "switches"). These "patches" update the normal behavior of BCP in certain ways that added functions of the new features can be performed under appropriate call situations.

It is feasible to use this mechanism when a small number of features exist in the network. However, with the proliferation of various features, the control inside the switching devices becomes very complex and difficult to design, test and maintain. More important, adding a feature to the system becomes tedious because of the excessive long time and high costs it takes to upgrade the software in every switching device of the network. By the time when the upgrade is done, the feature might have already evolved !

Intelligent Network (IN) has been proposed in which novel features can be efficiently designed, created and deployed. The primary goals of the Intelligent Network include uniform underlying network architecture, vendor-independent functional support, re-usable fragments of feature software library, standardized feature definition method, efficient deployment and maintenance, etc. Also short development cycle and cost minimization are important objectives.

In an IN-structured network, the responsibilities for normal call processing and novel call functions are separated. Switching devices mainly carry out the BCP and special intelligent units will provide the "feature functions". With this paradigm, the complexity of control software in the network is reduced and feature development on a large scale becomes feasible.

1.1.3 Feature Interaction Problem

The introduction of Intelligent Network has eased the difficulty of feature creation, deployment and maintenance. Thus more and more features are quickly developed and loaded into the communication systems. With the abundance of new features co-existing in the networks, a technical problem called *Feature Interaction (FI)* has been discovered in the late 80's [Bowen 89].

Generally speaking, Feature Interaction is understood as all kinds of unexpected interference among multiple features, which prevents at least one of them from performing the designated tasks correctly. One reason for FI is that the feature designer is unaware of the existence of other features, thus has not taken enough measurements to shield his feature from the possible implications of other features. Also there are many other reasons like system limitations, signaling ambiguity and semantic conflict of different features. *Due to FI, there is no guarantee that a currently functioning feature will continue to work well, when some other features are introduced into the system, no matter how careful we have designed, implemented and tested it.*

As a major research topic in IN field for years, FI has been proven to be a hard problem. It occurs in many forms and can be introduced during almost every phase of a feature's life cycle [Kimbler 97]. Probably it is not feasible to detect and resolve all kinds of FIs at any single stage of features' life cycles or with any single technique [Bryce 94]. However many academic researchers and large telecommunication companies are continuing their efforts to solve the FI problem, at least at a practical extent.

1.1.4 Formal Description Techniques (FDT)

Formal Description Techniques are techniques used to specify and analyze target systems in a formal way so that they can be characterized easily and without ambiguity. FDTs actually refer to a variety of mathematical modeling techniques that can be applied to computer system (software and hardware) design as well as many other domains such as telecommunication and industry automation. Formal methods are used to specify and model the behavior of a system and to mathematically verify that the system satisfies the pre-defined functional and safety properties. Formal specifications and formal reasoning provide potential for evidence or assurance for a safety case, which is usually more convincing than informal evidence, particularly when one is trying to satisfy legal constraints, or to demonstrate best practice or adherence to absolutely necessary principles. There are currently many FDTs available such as Estelle [ISO9074 89], MSC [MSC 96], SDL [Z.100 93], Promela [Gerard 91], LOTOS [ISO-L 89], etc.

Due to the inherent complexity of feature interaction problem, manual investigation techniques that deal with interactions one by one seem to be inadequate. The importance of formal specification approaches has been recognized after years of study. Many kinds of formal techniques have been adopted in the feature interaction research area (refer to [Blom 97], [Bostr 94], [Comb 94], [Conor 95], [FaciL 91], [FaciL 94], [Gamme 94], [Gupta 96], [KhenB 92], [Nakam 97], [StepL 93], [Stephen 97], [TadaY 94], etc.).

1.2 Related Work

Researchers have been using various tools to study IN and working on all feature development stages to address FI problem. We describe works related to our IN system modeling and FI detection study.

1.2.1 Modeling of IN and Features

The most important issue in formal modeling is deciding the abstraction level of the modeling. A higher level of abstraction is good for capturing the major characteristics of a system as well as for reducing the computational complexity in later works such as simulation and verifications. However, it might be too rough to describe the functional components of the system, their communications and their behaviors. On the other hand, a model that is less abstract tends to be of large size and takes longer to build. More important, while describing more detailed information, it provides less understandability and carries high computational complexity, which quickly makes the model unmanageable when the size of the system increases. Some results in modeling IN system and features are introduced below.

A group at the University of Trondheim (Norway) has developed an SDL-based Intelligent Network laboratory (INLAB) [Csurgay 94] that provides a fully functional IN system platform and feature development environment. INLAB models the IN architecture and supports feature development. Our interest, unlike theirs, is more on FI detection rather than feature development.

C.Morris and J.Nelson described a computer-aided feature development environment (CSDE) in [Conor 95]. This SDL-based approach provides a high level of abstraction and understandability. Due to the modularity of CSDE and the underlying SDL architecture, many of the components can be re-used to create new features in this environment. However, CSDE models the GFP (Global Functional Plane) and DFP (Distributed Functional Plane) as two entities in the same SDL "system" level specification, which is inappropriate from the viewpoint of ITU-T INCM.

Prof. Luigi Logrippo and his research group have been using formal language LOTOS [ISO8807] as their tool for specification and simulation of IN systems, features, and feature interaction for years. [FaciL 91] describes their results of formally specifying telephony systems in LOTOS using constraint-oriented style approach. [StepL 93] proposed a state-oriented style approach, which is similar to extended FSM (Finite State Machine), to specify features inside the telephone systems. [BoumL 93] combined the specification of both the system and the features running within the system. However I personally believe the "synchronize-on-gate" mechanism that LOTOS uses to describe parallelism and inter-component communication is not strong enough to express delayed message passing and delayed input signal processing.

Researchers also adopted other formal specification methods. [KhenB 92] described the usage of LTS (Labeled Transition Systems) and dynamic extension approach to specify the complex behavior of distributed systems such as Intelligent Networks. [TadaY 94] proposed a way to characterize features using FSM as well as STR (State Transition Rule) language to implement various features and study their behaviors. [Gupta 96] and [Blom 97] illustrated how to specify features precisely and efficiently using the Message Sequence Chart (MSC). They showed that FIs in telecommunication system can be specified and analyzed in MSC as well. [Nakam 97] employs extended Petri-Net model in probing behaviors of various features.

1.2.2 Feature Interaction Detection

Bryce Kelly et al. [Bryce 94] have developed two abstract models to describe features and detect FI using SDL. In Centrex Model, FIs are identified by discovering conflicts of the features' behaviors in the switching software of the "complete node". In Service Plane Model, potential FIs were detected by means of conflict "messages" sent to users from a set of active features as well as contradictory "feature return requests". However, the two models are too abstract to analyze many types of FIs.

In [Ohta 94], the authors proposed a way to detect and resolve FIs at different stages of feature development: specification, design, manufacturing and execution. FSM models were used to analyze the specifications and search for conflicts between features. FIs are detected by tracing for non-deterministic conflicts among the features. Their work involves detail feature behaviors and thus carries a significant computing complexity.

A group led by L.Logrippo in University of Ottawa has described their study results of FI detection. [BoumL 93] used a LOTOS specification of a sample telephony system and applied the step-by-step simulation to detect FI. [FaciL 94] developed a method based on feature composition and integration. They detect FIs by comparing features' individual behaviors and their supposed combined behaviors. [StepL 95] presents a method using backward reasoning technique in formally specified LOTOS-based IN model. And [JalelL 96] proposed to detect FI using goal-oriented-execution in a LOTOS IN model. However their methods seem to only deal with interactions between pairs of features.

[Pansy 97] utilized a State-Based Model for specifying the behavior using State Transition Machines (STM).FIs are detected by locating "control modification", "data modification", "resource contention", "reachability" and "assertion violations" in STMs that represent the combined and individual behavior of features. The major drawback is the state-explosion problem introduced due to performing reachability analysis in these STMs.

A run-time FI detection approach is introduced in [Simon 97]. In their work, feature behaviors were first monitored and "learned" during a feature testing phase. The information was then employed by the run-time FIM (Feature Interaction Manager) to determine if features are operating correctly. The problem is that the FIM could be slowing down the features' execution due to the monitoring, and FIM also might become a bottleneck of overall system reliability and performance.

Other FI detection methods also have been explored. [Frapp 97] describes a relational method for specifying feature requirements and detecting FI based on relational calculus and refinement lattice. They use inductive style specification and illustrated how feature interactions can be detected using Prolog. [Thist 97] gives out a supervisory control approach for FI modeling, detection and resolution. By viewing the development of features as modular synthesis of a collection of decentralized supervisors in network switches, they detect and resolve the feature interactions during the synthesis procedure. [Chen 97] belongs to the same category of supervisory control but employs priority functions of individual supervisors during the generation of joint control actions. [Mnakam 97] proposes a Petri-Net based detection method for non-deterministic type feature interactions. By using P-variant technique, their result indicates good detection quality and can deal with complex features like teleconference. [Roche 97] tried to address FI using Constructive Proof Systems. They suggest using the constructive logic in feature design to make sure every detail of the design specification can be satisfied, then automatically extract implementation code from the proved design, thus eliminate any possible interactions.

[Frapp 97] provides good performance when features are well specified but the FI detectors need extensive knowledge over relation calculus and detail feature behavior. [Thist 97] and [Chen 97] work well for features developed by a single manufacturer but can not deal with the actual multiple-feature-vendor situation in telecommunication industry. [Mnakam 97] only addresses non-deterministic type FIs but not the other types of FIs. The method presented in [Roche 97] has general significance, however it was weak in dealing with multiple-vendor situation. This method also will not work well if the feature specification does not cover 100% of the feature's functions and use cases.

1.3 Introduction to Our Work

Feature Interaction problem has become a major hindrance for rapid creation of new features since early 90's. However, FI research remains an on-going activity in the IN field. This thesis is mainly focused on two major issues: Modeling of IN system using Formal Description Technique (FDT) and Feature Interaction detection method.

IN System Modeling

IN architecture provides the fundamental infrastructure of the network in which various new features can be developed and utilized with minimal difficulties. However, informal descriptions of IN-structured network (such as natural language or illustration figures) are usually not concise and precise enough for technical analysis. Also they are hard to be studied and simulated using automated software tools.

We have worked out a model of IN-structured systems based on the principles of the Distributed Functional Plane (DFP) of ITU-T proposed Intelligent Network Conceptual Model. A widely used formal description language SDL is adopted in order to express an IN system efficiently with regard to both the constructional and functional aspects. During the modeling, the IN system and its functional entities are mapped into appropriate SDL concepts interconnected by communication pipes.

FI Detection Study

Our FI study is carried out on the development stage of features' life cycle and aims to improve the design and implementations of features so that as many as possible FIs can be detected and resolved at the features' development stage.

Two kinds of FI detection methods have been heavily explored in the recent years. The first type of methods adopted the "specification-oriented" idea, which aims to discover FIs by working on features' specification documentation. The second class of approaches take the "behavior-oriented" view and mainly concentrate on analyzing the features' detailed behavior and finding out if their combined behavior is consistent with their individual behaviors.

- Specification-oriented methods cannot detect FIs introduced after specification stage (e.g. an FI may be caused by a design decision to use a certain input signal that is also used by another feature). Also they have difficulty in studying the possible FIs among different instances of the same feature.

- Behavior-oriented approaches are more powerful in discovering FIs. However, substantial computational complexity is usually associated with such approaches because huge-sized Finite State Machine (FSM) or other formal models are needed to analyze the combined behavior of multiple features. The detection efficiency is low for these approaches and thus they are not suitable to detect FIs on large scale. Exploring FIs among multiple instances of the same feature is also a challenge for this kind of approaches.

We have chosen not to take either of these two paths. The focus of our FI study is searching for practical and efficient detection method based on the analysis of various major causes that lead to FIs. We categorize different kinds of possible conflicts among features and establish our own FI detection method based on searching for such conflicts. Because the amount of information needed for this FI detection strategy is much less than that of behavior-oriented approaches and no more than that of specification-oriented methods, the computational complexity we meet is favorably low.

Links between the two parts

Our IN system model in SDL provides a usable platform on which we can analyze the behavior of IN functional components and features. By using a CASE (Computer-Aided Software Engineering) tool -- ObjectGeode, we can perform simulations of various call situations and feature-using cases that are very useful for FI study. This model also helped us in performing FI cause analysis that contributed to the FI detection study.

Our FI detection method is based on a detection framework that separates the responsibilities of feature design and FI detection (refer to Chapter 5). The conflict-searching strategy used in this method is based on the FI cause analysis.

Moreover, the IN model can be used to verify the detection result of our FI detection method. The correctness of FI detection result is proved if we can reproduce the corresponding FI in a specific feature running situation in our IN model (in SDL) with the help of the software tool - ObjectGeode.

1.4 Organization of the Thesis

The remainder of the thesis is organized into the following chapters:

Chapter 2: ***Intelligent Network Fundamentals***

The origin and motivation of Intelligent Network are first reviewed. Then we introduce the basic IN concepts and architectural model that helps to grasp an overview of IN-structured system and understand the functions of major IN entities.

Chapter 3: ***Modeling of Intelligent Networks using SDL***

Based on the ITU-T standardized IN Conceptual Model, we introduce our model of IN-structured systems using SDL. This model reflects our efforts to formally specify the IN architecture, IN entities and features at the Global Functional Plane. Basic knowledge about SDL is also presented.

Chapter 4: ***Describing Feature Properties using CCM and FCA***

This chapter presents two key concepts we proposed to describe the features with respect to their associated call connection(s) and environmental assumptions. The Call Context Model and Feature Context Assumptions capture the features' most important information for FI detection purpose. The explanations of CCM and FCA are further illustrated by examples.

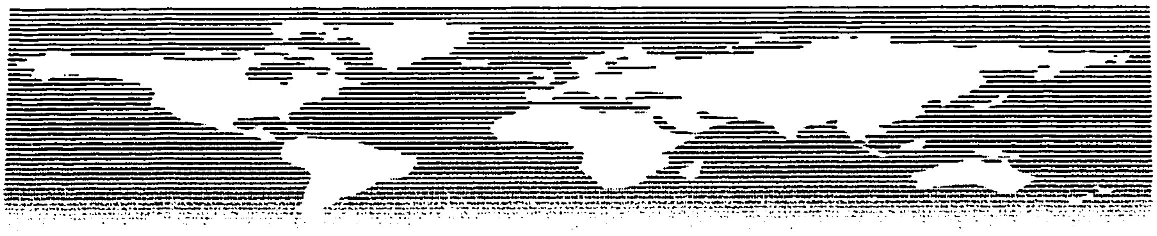
Chapter 5: ***Feature Interaction Detection***

We first introduce our FI detection framework that assumes FI detection responsibility to specialized FI detectors. Under this framework, our new detection method based on features' CCM and FCA is presented. The main strategy of this method is searching for conflicts in features' CCM-FCAs. Four concrete examples are given in this chapter as further illustration.

Chapter 6: ***Summary and Future Work***

We summarize our work and contributions as well as point out some important future directions.

Chapter 2 Intelligent Network Fundamentals



2.1 Origin of Intelligent Networks

The emerging of most new technologies has been the result of strong demands for solutions of certain inherent problem(s) that could not be solved under the existing technologies. IN is no exception. It is the growing demands for more powerful and intelligent telecommunication features that invoked the development of Intelligent Networks.

2.1.1 How the Features Emerged?

The telephone system has experienced a few generations since its first introduction. Although technology advances were achieved in almost every corner from telephone sets to transmission wire, the most important one has been the improvement of the switching systems that allows telephone communication to be pervasive in our life. The switching systems have come through four major generations and are evolving towards the fifth.

As shown in Figure 2-1, *dedicated wire connection* is the most primitive approach that only allows limited number of fixed phone connections. *Operator-performed switching* is more flexible in call connections but very inefficient. The capacities of such systems are small. *Electromechanical switching* is better than (1) and (2) but still could not meet the demand of the telephone users because of low capacity and speed. Only after the introduction of *computer-program-control* technique and *hierarchical switching* concept did the telephone system come into a mature and widely used era.

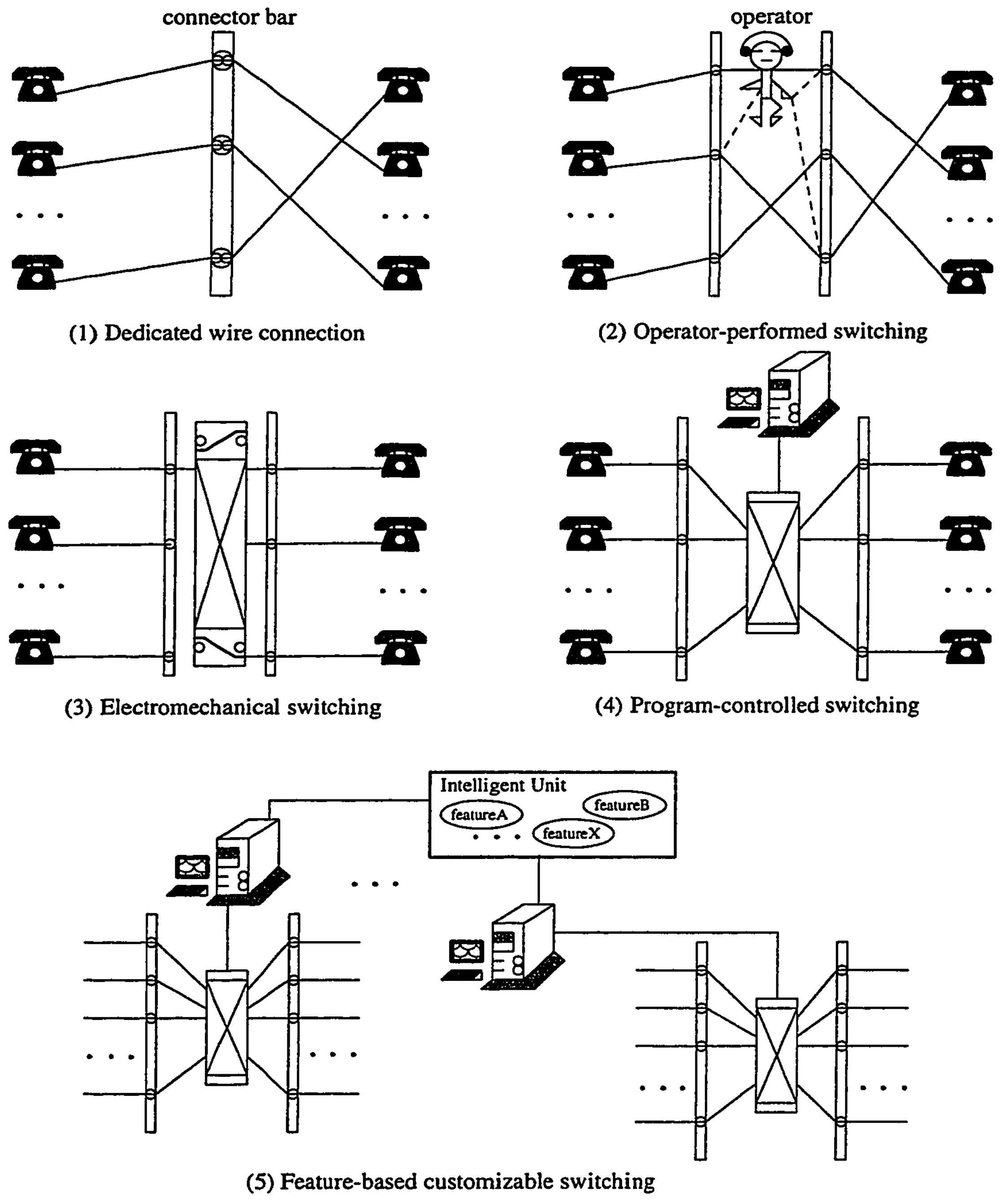

connector bar
operator
(1) Dedicated wire connection
(2) Operator-performed switching
(3) Electromechanical switching
(4) Program-controlled switching
Intelligent Unit
featureA
featureB
featureX
(5) Feature-based customizable switching

Figure 2-1: Evolving of Telephone Switching Systems

However, the style of telephone communications has hardly been changed throughout (1) to (4) stages. The reason is that all the four generations are mainly focused on how to improve the capacity, voice quality and speed of the telephone system. Little effort was made to allow users to enjoy customized communications.

For example, if you are busy writing your thesis for the defense next month, you can hang a "Do Not Disturb" sign on your door so that nobody will interrupt you unless there is a fire or earthquake. But can you do the same thing with your phone so that no in-coming call will ring to bother you unless those you really love to accept or those of emergency ?

Unfortunately, you cannot do that with even the fourth generation telephone systems, despite that they provide you with satisfactory speed and voice quality. What you now really need are powerful call functions that allow you to communicate in your own style and rules. This is exactly why the fifth generation of systems are being developed. The goal is to provide the users with the freedom of having control over their own communications that had always been controlled by the rigid switching devices for the past decades.

Intelligent features came to satisfy such demands. They are designed to partially modify the old-style behavior of switching equipment and give you the ability to customize them according to your preference (to some extent currently). For example, in the above mentioned example, you can specify under what situations the phone should ring to get your attention and under what situations it should not. This can be realized using a feature called "Terminate Call Screening (TCS)" that allows all in-coming calls to be screened based on an "authorized caller number list" given by you. Callers whose numbers are not on the list will always get a busy tone should they call you. Of course telephone numbers of your spouse, parents and boss can be put on the list, but not those of telemarketing companies. Now, equivalently you have put a "Do Not Disturb" sign on your telephone line. This is an example of what functions a feature can provide to you.

Due to technical and commercial reasons, the telephony industry usually develops and offers such advanced supplementary call functions, that are called *features*, one by one and in separate commercial units.

2.1.2 User Needs Request for IN

With the introduction and success of a few widely used features such as "1-800 free call",

"Call Waiting" and "Centrex", both the network providers and the users have been attracted by the potentials that telecommunication features can provide. Users demanded more features to satisfy their individual communication needs. This led telephony manufacturers to develop various new features in order to surpass competitors.

Existing POTS Systems

In order to develop new features, we need to have a look at the architecture of existing telecommunication networks. Computer program controlled switching technique has been used for call processing in nearly every network. Almost every task is done under the control of the sophisticated software residing in the computers that are constantly monitoring the running of various network devices.

The most important elements in such systems are a hierarchy of switches at different levels that perform call routing and relaying. Usually, there are three levels at which switches can reside: *local-exchange level*, *intermediate-exchange level*, and *transit-exchange level*, as shown in Figure 2-2.

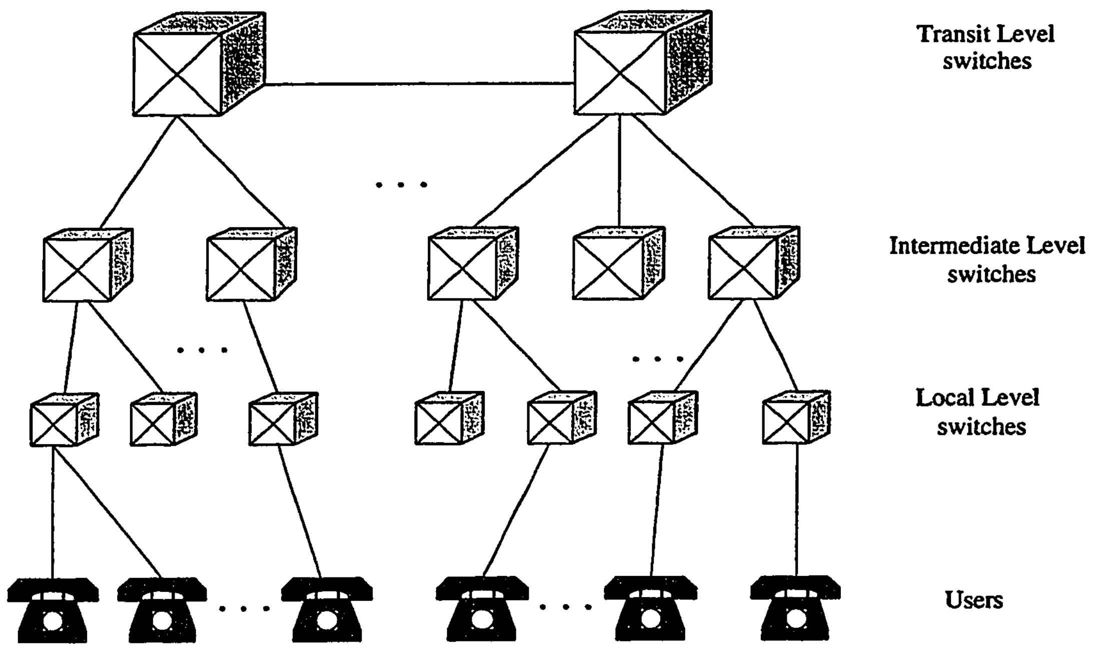


Figure 2-2: Hierarchical Switching Structure

Each switch can deal with both intra-switch and inter-switch call connections (an intra-switch call is defined as a call connection in which both parties' telephone lines are under the direct control of the same switch, and with an inter-switch call there is one party's line out of the direct control of this switch). This kind of hierarchical switching strategy works very well, even the world wide web (Internet) uses a similar mechanism for its routing.

Feature Development in POTS

Given that computer software fully controls the behavior of the processing devices and the call functions provided to the users, it is naturally that the development of new features needs to be based on existing call processing software. That is why many early features are technically implemented as software "patches" attached or embedded into the traditional Basic Call Processing (BCP) software in the switches. That is why those features are called *"switch-based features"*. At this stage, each feature is designed as an independent software entity that modifies the basic call processing inside the switch devices. These features are usually non-reusable software programs that can neither make use of one another nor be combined to construct larger features.

At first, these features are located at the transit-exchange-level in the hope that only a small number of highest level switches need to be re-designed when a new feature is introduced, the case is shown in Figure 2-3a. However due to the excess amount of overhead in accessing a feature, which incurred by the number of switches and related trunks that need to be involved along the path from a telephone user to the transit exchange switch and vice versa, the features had to be migrated to lower levels of the switching hierarchy in order to reduce the overhead for feature-using.

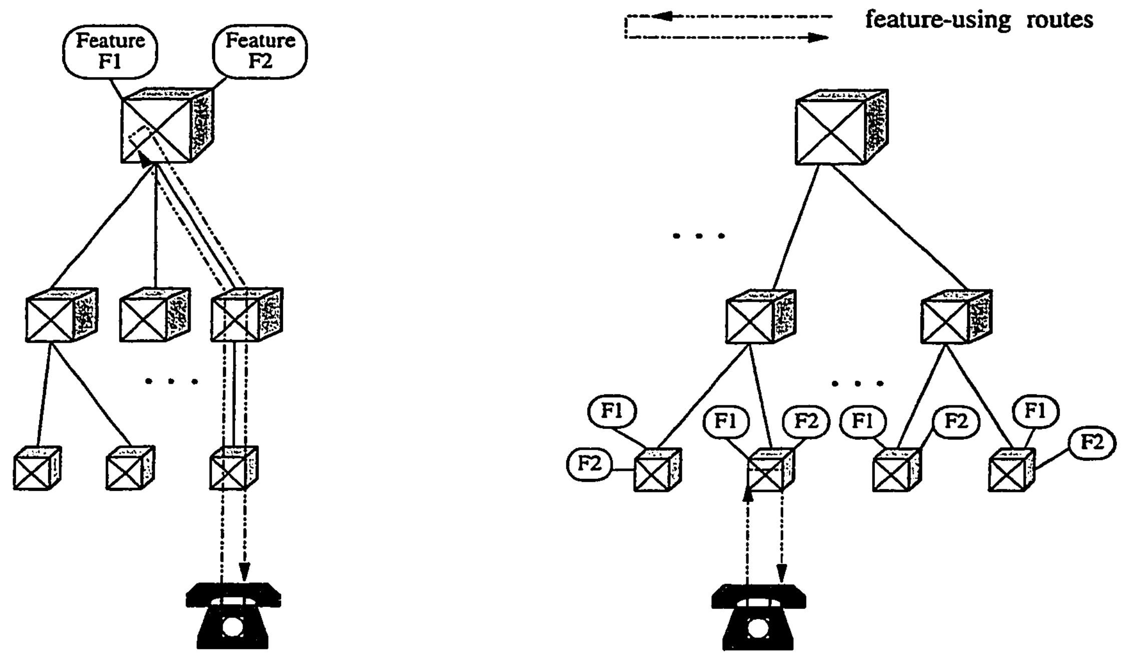
More recently, most features are designed to reside in each local exchange level switch to speed up the execution of features (as shown in Figure 2-3b). However this means the software units of every feature must be loaded into every local switch before the feature can be used. This task could take a relatively long time to complete because of the large number of local switches in the networks.

IN Solves Problems

Shortly after the introduction of first several features in the telecommunication system, critical problems arise and prevented the fast and effective creation and deployment (make it ready for network-wide user utilization) of more features.

- **Problem 1:** The inclusion of slightly more features made call switching-and-processing software much more complicated and thus hard for testing and maintenance, and the designers struggled to meet the quality and performance requirements of these new features.

- **Problem 2:** Adding new features into the system has been proved to be tough because it incurred massive amount of work to modify or upgrade large number of network switching-and-processing devices in the network. The time it takes to complete the deployment of the feature is also too long to be acceptable.
- **Problem 3:** In a multi-platform, multi-vendor environment of telecommunication networks, it is quite difficult to achieve the universal support from different networks run by different companies. Compatibility and portability of features can hardly be achieved and maintained.



a. Features reside in transit switch b. Features hooked on local swiches

Figure 2-3: Features' Location in POTS Systems

Facing above problems, ITU-T proposed its Intelligent Network (IN) as the skeleton of future telecommunication networks to facilitate the fast and cost-effective development and deployment of large number of features. From an intuitive point of view, IN architecture addresses these problems using the following mechanisms:

- **Separate Network Intelligence from BCP:** The functions of Basic Call Processing are separated from the processing of features in the IN systems. Specialized intelligent units are introduced to deal with large number of co-existing features. The normal switching-and-processing devices are liberated from the burden of doing that and only manipulate normal calls with a little extra task that is to check for conditions or signals that may trigger a feature into execution. This idea solves problem 1.

- **Non-Distributed Intelligence:** Feature inclusion used to be tough because we have to modify software in large number of switching-and-processing devices. Under the new architecture, addition of a new feature only means to update the software inside a few intelligent network units, and send a message (that contains a few feature-triggering conditions and/or signals) to each switching device asking them to check for a few more standard conditions/signals. Thus greatly reduced the time needed to deploy a new feature, which deals with problem 2.

- **Standard Functional Entities and SIBs:** IN architecture consists of a set of standard functional entities that have well-defined interfaces and signaling protocols. Also the distinctions between different entities are clear and independent of the network product vendors. Furthermore, every feature inside the IN system will be developed by using standard Service Independent Building-blocks (SIBs) that are essentially platform-independent call processing actions/routines. These SIBs provide identical network call functions even they may be implemented differently by different vendors. This addressed problem 3.

2.1.3 Key Technologies that enable IN

Two important telecommunication technologies have been mature and applied into actual networks by the time IN concepts are proposed. They are *Common Channel Signaling System* (CCSS) and *Non-Switching Node* (NSN). Based on CCSS and NSN, Intelligent Network systems have become technically feasible to be put into practical industry development.

Common Channel Signaling System

In POTS style telephone systems, the actual data (voice, image, etc.) and system signaling (information of how to transfer the actual data) are transmitted through the same transmission channel over the media. Thus the data and signaling messages are mixed onto the same channel after being sent on their ways. Such a signaling mechanism is called In-Band-Signaling (IBS). Networks using IBS usually have lower efficiency because critical signaling messages cannot be delivered quickly enough, and sometimes even not reliably (in case of a congestion and data discarding).

CCSS, which is also called Out-Band-Signaling, gives high priorities to system signaling messages over normal data. The main idea is to set aside a separate transmission channel to send and receive signaling messages, as shown in Figure 2-4. The signaling channel is reserved for exchanging relatively small-sized system messages and thus allows the network devices to communicate each other more efficiently. Network performance is also improved because it is possible to promptly stop transmitting unnecessary data. e.g. almost everybody surfs the Internet, try to recall how many times you have pressed the "stop" button or changed to another web page when the current one is still busy loading.

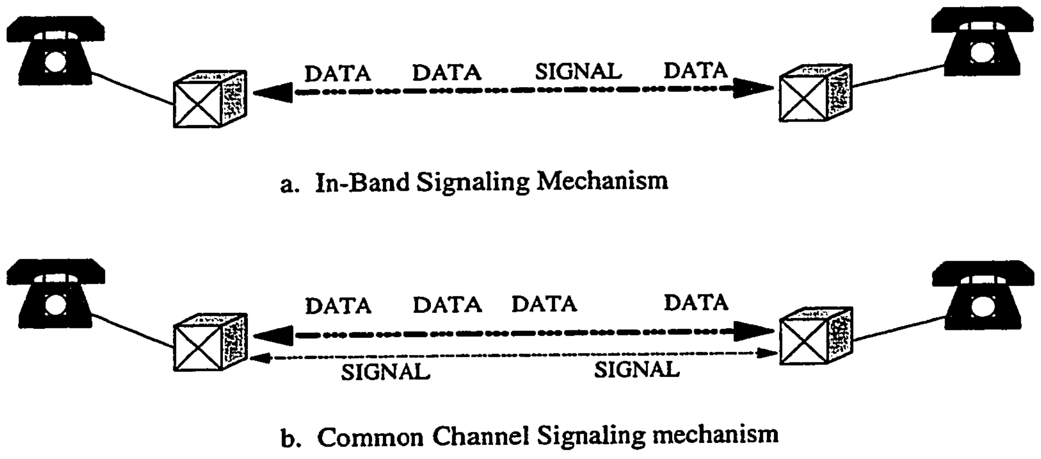

DATA
DATA
SIGNAL
DATA
a. In-Band Signaling Mechanism
DATA
DATA
DATA
DATA
SIGNAL
SIGNAL
b. Common Channel Signaling mechanism

Figure 2-4: Mechanism of Common Channel Signaling

There has been two standard CCSSs defined by CCITT (the former name of ITU). The CCSS #6 uses analog voice channel for signaling message transmission. And CCSS #7 utilizes a standard 64 kbps digital channel. Since digital signal is more suitable for computer processing, CCSS #7 is currently the most widely used CCSS in telecommunication networks, and it is also referred to as "SS 7" in technical literature.

Intelligent Network could be considered as impractical without the technology support from Common Channel Signaling System. The reason lies in that the intelligence units must rely on CCSS to efficiently communicate with basic call switching-and-processing devices in order to execute features correctly. An In-Band-Signaling system can never satisfy the signaling performance required by feature execution, given the fact that the intelligent units of IN systems are geographically located far away from the switching nodes and the feature users.

Non-Switching Node

Non-switching nodes are where the control software and data of features reside. They serve as the intelligent units in the network and act as the actual feature executors that manage the concurrent feature instances (possibly large number of them). Non-Switching Nodes usually do not involve in the BCP functions and actual data transmission. That is why they are given the name. There is no data transmission channel between NSNs and switching devices (in switching nodes), only the signaling channels.

Two kinds of NSNs exist in telecommunication networks: Processing NSNs and Data NSNs. They are meant to improve the function capability of existing networks in both the processing and data aspects without incurring massive modifications to basic call switching-and-processing devices.

- Processing NSNs provide extra feature processing ability, which forms the network intelligence.
- Data NSNs offers vast data storage capacity and global data manipulation functions.

Although both kinds of NSNs are running in a relatively independent way outside the normal switching nodes, the functional support and call instance data support from switching nodes are indispensable for NSNs to work correctly. Usually NSNs maintain their communication with switching nodes using CCSS #7 as signaling protocol.

To summarize, we are clear that Common Channel Signaling System and Non-Switching Nodes are essential elements that have formed the basis for the introduction of Intelligent Network systems [Visser 95].

2.2 IN Functional Requirements

Supported by CCSS and NSN technologies, it is possible for different companies to work on the same telecommunication network and running their businesses in different kinds of aspects. For example, a network provider may only manage the running of BCP network, and other feature-providing companies offer the advanced processing functions. Network devices operated by different companies can communicate effectively via CCSS. Adding a non-switching node to offer a self-designed feature is much easier than before because you will not ask to modify the switching devices operated by other companies.

The functional requirements of IN systems come from two aspects: *Customer Needs* and *IN Provider Needs*. Requirements from IN customers will assist in identifying specific functions that will be offered to customers. Requirements come from IN provider's demand for network abilities ranging from establishing, operating and maintaining the network capabilities in order to allow IN customers to work correctly in the IN systems. Figure 2-5 gives a view of requirements from the two sides.

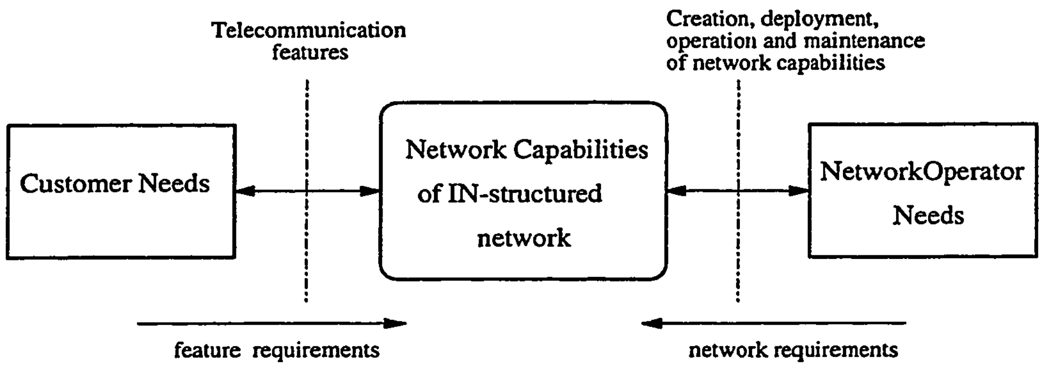

Telecommunication features
Creation, deployment, operation and maintenance of network capabilities
Customer Needs
Network Capabilities of IN-structured network
NetworkOperator Needs
feature requirements
network requirements

Figure 2-5: IN Customer Needs vs. IN Provider Needs [Q.1201]

Here the term "IN customer" refers to an organization who works in IN systems to provide features to end users. This organization serves as the provider of IN features but is the customer of the IN networks. The term "IN provider" represents an organization that operates an IN system. An IN provider offers a telecommunication framework that allows various IN customers to work in so that they can provide their own features to end users (feature subscribers).

IN customer Requirements

Major requirements from IN customers when defining the IN architecture [Q.1201] are:

- Being convenient and efficient to define and introduce new features into the system, also maintain and upgrade them.

- Capable of activating a feature both on a call-by-call basis and for a period of time. For the second case, deactivating the feature may be needed at the end of the period if no renewal is desired.
- The ability to provide and exchange network information and call specific data. Supports for recording feature usage and related information in the network such as tests, performance, statistical data and charging are also needed.
- Able to deal with complex features that involve two or more parties and retain control over different invocation of the same feature.
- Features should be accessible through normal network terminal devices. It should also be possible to provide and access features that need to use functions in multiple interconnected networks.

Specific requirements of IN customers are divided into several aspects: Feature Creation, Feature Management (in both deploy stage and utilization stage), Feature Processing, and Feature Interworking. We do not elaborate on this issue because our focus is not IN system design but FI detection. More information is in ITU-T recommendation [Q.1201].

IN provider Requirements

IN providers also have their own requirements of IN architecture. In general, the following reflects their overall expectations:

- Be able to cost-effectively migrate from existing networks to the target IN systems in a practical and flexible manner.
- Redundancies among network functions in physical entities can be reduced, and allow for flexible allocation of network functions to physical entities.
- Need a well-designed communication protocol that allows for efficient signaling and the flexibility in the allocation of functions.
- Network integrity can be guaranteed despite the introduction of varieties of telecommunication features.
- Capable of managing network elements and resources so that the quality and performance of features can be guaranteed.

Same as IN customer requirements, IN provider requirements are also specified in several aspects: Feature Creation, Feature Management, Feature Processing, and Feature Interworking. ITU-T Q.1201 recommendation provides the official guidelines of IN systems and describes these issues in detail.

2.3 IN Conceptual Model (INCM)

IN has been proposed to solve critical problems met in telecommunication networks such as those three we mentioned in section 2.1.2. ITU-T describes the Intelligent Network Conceptual Model (INCM) as the formal architectural framework for the study and development of both current and future IN systems. INCM defines important concepts and technologies in various aspects of an IN system.

2.3.1 Functional Plane Organization

The essence of INCM is the definition of a universal framework that captures the skeleton of IN technology. Although the specific architecture, signaling system, entity design and implementation may evolve with time, the INCM is meant to remain consistent and theoretically applicable to every IN system.

According to ITU-T Q.12xx series recommendation, INCM is intended to represent an integrated and formal framework in which various "concepts" and "models" are identified, characterized and related. To achieve this, the INCM is designed to consist of four "*planes*", where each plane represents a different level of abstract view of the capabilities provided by an IN-structured network. These views address feature-using aspect, global functionality, distributed functionality and physical aspect of an IN system.

- **Service Plane**

  It represents an exclusive feature-oriented view of the IN system. At this plane, we actually talk about the use of features but not how the IN system realizes them.

- **Global Functional Plane**

  GFP models an IN-structured network as a single entity concerning how the features are globally represented, how they are interfaced with basic call functions, and how they are executed.

- **Distributed Functional Plane**

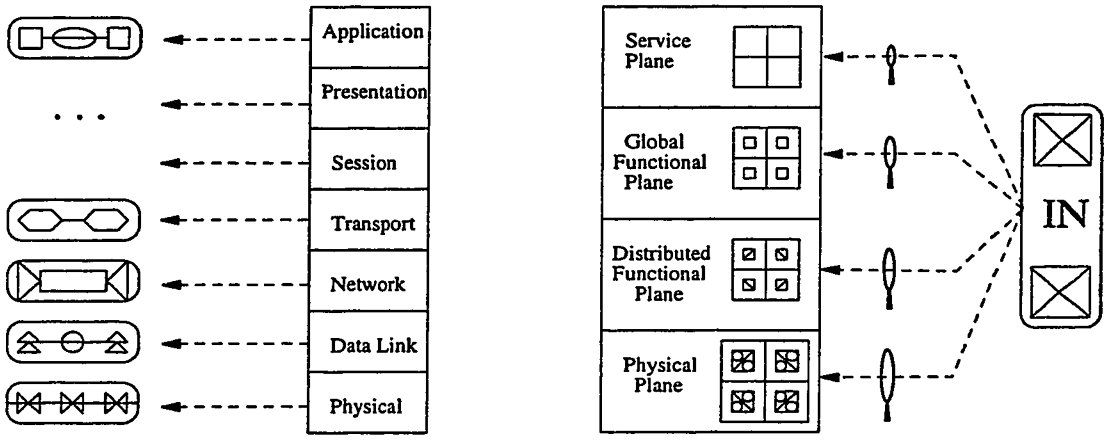
  DFP reflects a view of IN system that consists of various distributed functional entities, and how they cooperate to realize the global functions specified in GFP.

- **Physical Plane**

  This plane describes the principles of physical devices that make up the IN system and how they are organized to satisfy all the IN functional requirements.

These four planes represent distinctive descriptions of the same IN system, only that at different abstraction levels, or we say they specify IN system using different granularity. We usually work at one specific stage without having to know how things are done in another plane unless we are working on the mapping of two adjacent planes.

It is very important NOT to think that INCM is analogous to the OSI (Open System Interconnection) seven-layer model. They are different concepts. OSI seven-layer model describes a stack of layers that are essentially individual entities, and each of them performs part of the communication tasks. While all four INCM planes refer to the same entity -- IN system, they describe it in different ways and at different abstraction levels. Figure 2-6 tries to give you an idea of their distinction. IN looks different at the four INCM planes because different "magnify glasses" are used to study the same object -- IN. However, each layer in OSI network model corresponds to different actual entities.



OSI 7-Layer Structure

INCM 4-Plane Structure

Figure 2-6: OSI 7-Layer Structure vs. INCM 4-Plane Concept

2.3.2 Service Plane (SP)

This plane is of the interests of feature users and feature vendors. In this plane, an IN is described as a magic box. It contains various features that are ready to be subscribed or be used on a call-by-call basis. It describes features and sub-features (see below) from users' perspective. People only discuss what functions that each feature has to offer, how the users benefit from using them, and how they are organized using sub-features.

Sub-features are re-usable, atomic construction components of features. Different features may be using the same sub-feature in different ways. Some sub-features are indispensable parts of some features, others are optional and may be added or removed when desired. A feature may contain one or more sub-features. Only the external behavior of the features and sub-features are observable by feature users at this plane. There is no information on this plane that touches the actual design or implementation issues.

2.3.3 Global Functional Plane (GFP)

The GFP models the IN system functionality from a global perspective and considers the whole IN system as one single entity rather than a set of distributed functional entities, which is the actual case. Under this plane, features and sub-features are redefined in terms of the global network functions that support them. In SP, the sub-features are the smallest components. We will discuss how to construct and realize them using concepts on GFP, which is a lower level of abstraction of IN systems.

A major idea of IN is to separate basic call processing functions from newly introduced network intelligence that is composed of various features. At the GFP we will begin to indicate this idea when we study how the GFP describes the IN system. From a global viewpoint, every feature is based on sub-features, and each sub-feature is supported by three kinds of GFP functional elements:

- BCP (basic call processing), which deals with normal call initiation, switching, maintaining and termination.
- Intelligent logic related to fulfilling the extra call functions designated to this sub-feature.
- Interfaces between BCP and intelligent logic, which allow the passing of control power and supporting data.

A sub-feature is represented at GFP by a so-called Global Service Logic (GSL) that specifies how this sub-feature is organized and constructed. The GSL contains information in a few aspects described using concepts like BCP, SIBs, chain of SIBs, POI and POR. These concepts are explained below.

Basic Call Processing (BCP)

BCP is a functional element that is capable of setting up and managing a normal two-party call connection in the telecommunication networks including IN. No matter if there is any feature involved in a call, BCP is always needed. Thus BCP is a never absent part in the GFP. It cooperates with Global Service Logic, which represents the sub-feature, on two kinds of points: POI and POR(s). The switching-and-processing ability of BCP in IN is the same as that in the existing POTS networks.

Service Independent Building-blocks (SIBs)

SIBs are standard, re-usable network-wide capabilities residing in GFP and are used to construct sub-features and in turn features. Each SIB performs a standard call processing or call control function that can be combined together to realize large functions. Although the SIBs may have different designs and implementations based on different implementations of lower plane entities, they are not visible to the sub-feature designer who creates features using these SIBs.

As shown in Figure 2-7, there is one and only one logic start associated with a SIB. This is where the SIB should start executing. A SIB may have one or more logic ends that present a set of possible exits after the execution of the SIB. The choice of which logic end to exit depends on the data parameters given to the SIB when invoking it into execution as well as the call situation that may turn out different results. We can see in the figure that two kinds of data are needed during the execution of a SIB: Service Support Data (SSD) and Call Instance Data (CID).

SSD provides information that is needed to support the task of this feature, and SSD is common for all invocations (different instances) of the same feature. CID defines data that can be determined only when a feature instance is actually invoked by a specific call. The CID has a dynamic nature and varies from instance to instance even they are of the same feature.

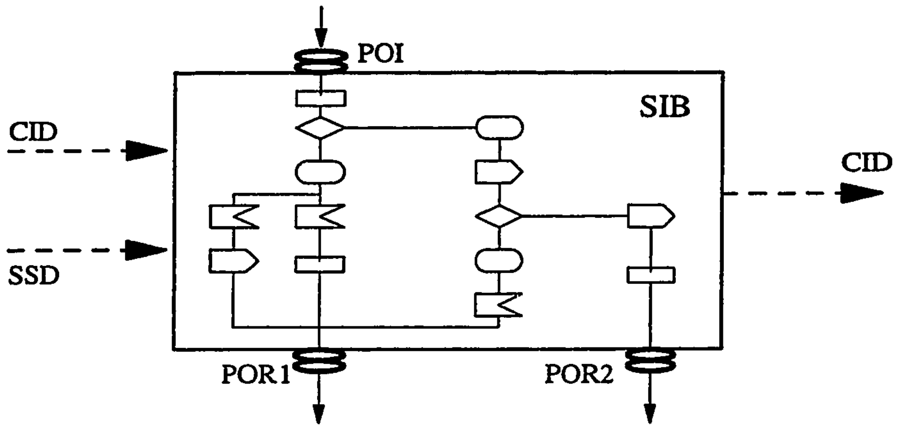


Figure 2-7: Illustration of an SIB

In the first standardized IN development phase (referred to as CS 1), ITU-T has proposed and defined 13 general purpose SIBs that can be used to construct various features, although their capabilities seem far from enough. Examples of SIBs include *the User Interaction SIB* that allows the exchange of information between a call party and the IN system (specifically the running feature instances inside the IN system), and the *Translate SIB* that provide the ability to translate a dialed telephone number into another destination directory number based on the given parameters when this SIB is invoked into execution.

POI and POR

Point Of Initiation and Point Of Return are the connection joints of BCP and the intelligent logic (i.e., the GSL) that constitutes the sub-feature. POI is a call processing breakpoint where the BCP transfers the power of control to a SIB and thus starts the execution of the sub-feature's intelligent logic. On the contrary, the POR is the call processing breakpoint that the BCP should continue with when the intelligent logic ends its execution and switchs back the power of control to BCP.

POI and POR are relatively independent from the GSLs that define the behavior of sub-features. Different sub-features may have the same POI and POR but completely distinct GSLs. However, once the GSL of a sub-feature is decided, the POI and POR are also determined in order to match the GSL and fulfill the sub-feature's tasks correctly.

Global Service Logic (GSL)

GSL is the only element in GFP that depends on specific sub-features. GSL defines what the sub-feature's behavior will be, and describes how it is organized by chaining one or more SIBs together in a meaningful way to form a combined logic. GSL of a sub-feature also uniquely determines the POI and POR(s) that will be used for this GSL.

Because the SIBs are feature independent components, they have no knowledge about the previous or next SIB that connected to it. It is the responsibility of the GSL that guarantees the chaining of the SIBs is correct and can accomplish the sub-feature's functions. Figure 2-8 shows an example of a GSL.

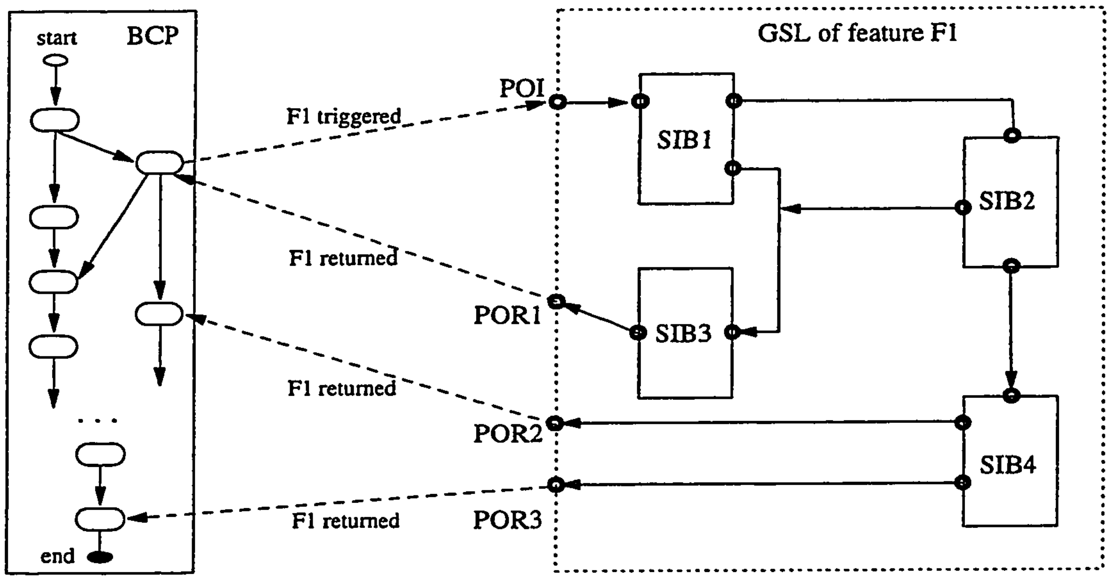


Figure 2-8: An example GSL in GFP

2.3.4 Distributed Functional Plane (DFP)

As its name implies, DFP represents a distributed view of an IN system. At this abstraction level, IN functions are seen to be performed by a group of network-wide distributed functional entities.

DFP provides a functional architecture that models all the supported functions in an IN-structured network. It identifies the specific elements (i.e., Functional Entities) and the relationship among them that are necessary to allow IN functions to be realized. DFP offers the flexibility of supporting a large number of telecommunication features and convenient evolution of IN by organizing the functional capabilities in an open-ended and modular structure. What also important is that DFP is vendor independent and implementation independent, which means universal compatibility of IN functional entities and interfaces.

Functional Entities of DFP

One functional entity (FE) is a unique group of IN functions in a single location. Each FE is part of the complete set of functions provide by DFP to support the realization of the GFP elements. Different FEs may be distributed in a number of geographically separated sites. Communications among FEs are carried out by means of the "Information Flows (IF)" using signaling data pipes.

According to ITU-T Q.1204 recommendation, the following FEs are defined in the distributed functional plane model:

FE Types	List of FE (s)
Management Related	SMF, SMAF, SCEF
Feature Control & Data	SCF, SDF
IN Specialized Resources	SRF
Call Control & Feature Trigger	CCAF, CCF, SSF

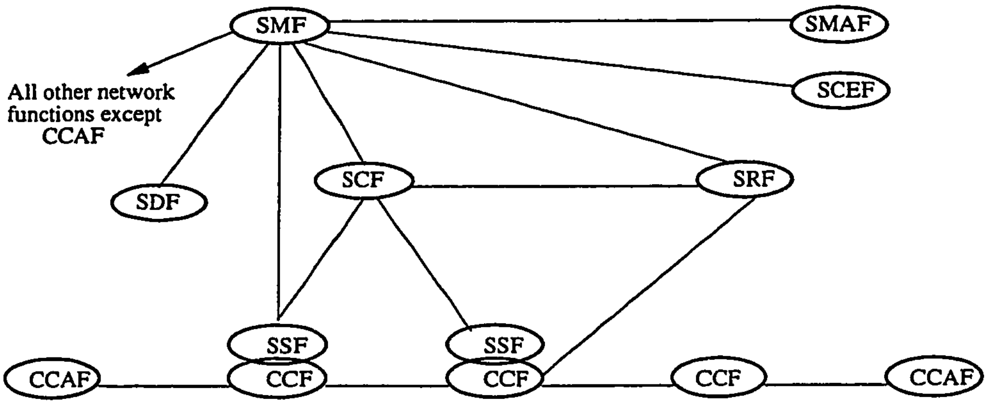
Table 2-1: Functional Entities defined on DFP

A brief description of these FEs is as follows:

- **SMF** allows the deployment and provision of various features. It also manages the IN feature related information in SRF, SSF and CCF. SMAF provides the network operator with an interface to access SMF and perform management operations. SCEF provides the environment to create new IN features efficiently.
- **SCF :** The Service Control Function is responsible for the call control and IN feature processing. The SCF may interact with other IN entities to access additional logic or to obtain supporting data required to process IN features.
- **SDF :** The Service Data Function contains customer and network data for real-time access by the SCF in the execution of IN feature instances.

- **SRF** provides special resources or devices (such as speech recognizer, voice announcer, multi-end call bridge, etc.) to certain kinds of features that need them.
- **CCAF** : The Call Control Agent Function provides access for users. It is the interface between users and the networks.
- **CCF** : Call Control Function provides call control capabilities, among which include establishing, manipulating and releasing calls. It also supports feature-triggering mechanism that enables feature users to access IN functionality.
- **SSF** : The Service Switching Function, usually coupled with the CCF, provides a set of functions required for signaling between the CCF and a service control function (SCF). SSF mainly deals with the triggering checking of activated features associated with the users.

The relationships between DFP functional entities are depicted in Figure 2-9.



CCAF	Call Control Agent Function	SMAF	Service Management Access Function
CCF	Call Control Function	SMAF	Service Management Function
SCEF	Service Creation Environment Function	SRF	Specialized Resource Function
SCF	Service Control Function	SSF	Service SwitchingFunction
SDF	Service Data Function		

Figure 2-9: Functional Model of DFP

Basic Call State Model (BCSM)

As the core content of CCF and SSF, BCSM covers (from the perspective of distributed network components) how normal call processing functions are achieved as well as how the feature logic cooperates with the basic call processing units. BCSM is the DFP correspondent of the element BCP in GFP. *A clear comprehension of BCSM is very important for understanding the IN FEs and the working mechanism of IN features at the DFP level,* because all the features are designed to work on the back of BCSM.

The initiation, switching and maintaining of normal call connections in IN are described using IN call connection model. The major elements defined in this model are users, local exchange nodes, O-BCSM, T-BCSM and call connections.

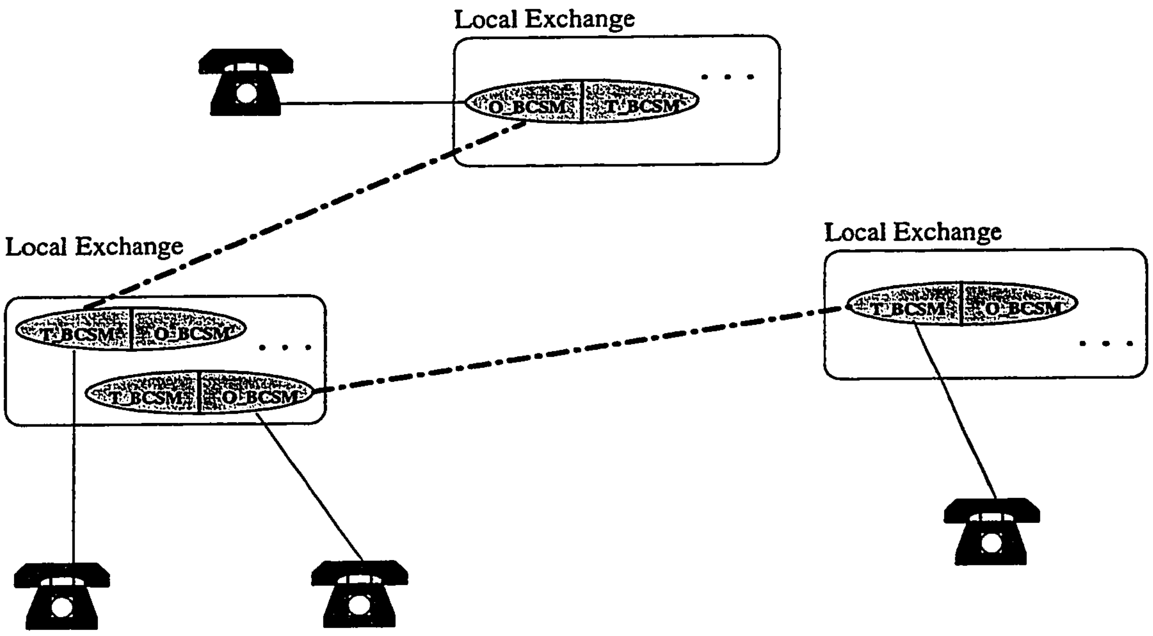


Figure 2-10: Intelligent Network Call Model

The call connection model of IN system is roughly depicted in Figure 2-10. The control logic for processing a normal two-party-call resides in the local switch that serves the user's line. The logic consists of two parts: Originating BCSM (O-BCSM) that offers the ability for the user to initiate calls and Terminating BCSM (T-BCSM) that supports the processing of in-coming calls. They are usually designed as two separate sets of processing logic.

The term BCSM refers to both O-BCSM and T-BCSM. It is a Finite State Machine description of CCF activities required to establish and maintain communication paths for users. However, because some aspects of these activities are not visible by the IN feature instances, BCSM is primarily an explanatory tool to give a representation of visible aspects of basic call processing that are of interest to features.

Important elements in BCSM include: Point In Call (PICs), Detection Point (DPs), transitions, and events. PICs are call states during the processing of a call. Each PIC represents a specific stage of this call that is of interest of the features. DPs indicate points in the call processing where the transfer of control can occur. Transfer of control means the transfer of control power of processing the call between CCF and the feature instances in SCF. Transitions are the migration actions in the BCSM from one PIC to another. Events are signals or conditions that enable the transitions.

Figure 2-11 and Figure 2-12 illustrate examples of O-BCSM and T-BCSM.

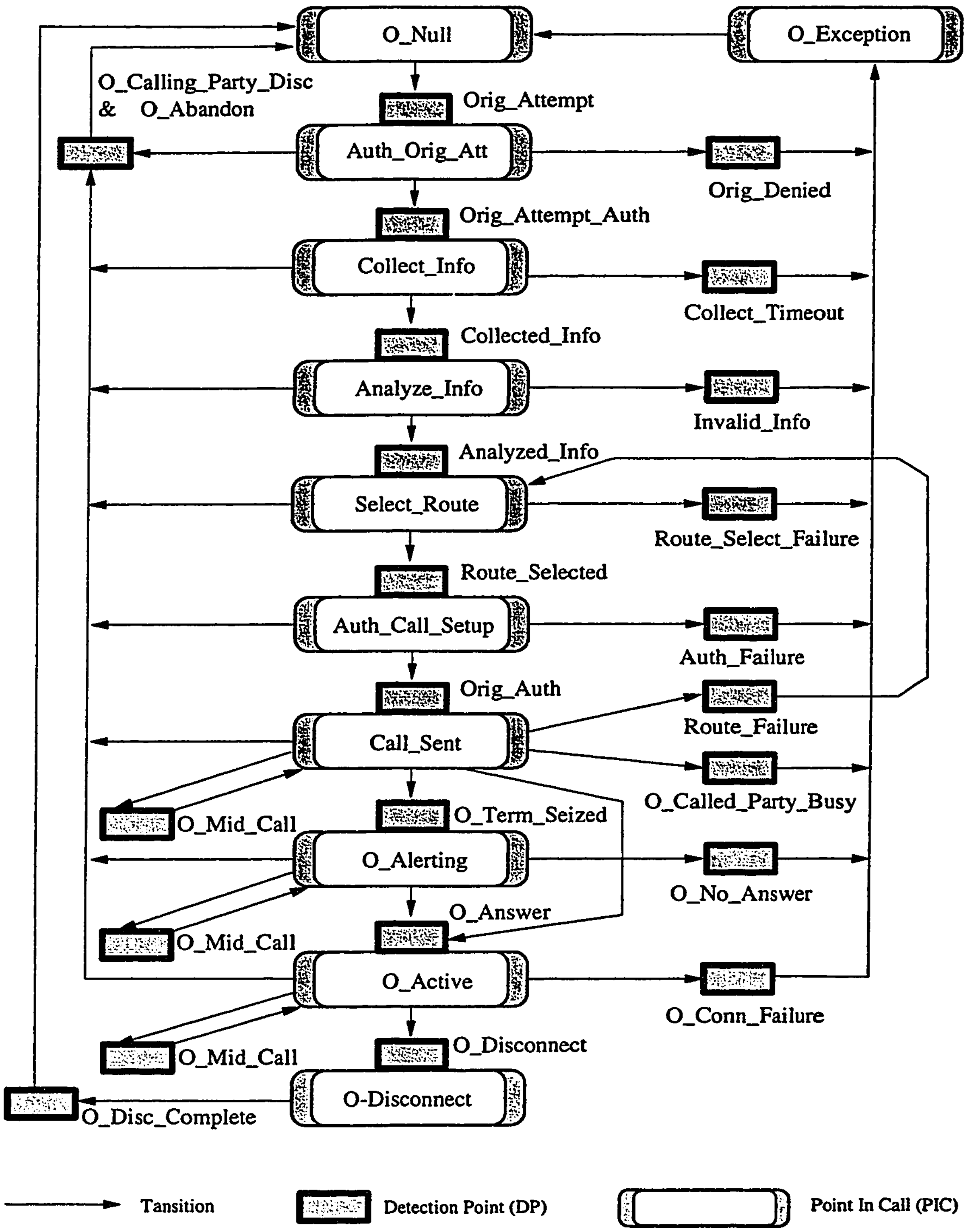


Figure 2-11: Example of Originating BCSM [Q.1204]

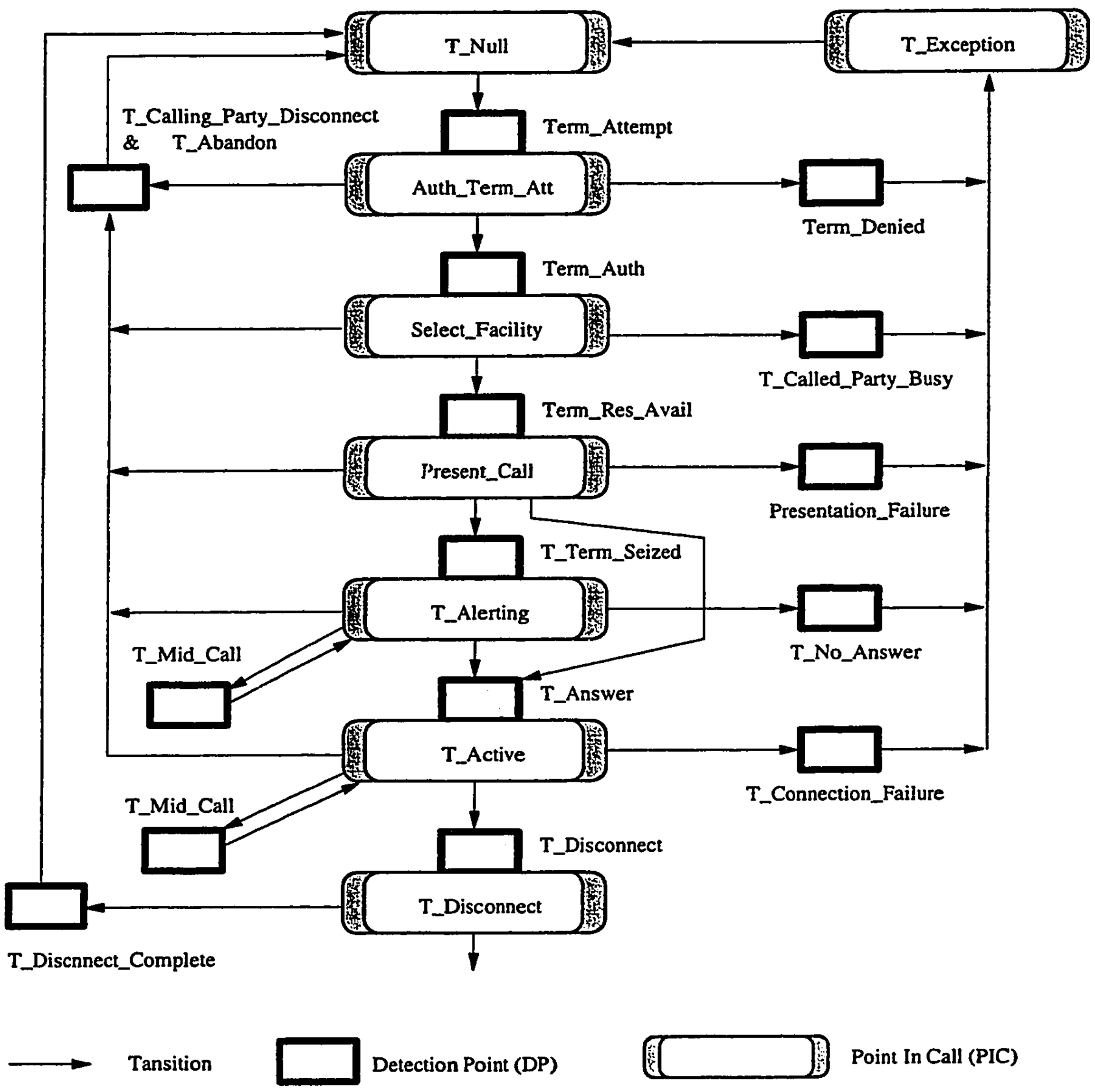


Figure 2-12: Example of Terminating BCSM [Q.1204]

Feature Working Mechanism

During the normal call processing, an IN feature is triggered when certain pre-defined conditions are satisfied. The BCSM suspends the processing of the current call and hands over the power of call control to the feature instance inside SCF. The PIC where the BCSM gives its control of processing to feature instance is POI. Upon getting the call control power, that feature instance will perform its task and be fully responsible for processing the call in this period. When the feature instance terminates, the call processing control will switch back to BCSM. However, the feature instance can ask the BCSM to resume call processing at a specific PIC or even DP. The specified PIC may or may not be the next PIC of POI that BCSM would go if there were no feature-triggering.

2.3.5 Physical Plane (PP)

The physical plane describes the physical aspect of IN-structured network. The PP identifies the different physical entities (PEs) and protocols that may exist in real IN systems. Also physical architecture alternatives and implementation techniques are chosen at this plane.

PP is the plane that explains how each of the FEAs (Functional Entity Actions) is allocated to a PE or a set of PEs, and how these FEAs are realized by specific and concrete physical entities. The PP also covers how these entities cooperate to get those actions done.

In order to support all the IN functionality on the physical plane, various PEs have been defined. Table 2-2 lists the physical entities defined by ITU-T in [Q.1205].

PE	Functional Components	Main Responsibility
SSP	CCF, SSF, optionally CCAF	SSP performs the switching and call control functionality and makes IN capabilities available to network users.
SCP	SCF, and optionally SDF	SCP is where the IN feature programs reside. It provides various IN features for user access. From the viewpoint of feature execution, SCP is the entity that co-ordinates other physical entities to fulfill the feature's tasks.
SDP	SDF	It provides the data used by feature programs throughout their running. Both SCP and SMP can access SDP. Different SDPs can access data in one another.
IP	SRF	The IP offers specialized call devices for customization of features and supports flexible information interactions between a user and the network. Functionally the IP manages the usage of various resources.
AD	Same as SCP	The Adjunct PE is functionally equivalent to an SCP (i.e., contains the same functional entities), however it is directly connected to SSP rather than remotely via a signaling network.

SN	Same as SCP	It can control IN features and engage in flexible information interaction with users. SN is similar to SCP and AD except that it has one direct point-to-point connection with each of the SSPs with which it works.
SSCP	SCF, SDF, CCF, SSF and CCAF.	The combination of an SCP and an SSP and provides the same functionality as them, designed for situations where SCP and SSP are tightly coupled and proprietary.
SMP	SMF	Provides feature management, feature provision and feature deployment control, etc.
SCEP	SCEF	It is used to define, develop, test an IN feature and input it into SMP.
SMAP	SMAF	Provides access tools to special users that manipulate SMP to perform feature management.

AD	Adjunct	IP	Intelligent Peripheral
SCEP	Service Creation Environment Point	SCP	Service Control Point
SDP	Service Data Point	SMAP	Service Management Access Point
SMP	Service Management Point	SN	Service Node
SSP	Service Switch Point	SSCP	Service Switch & Control Point

Table 2-2: Physical Entities defined in [Q.1205]

2.3.6 Mapping of Planes in INCM

Either from the perspective of functional supporting relationship or that of the development cooperation between INCM planes, the mappings between adjacent INCM planes are necessary and important. Mapping means match entities on one plane onto entities of another plane so that every entity at a higher plane can be realized by the entities on the lower plane. In INCM, from high to low, planes are arranged like this: Service Plane, Global Functional Plane, Distributed Functional Plane, and Physical Plane.

From SP to GFP

The atomic construction components of service plane are the sub-features that constitute the features. Sub-features are mapped to GSLs in GFP that consists of a chain of SIBs and matching POI and POR(s). This is depicted in Figure 2-13.

From GFP to DFP

The basic components in GFP are various SIBs. The corresponding entities that realize a SIB is a sequence of FEAs performed by FEs. Some of the FEAs may result in Information Flows (IF) between FEs. (e.g., a data request from SCF to SDF). The BCP in

GFP is mapped to BCSM in DFP. And POIs and PORs are mapped to DPs and PICs in a BCSM Finite State Machine. Please refer to Figure 2-13.

From DFP to PP

Functional entities in DFP are mapped to specific physical entities (PEs) in PP. However usually one or more FEs may correspond to the same PE, and each FE is not convenient to be split among more than one PE. Figure 2-13 illustrates the relationships.

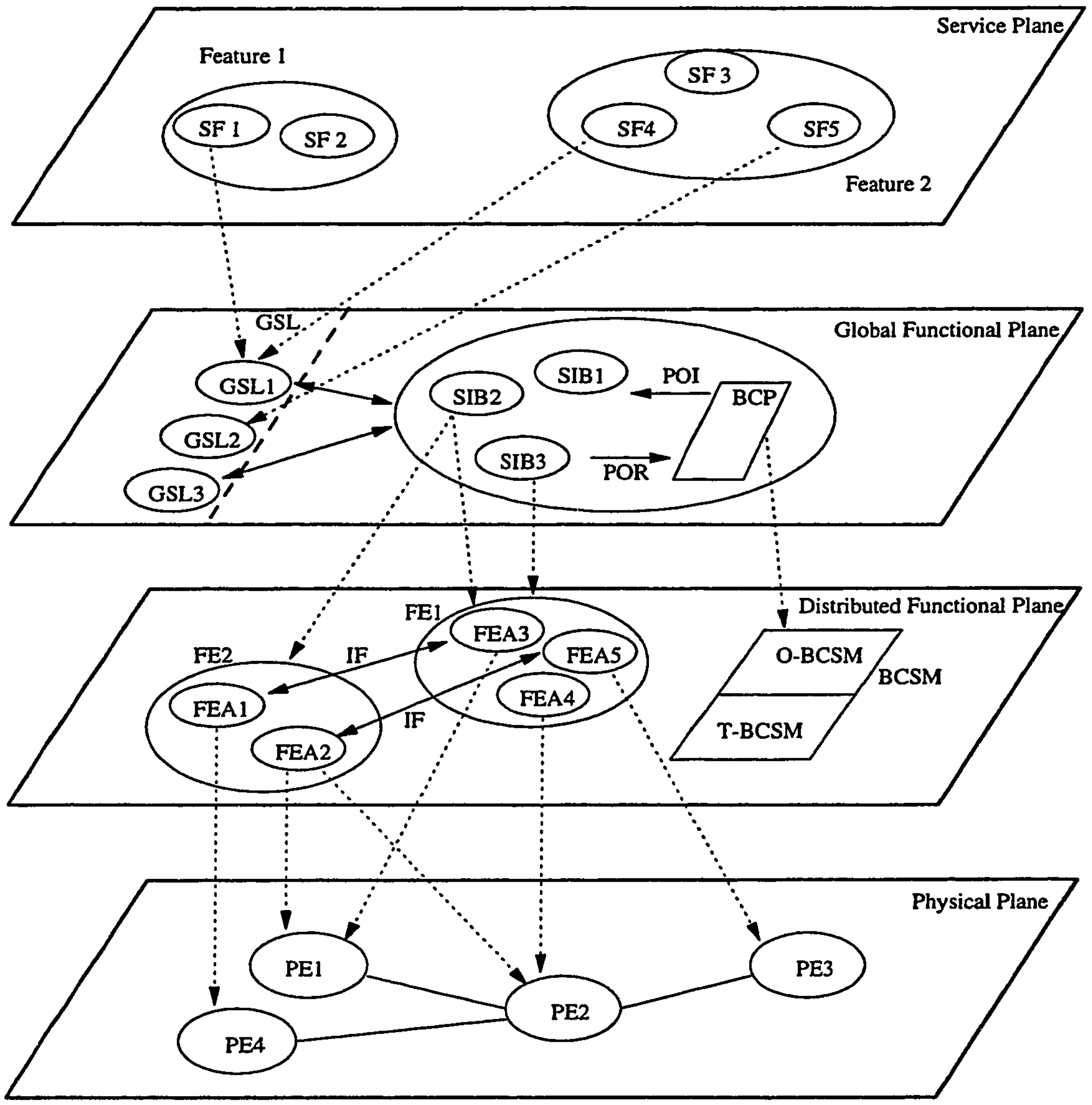


Figure 2-13: Mapping of INCM Planes [Q.1201]

2.4 Phased IN Development

During this decade, telecommunication area has seen more new technologies introduced than most other industry areas. This led ITU-T to acknowledge that IN needs to be able to evolve in order to survive. Also, it seems there are still many detailed technical issues that have to be addressed by further research. As a result, ITU-T has adopted a phased standardization process towards the target IN architecture, defining a capability set for each phase. Each IN capability set (CS) is intended to address requirements for one or more of the following: feature creation, feature deployment, feature processing, and feature management in one phase. A series of Capability Sets (CS) are defined to represent different groups of IN capabilities that are subjected to be standardized at different evolving stages of IN systems.

The ultimate CS would be the Long Term Capability Set (LTCS) for the target IN. Currently we have CS1 and CS2. CS3 and higher sets may become available in the future. All the ITU-T recommendations are grouped according their CS numbers, as shown in Table 2-3.

Q.1200 – Recommendation Structure	
Q.120x -- IN Overview	
Q.121x -- CS - 1	Q.12x1 -- Principle Introduction
Q.122x -- CS - 2	Q.12x2 -- Service Plane (not included for CS-1)
Q.123x -- CS - 3	Q.12x3 -- Global Functional Plane
Q.124x -- CS - 4	Q.12x4 -- Distributed Functional Plane
Q.125x -- CS - 5	Q.12x5 -- Physical Plane
Q.126x -- CS - 7	Q.12x6 -- For Future Use
Q.127x -- CS - 7	Q.12x7 -- For Future Use
Q.128x -- CS - 8	Q.12x8 -- Interface Recommendations
	Q.12x9 -- Intelligent Network User's Guide
Q.129x -- Glossary	

Notes:
1. The ten's digit stands for the corresponding Capability Set.
2. The one's digit represents the particular aspect inside each Capability Set.
3. Q.1290 is the terminology & acronym glossary.

Table 2-3: Q.12xx Recommendation Framework Structure [Q.1200]

The Q.12xx series of standard is organized according to the separation of capability set. Beside the IN architecture defined in each of the capability sets that may evolve from time to time, ITU-T also presented an important framework for the description and design of IN system -- Intelligent Network Conceptual Model (INCM). It represents an integrated formal framework in which models and concepts of IN system are identified, described and associated. This is the foundation of IN systems and is intended to remain stable for a relatively long period.

2.5 Chapter Summary

In this chapter, a brief introduction is given on the motivation and emerging of the Intelligent Networks. The most important goal of IN is to facilitate the fast and cost-efficient development of telecommunication features.

Based on ITU-T Q.12xx series recommendations, we described the overall functional requirements of such IN systems as well as the Intelligent Network Conceptual Model (INCM), which is meant to serve as a framework of both the current and future IN systems. INCM consists of four functional planes that are essentially different technical descriptions of the same IN system at different abstraction levels.

Among the four functional planes, Service Plane (SP) is considered as general functional specification. Global Functional Plane, Distributed Functional Plane and Physical Plane are IN specific. Mappings between adjacent planes are also defined. The development and standardization of IN adopted a phased paradigm that allows IN-structured system to evolve from CS1 to CS2, CS3, ...

Chapter 3 Modeling of Intelligent Networks Using SDL

3.1 Overview of SDL

Formal languages and methodologies can be used to specify and model the behavior of a system and to verify that the system design and implementation satisfy the pre-defined functional and safety properties. SDL is such a formal language for the specification and description of complex entities. It has been developed and standardized by ITU-T and has become one of the most widely used languages for the description and analysis of real-time, concurrent, distributed and interactive telecommunication systems.

3.1.1 History of SDL

Starting from 70's, there was a significant change in telecommunication systems from electromechanical systems with simple signaling methods to complex computer-controlled systems with sophisticated signaling protocols. During this period, CCITT (former name of ITU) began the development of SDL and it was put into use by a substantial portion of switching system engineers. The first version of the language was available as early as 1976, followed by newer versions published in 1980, 1984 and 1988. The language actually reached a stable form as described in the single recommendation

Z.100 of the CCITT Blue Book, to which a formal Mathematical Definition is attached as an appendix ([Olsen 94]). Currently the latest version of SDL is the SDL-96.

Many versions of SDL have been in use for a relatively long period of time that had allowed more practice experience to be obtained. The experience from actual use has contributed to the improvement of the language. Even 10 years ago, a survey at SDL forum had shown that over 5000 researchers and engineers were using SDL worldwide. The CCITT Study Group also uses SDL in their technical recommendations. The forum further revealed that many activities of SDL focused on developing and using of computer-aided SDL tools [Ferenc 89]. Today, SDL enjoys more and more popularity with new functions added in and the supporting tools becoming powerful and mature.

3.1.2 A Versatile Formal Tool

SDL is well known within the telecommunication field, however it has a much broader applicable area. Systems that having follow characteristics are ideal for using SDL:

- Type of systems: *Real-time, Concurrent, Interactive, Distributed*
- Describing Focus: *Structure, Communication, Behavior*
- Abstraction Level: Any level from *Overview* to *Most Detailed*
- Construction Type: Both *Homogeneous* and *Heterogeneous*
- System Control: Multiple *Concurrent Processes*

Although SDL is mainly used in telecommunication systems, it is also suitable for other application systems having a *real-time, interactive* and *distributed* nature, such as Credit-Card Processing Systems, Automatic Vending Machines, etc. It is efficient to capture the characteristics of a system, regardless of its size and complexity, from three different viewpoints (*structure, communication and behavior*) in the same model using SDL. This definitely helps to understand the system and avoid inconsistency throughout the design and implementation stages. SDL covers all abstraction levels and is applicable to either architectural design, module design, and detail design. The fact that SDL allows system descriptions to be communicated without ambiguity and be understood by people working on different stages makes product development highly efficient and easy to manage.

SDL can be used by various organizations in several ways [Rolv 96]:

- For international standards in communication area: to define unambiguous and consistent documents.
- For use in tendering: to specify the required behavior and compare provided behavior from different vendors.
- For use in system development: to design and generate an optimal implementation and to document the provided behavior.
- For verification and validation of critical behavior.

3.1.3 General Consideration over SDL

Due to the inherent complexity associated with IN system, features and FI problem, manual analysis without the help of automated software tools is very tedious and almost infeasible. That is why most people would like to choose a formal language that has commercial support tool sets to make their study more efficient and fruitful.

Choosing an appropriate technique and tool is no less important than determine the strategy of attacking the problems. Various Formal Description Techniques (FDT) have been used in the specification and design of telecommunication systems and the study of Feature Interaction problem. For example, [Bryce 94] used SDL, [FaciL 94] utilized LOTOS and [Comb 95] adopted MSC.

We decided to choose SDL as our modeling tool in the study of IN architecture, entities and features due to the following reasons:

- SDL supports different abstraction levels, which is very suitable for both the fast prototyping and fine-tuning analysis.
- The powerful modeling ability in either the behavior, structure and inter-module communication aspects make SDL ideal for system modeling.
- Object-oriented capability has been included into SDL, maximizing information hiding and re-usability, which leads to clear and efficient system descriptions.
- SDL has an intuitive Graphic Representation (SDL/GR) in addition to the text-oriented representation. This provides superior understandability and reduces the possibility of introducing errors during the system development stage.

- Excellent commercial software tool sets are available to support SDL, such as *SDT* from Telelogic in Sweden [SDT 93] and *ObjectGeode* from Verilog in France [Geode].
- Years of utilization in telecommunication area and widely industrial acceptance [Lucidi 96].
- Backed up by international telecommunication standards proposed by ITU-T.

3.1.4 System Modeling using SDL

SDL is a formal description language with explicit behavior-describing ability. The basis for description of behavior in SDL is the communicating Extended Finite State Machine (EFSM) model that maps to the definition of SDL process. Communication is represented by passing signals through signal routes and can take place between processes or between processes and the environment of the target system. Some aspects of communication between processes are closely related to the description of system structure. The Extended Finite State Machine is based on conventional FSM that consists of call states, transitions connecting the states and signals that enable the transitions. In EFSM, output signal set is appended in order to reflect the impact exerted to other entities and internal timer mechanism is added to deal with sequential conditions. The EFSM used to describe SDL behavior is illustrated in Figure 3-1 (adopted from [Rolv 96]).

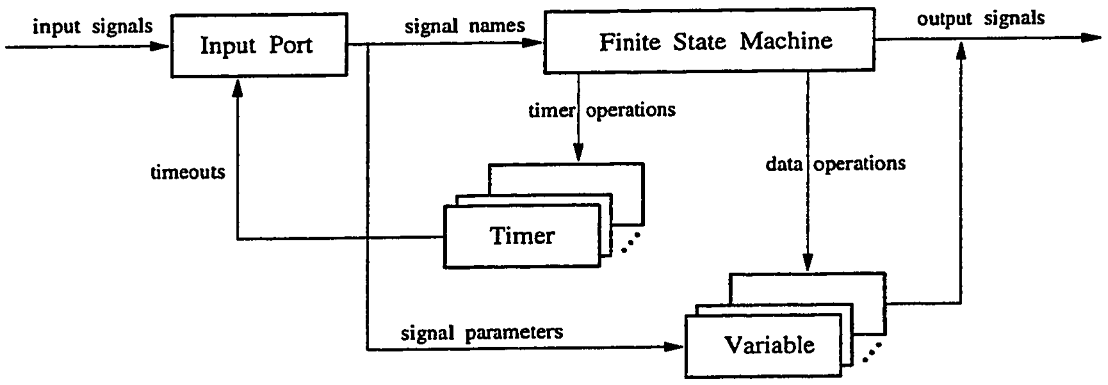


Figure 3-1: Illustration of EFSM used by SDL

There are two distinct forms of SDL representations: the SDL-GR (Graphic Representation) and the SDL-PR (Prose Representation, i.e., text form). Both forms are officially supported and can be converted to each other. The SDL-GR is more diagram-oriented, while the SDL-PR is text-based and the style is similar to a programming language like C++. We use SDL-GR because of its good understandability and intuitiveness.

The following are brief introductions to the major concepts and elements in the language SDL and how they are used to describe a target system. Please note that all SDL reserved words are in **bold** font.

3.1.4.1 Structure and Behaviors

The concepts introduced here are used for structuring of target systems especially large and complex systems.

System Structuring and Behavior

System structure is the structure of the SDL **system,** i.e., how is the system composed ? In SDL, the target system is represented as an SDL concept **system** surrounded by its environment, which is denoted as **env**. The objects that constitute the target system are called SDL components. These components may be of different types and their properties can either be defined directly or by means of **types**. The component concepts to construct a **system** in SDL include **block, service, process, procedure**, variable definition, **signal** definition and **type** definition.

As shown in Figure 3-2, the SDL **system** is the highest level concept, which represents the whole target object we are modeling. A **system** can be organized either from **blocks** or directly from **processes**, but NOT a mixture of them. Moreover, if the system is directly composed of **processes**, it cannot have any block defined in it. Each **block** is composed of **process**(es) that are independent entities having individual behavior. While a **process** can consist of a set of **service**(s) as well as a simple EFSM (Extended Finite State Machine), an EFSM that contains **procedure**(s), or a **service** can only be a simple EFSM or a EFSM that contains **procedure**(s). The significance of **service** is that only one **service** in a **process** is active (its program in execution) at any time. **Service** is very useful in describing different parts in a **process** that may have different kinds of behaviors when meeting certain situations or input **signals**.

A SDL **system** is composed of **blocks** that enclose a part of the system and permit a view of the system at a certain abstraction level. A **block** may again contain **block**(s), resulting in a hierarchy of **blocks** within the **system.** Each "atomic" **block** (does not contain any other **block**) is specified as a number of communicating **process** sets. Each **process** set stands for zero or more **process** instances. This number may change during the lifetime of the **system** because process instances may be created or terminated dynamically. A process instance may be created by other process instance but terminates only by itself.

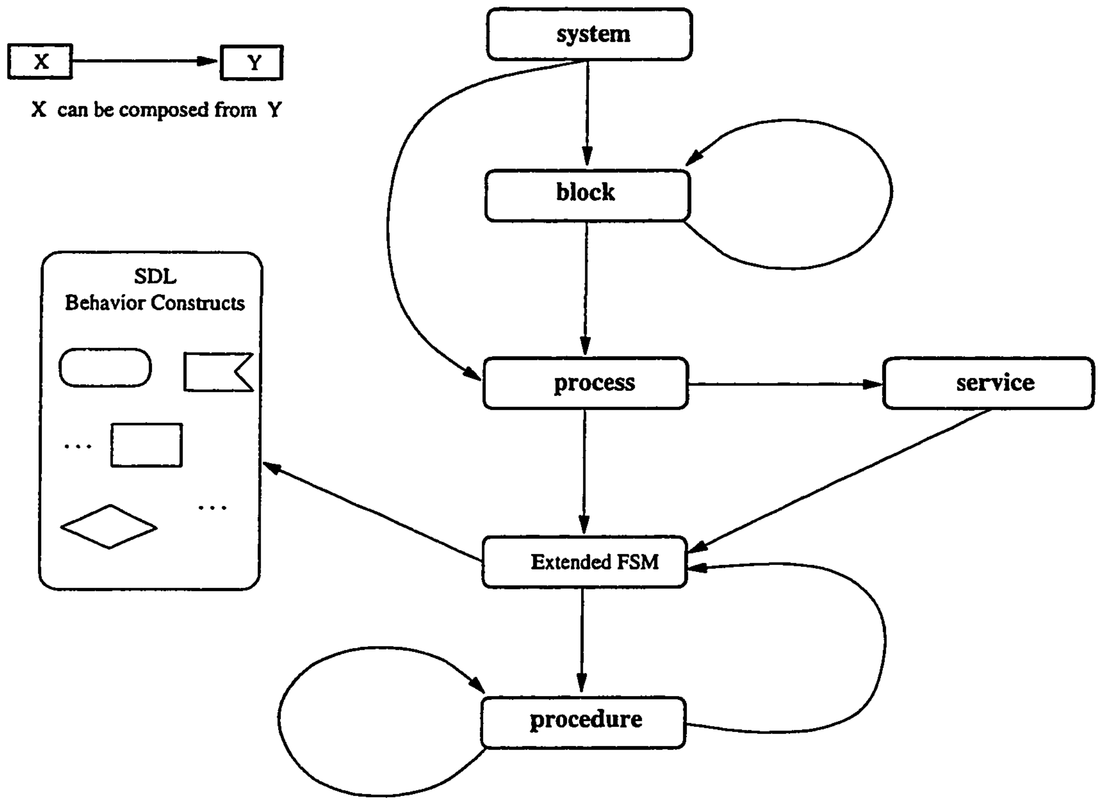


Figure 3–2: Hierarchy of system construction in SDL

The example in Figure 3-3 shows an example target system that consists of two **blocks** called *Coding* and *Controller*. They communicate with each other through **channel** *Internal*, and with the environment via **channel** *Line* and *Bus*. There are many specified **signals** that can be transmitted using these **channels** that are defined on the signal definition sheet in the **system**. The *Coding* is a **block** reference that stands for a concrete block entity, and the *Controller* is an instantiation of a **block type** *CType*. The GenParity is a **process type** that can be used to instantiate any number of process instances.

An example **block** shown in Figure 3-4 consists of two **process** sets, called *Parity* and *CodeServer*, together with **signal route** *GetPar* that carry information between the two **process** sets and **signal route** *ToCon* and *InOut* that allow this **block** to communicate with outside entities. CodeServer is a **process** reference that represents a concrete **process** instance. However the *Parity* is a set of **process** instances (zero or more) of the **process type** *GenParity*. Each of them is dynamically created by CodeServer as indicated by a directed broken line pointed to Parity in Figure 3-4. All the **signals** that can be carried by these **signal routes** must be specified by signal definition sheet.

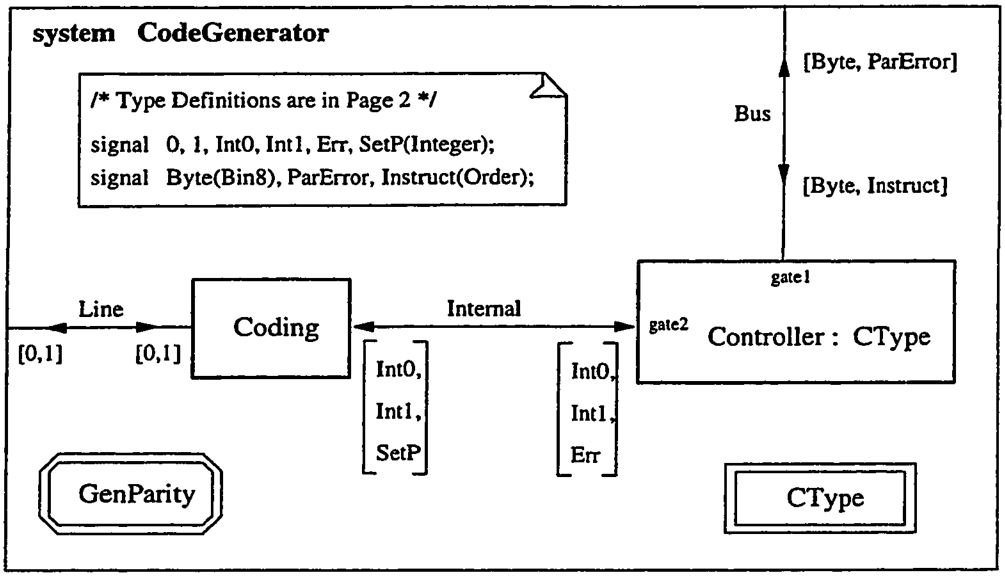
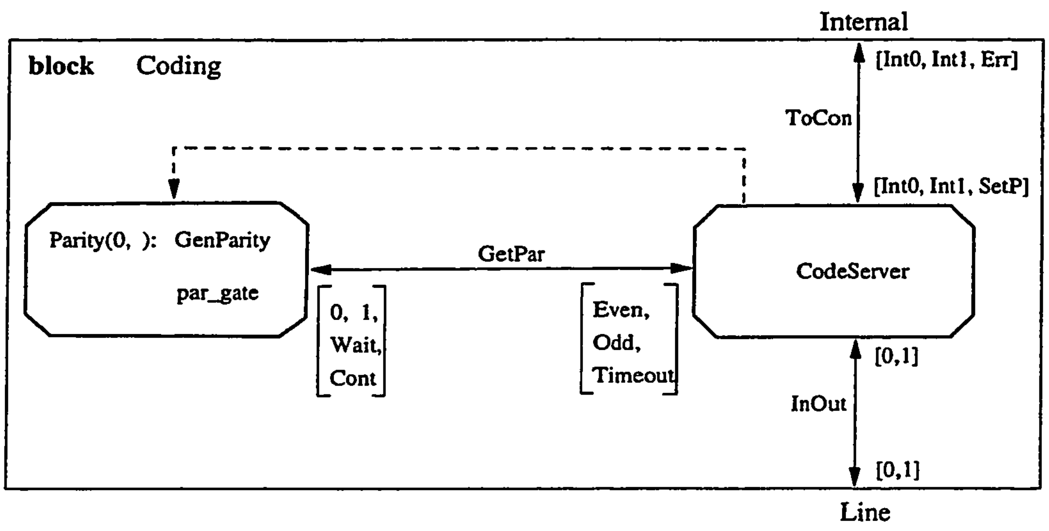


Figure 3-3: Example of a Target System in SDL

An example **block type** *CType* is also shown in Figure 3-4. A **block type** is viewed as the blueprint of a **block**, it can be used to instantiate an actual block in the **system.** The structure of a **block type** is almost the same as that of a **process**, except that **block type** uses **gates** instead of **signal routes.** This is because the **gates** need to be matched to outside **signal routes** upon the instantiation of the **block type.**

Most of the elements, being represented with various symbols, used in **process** description are illustrated in Figure 3-5. The mechanism is largely based on the concept of extended FSMs that are defined by states, events (input signals), output signals and transitions. In the SDL diagram, the state without a name is the **start** state where the process starts executing upon creation. The asterisk " * " specified in the state means all states, however certain states may be excluded by putting them in a pair of brackets "( )" under the asterisk. And similarly an asterisk in an input symbol means all **signals** are eligible. In this case certain **signals** can be excluded by listing them in a pair of brackets "( )" under the asterisk.



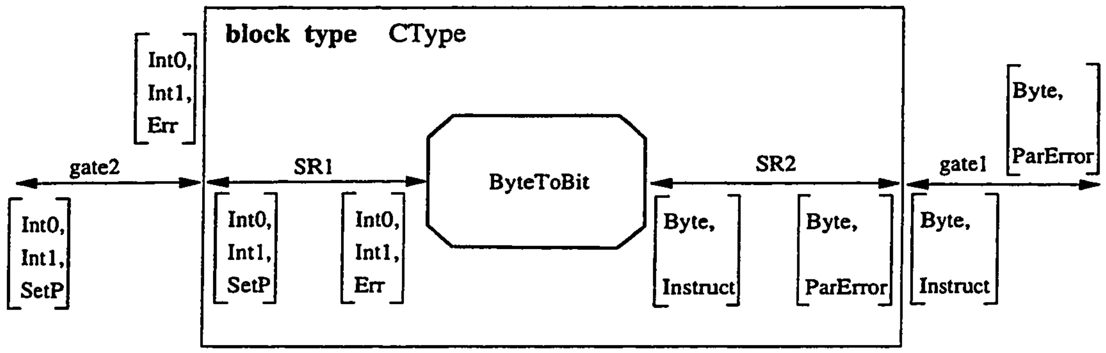


Figure 3-4: Example of an SDL block specification

Another characteristic of SDL is the strong capability to express *non-determinism.* There are two frequently used ways to describe non-deterministic behavior: *spontaneous input* and *non-deterministic decision.* The former is specified using a **none** in the input symbol, which means the transition can happen at any time without specific input. The latter uses "**any**" keyword as the decision condition, which signifies that the **process** may randomly choose one from a few pre-defined transition paths that lead to different behaviors.

An SDL **process** can be specified directly as an EFSM or by means of SDL **services.** Each **service** represents part of the **process**'s overall behavior. **Services** in the same **process** do not run in parallel but are executed sequentially in a pre-determined order or based on input **signals.** A **process** can be split into **services** as shown in Figure 3-6.

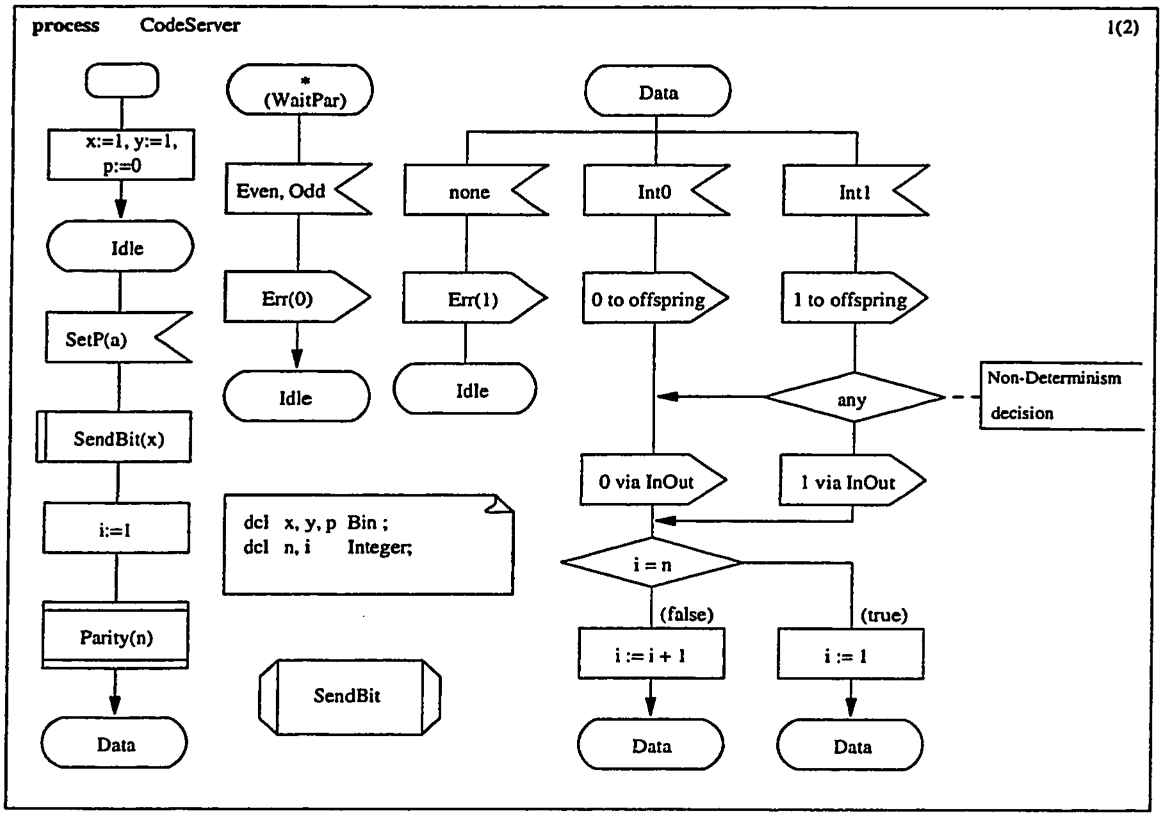


Figure 3–5: Example of an SDL process (partial specification)

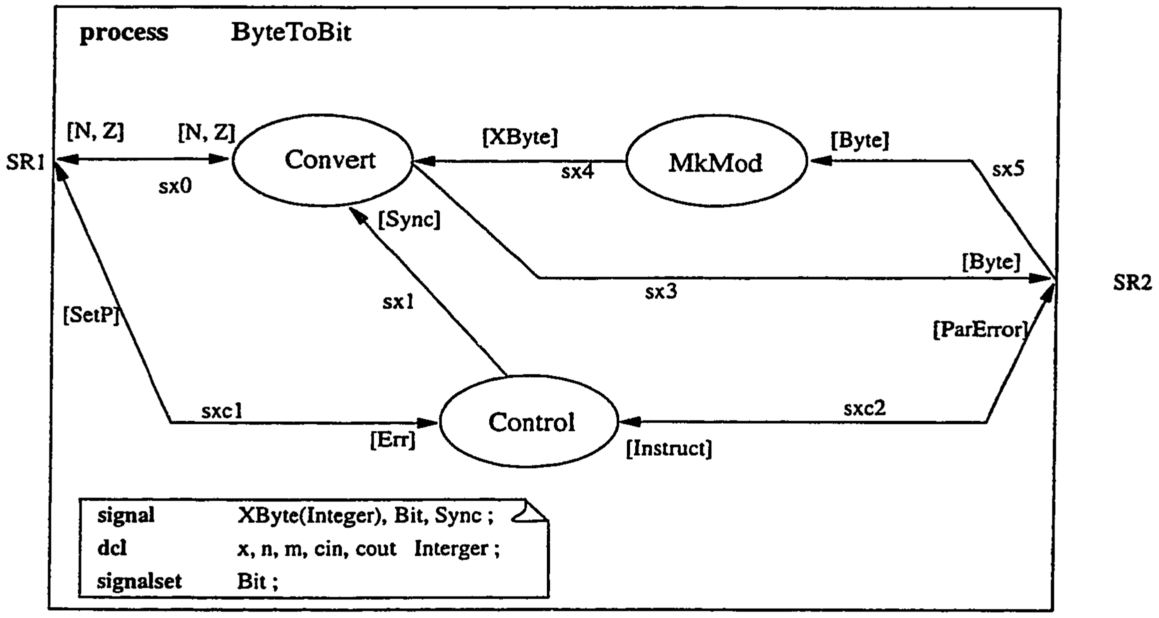


Figure 3–6: An SDL process that consists of services

The behavior of the **process** will be ultimately based on the EFSM model that essentially consists of states, events, output, actions, transitions, etc. One point we need to note is that, in the specification of EFSMs, the concept **procedure** can be used to gain better understandability, information hiding and design efficiency. A **procedure** is a re-usable routine that can be called by name in a EFSM using a procedure instantiation symbol to get a certain task done. Also a procedure can recursively call other procedures as well. Figure 3-7 shows an example procedure call and procedure specification.

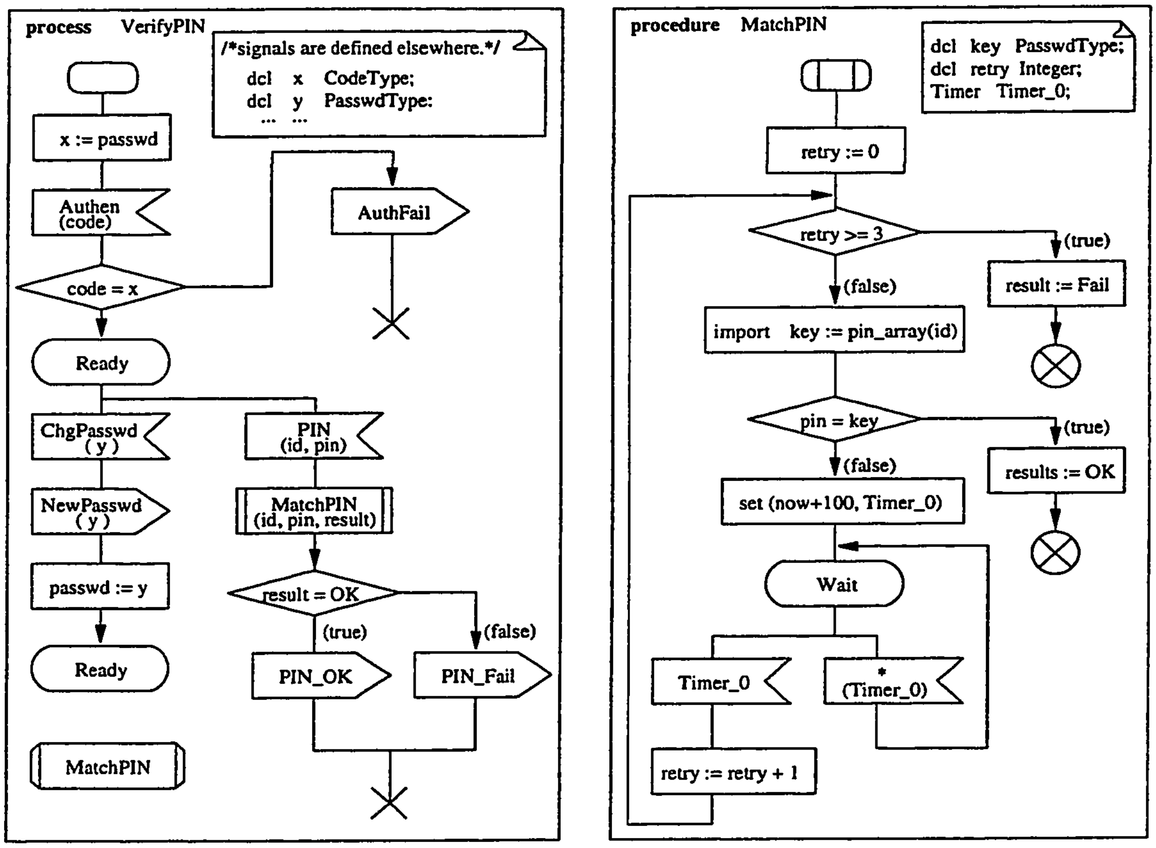


Figure 3-7: Example of procedure and procedure call

Generic System Specification

System Specification structure is the structure of the specification (document) itself. For example, divided into diagrams, pages, use of macros, and alternative specifications. System structure usually implies a certain structure of the specification, however when specifications become large and complex, structuring is also desirable to be applied to the specification.

- Referenced definitions and referenced diagrams are used to simplify the nested system definition and make the specification more readable.

- Package can provide the ability of a sharable "part-library" in order for other systems to re-use components or type definitions of the current system.

Generic system specification means the same SDL generic system specification can be interpreted as different SDL system specifications when different actual values are applied to generic parameters, as shown in Figure 3-8.

Macros can also be used to replace repetitive parts of specifications. This is similar to the macro concept in a programming language.

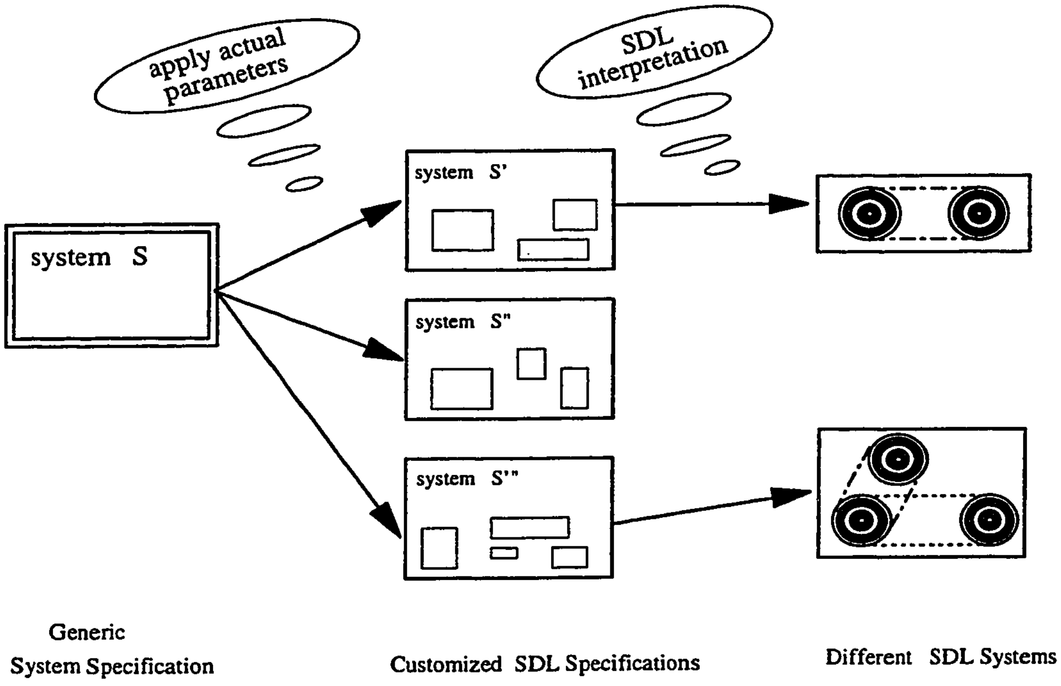


Figure 3-8: Principle of Generic System Specification

3.1.4.2 Modeling Communications

Signal interchange is the fundamental communication scheme in SDL. In SDL specifications, communications inside or outside the system are carried out through certain kinds of "information pipes". **Channel** and **Signal Route** are the frequently used ways of communication in SDL specifications. **Gates** are the connection points for channels and signal routes. **Channels** are used as the conduit of messages between the whole system and its environment as well as between different blocks or block sets inside the system. **Signal Routes** interconnect different processes inside a block and serve as the paths for the exchanging of signals.

Both the channels and the signal routes can be either unidirectional or bi-directional. Directions must be specified using "arrow(s)" on the channel symbols or signal route symbols. **Channels** may be either delaying channels, which attach a random time delay to signals transmitted through it, or non-delaying channels, which deliver messages instantly. Both types of channels are quite useful in SDL system specifications. **Signal Routes** are only of non-delaying type. Figure 3-9 depicted different channels and signal routes.

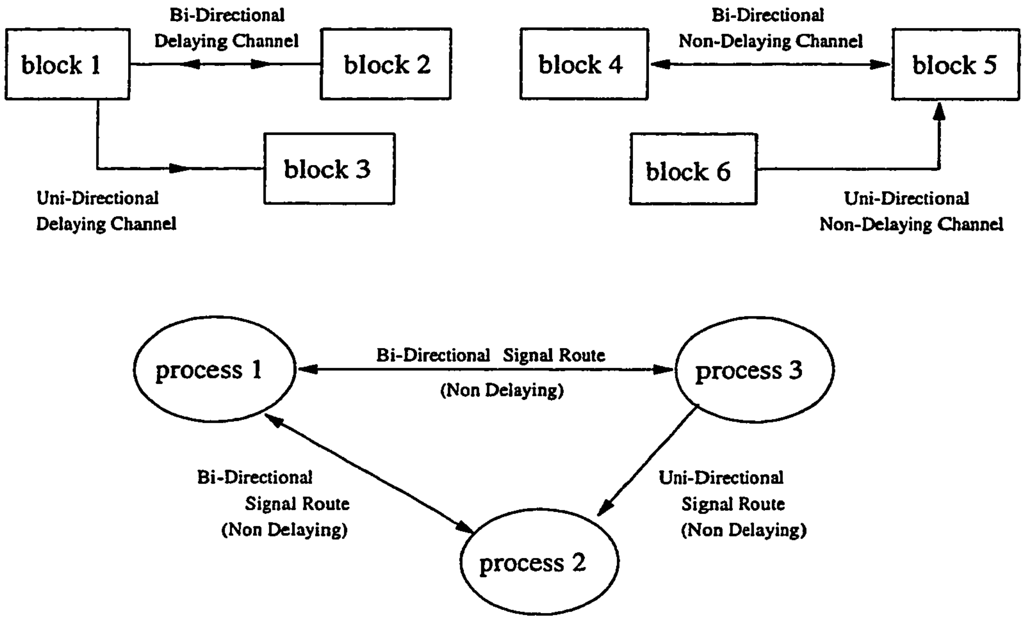


Figure 3-9: Examples: delaying, non-delaying channel and signal route

Actual communications among system components are based on the communication primitives called **signals** that can convey both messages and values. **Signals** are atomic communication events appearing between the **system** and its **environment** or internally between different components of the system. They are created when a **process** executes an output action and are consumed by the receiving **process** after executing an input action.

3.1.4.3 System Data

Two levels of definition and usage of data are possible in SDL specifications: ***built-in*** *data manipulation* and *self-defined data operations*.

- SDL includes many pre-defined data types and built-in constructs, which makes it possible to use SDL in the same way as any other powerful programming language.
- The ability to define and use new data types and operators in SDL is no less than that of built-in data types.

SDL data types are ***abstract data types*** (ADT), which means each data type has an interface part defining how and which literals and operators can be used and a behavior part defining the semantics of the literals and operators. The concept of ADT reflects the fact that most operations can be applied without knowing any details about how things are really done. This helps to gain better information hiding and appropriate abstraction level in the specification of target systems.

3.1.4.4 Advanced Issues

There are many powerful mechanisms in SDL that can be used to describe subtle behavior or characteristics of the target systems. Here we only mention a few of them. For more about the SDL language and its application please refer to [Olsen 94].

Object-Oriented Specification

The basis of object-oriented modeling in SDL is the multiple abstraction levels allowed in the system specification. As shown in Figure 3-10, the existence of multiple abstraction levels is supported by a hierarchical system structure and abstract data types and operators.

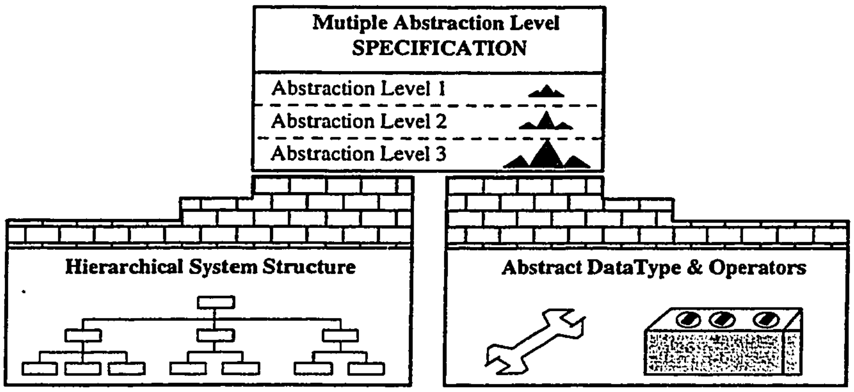

Mutiple Abstraction Level
SPECIFICATION
Abstraction Level 1
Abstraction Level 2
Abstraction Level 3
Hierarchical System Structure
Abstract DataType & Operators

Figure 3-10: Supporting of Multiple Abstraction Levels

Type specialization in SDL is realized by means of *inheritance*, *virtual types* and *virtual transitions.*

- *Inheritance* means obtaining properties (or attributes) from base type(s) and adding new ones, which is similar to that in a OO programming language like C++.
- A *virtual type* denotes a type where some of the attributes are yet to be further specified in types that inherit it. This is somewhat analogous to virtual class in C++.
- *Virtual transition* allows part of the behavior of an entity type to be left for further specification in a specialized entity type. This is more powerful than the virtual function mechanism in C++ since virtual function does not allow specialization on such a tiny granularity -- a decision action or a transition.

Parameterized Type

An SDL type that defines system construction components (such as block type, process type, etc.) can be parameterized so that it is partially or completely independent of the definitions in its enclosing scope units. A parameterized type can be used in two ways:

- Actual values are provided as part of the specification of an instance set (e.g., number of processes in a block). All parameters must be provided when utilizing this type.
- Actual values are used as running context of an instance set (e.g., dynamic values passed in which may determine the execution results of this instance set). Depending on internal design of the instance set, some parameters may not be mandatory.

Parameterized types provide the flexibility to construct the target system components based on the actual parameters given.

RPC and Variable Reveal/Peep

A few other mechanisms are provided by SDL to describe the many possible ways that concurrent system parts can interact:

- A remote procedure is a procedure that can be called in a process different from the one in which it is defined. RPC (remote procedure call) is a popular communication model in distributed systems (e.g., for application layer protocols in telecommunication standards). The way of using RPC in SDL is illustrated in Figure 3-11 [Olsen 94].
- The reveal/peep mechanism in SDL allows a process in one block to "peep" at the value of variables in other processes within the same block, if these variables are defined as **revealed** in their owner process. However a process cannot peep at the

variables in another process if that process is not in the same block. Under that situation, those variables need to be **exported** by its owner process into a global environment and **imported** by the user process. But the values of such exported variables do not change with the actual variables inside the owner process. The only way of refreshing a variable is for the owner process to execute an export action whenever the variable is updated.

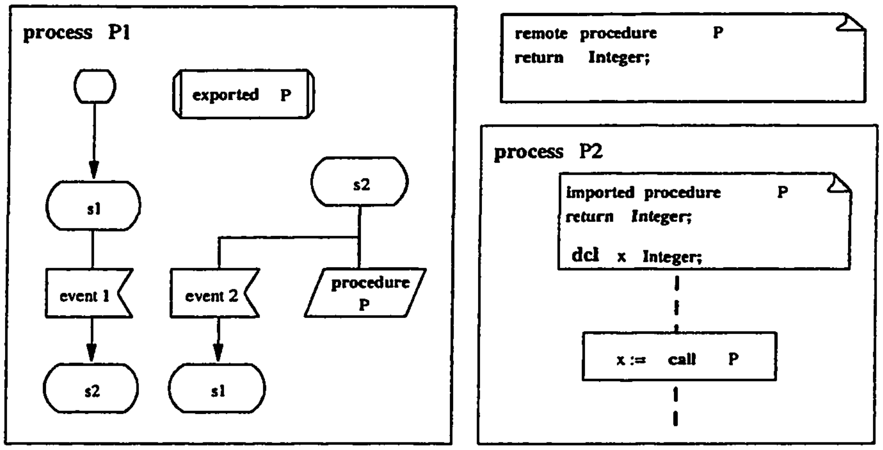


Figure 3-11: Example of remote procedure call

3.2 Modeling IN system using SDL

Intelligent Network is a new network architecture. From the feature development point of view, it aims to achieve a few major goals: (1) Effective solution that enables new features to be designed, implemented and deployed into the system in a reasonable period of time; (2) The network must be able to accommodate large number of features at the same time, and be efficient in handling the situation when many feature instances are triggered simultaneously by the feature users; (3) Maximizing the re-usability of the call-processing software components and enhance the portability of features from different vendors. All these have been taken into consideration when IN principles are presented by ITU-T. The ITU-T Q.12xx series IN recommendations are used as the technical standards of many actual IN systems. Our IN model is also based on them. We work mainly on the Distributed Functional Plane of INCM in specifying the IN system model. However, many important IN concepts in SP and GFP are also covered.

Intelligent Network is technically quite complex. It has been separated into phased development and standardization (in the form of different Capability Sets) by ITU-T in order to sweep out a way for building practical IN systems when some aspects of IN technology are still under research. Thus the IN standards are still evolving and expanding (e.g., from CS1, CS2 to CSx...). Our IN model is meant to help us analyze the IN system, features and feature interactions mainly under CS1.

The reason we develop such a model on DFP is to reflect the essence of IN architecture, characterize the behavior of relevant IN entities, capture the relationship between IN entities. Another important point of our IN model is to be able to verify the existence of feature interactions and analyze the reasons that cause the interaction by simulating the interaction situations. To express the real-life IN systems more precisely and objectively, our model of IN system is supposed to include the operable framework and function elements. Our model also considers the ability to specify and accommodate various features as well as the components used to build them without affecting the framework and the IN entities. We are more interested in the IN entities that are directly involved in the triggering and execution of the IN features. Some less relevant functional entities (such as feature creation and feature deployment entities) are not described in our model.

3.2.1 Overall IN system

In the modeling of IN system using SDL formal language, the major decision is how to choose appropriate SDL constructs to represent the IN system, the functional entities inside it, and the behavioral activities of these entities. The decision is always based on the modeling target, modeling objectives and the scope of modeling.

Within the group of people who use SDL as their modeling tool, different researchers have adopted different mappings between IN concepts and SDL constructs. [Csurgay 95] described their mapping of using a separate SDL **system** to represent each IN Functional Entity (FE). That is why they had to develop some mechanism outside these IN FEs, so that the SDL **systems** that represent these FEs can be run cooperatively. They took this mapping because their modeling goal was only the creation and simulation of IN features. [Conor 95] used SDL **system** to model two functional planes, GFP and DFP, because they only focused on SIB-based feature specification and creation rather than an operable IN system model. [Lucidi 96] mapped each IN feature into an SDL **system** since they studied the behavior and characteristics of IN features but not the IN functional entities.

Our goal is to analyze the IN architecture, various features, and FIs among these features. We need to target the whole IN system. Hence, we consider the IN network as a whole with distributed basic call processing capability and centralized feature processing power. Features are located, triggered and executed inside this network. With this view of IN system, the concept "**system**" in SDL has been used to represent the overall IN network in our model. Correspondingly, other IN entities/concepts are mapped into various SDL constructs as summarized in Table 3-1.

Intelligent Network Entity / Concept	**Corresponding SDL Concept**	**Comments**
The whole IN	**system**	End users are outside the system
Functional Entity/FE Group	**block**	A block may in turn contain component blocks
Information Flow (IF)	**signal**	Signals that passed among different FEs.
Feature / Sub-Feature	**process type**	Sub-Features can be used when designing Features.
Feature Instance	**process**	This refers to actual invocations of Features.
SIB	**service**	Re-usable Service Independent Building-block
Point Of Initiation (POI) & Point Of Return (POR)	**signal**	They indicate the transfer of call control power. POI and POR information is passed via signals.
Signaling Paths among the Functional Entities	**channel**	Modeled using SDL delaying channels, which have a random transmission delay.
FE Action (FEA)	**procedure**	Represent call actions executed in various FEs.
Basic Call Processing (BCP)	**process (type)**	A BCP process is created to work for one call-connection of a user (one user may have >1 call-connections simultaneously)
O-BCSM and T-BCSM	**procedure**	Called by BCP depending on call situation.
Service Support Data (SSD) & Call Instance Data (CID)	**data type**	Encapsulated in and passed through signals.
Global Service Logic	**process type** definition	Define how Features / Sub-Features are constructed using SIBs.

Table 3-1: Mapping between IN entities/concepts and SDL concepts

Our model is based on the DFP of INCM. On this plane, an IN consists of a set of FEs. Recall that we explained in Chapter 2, Call Control Agent Function (CCAF) serves as the interface between the user and IN system; Call Control Function (CCF) deals with basic call processing and signaling, also making use of special call equipment; Service Switch Function (SSF) performs feature-triggering checking, call-processing control switching and call data providing to SCF; the Service Control Function (SCF) is responsible for feature instantiation, execution and special call resource allocation as well as data access; Service Data Function (SDF) provides the necessary service support data (SSD) and/or call instance data (CID) to SCF so that features can be executed correctly.

As shown in Figure 3-12, the whole IN system consists of a few **blocks**: *The_CCF_SSFs* is responsible for normal call processing and feature request reception. It contains two IN entities: CCF and SSF; *The_SCF_SDF* is the intelligent unit that supports various telecommunication features. It actually contains two sub-blocks that represent SCF and SDF respectively. *The_SRF* provides special purpose call devices used by some features. Each of the CCAFs (*CCAF_1, ... CCAF_n*) provides access to the IN, delivers communication data, and presenting system responses to the users.

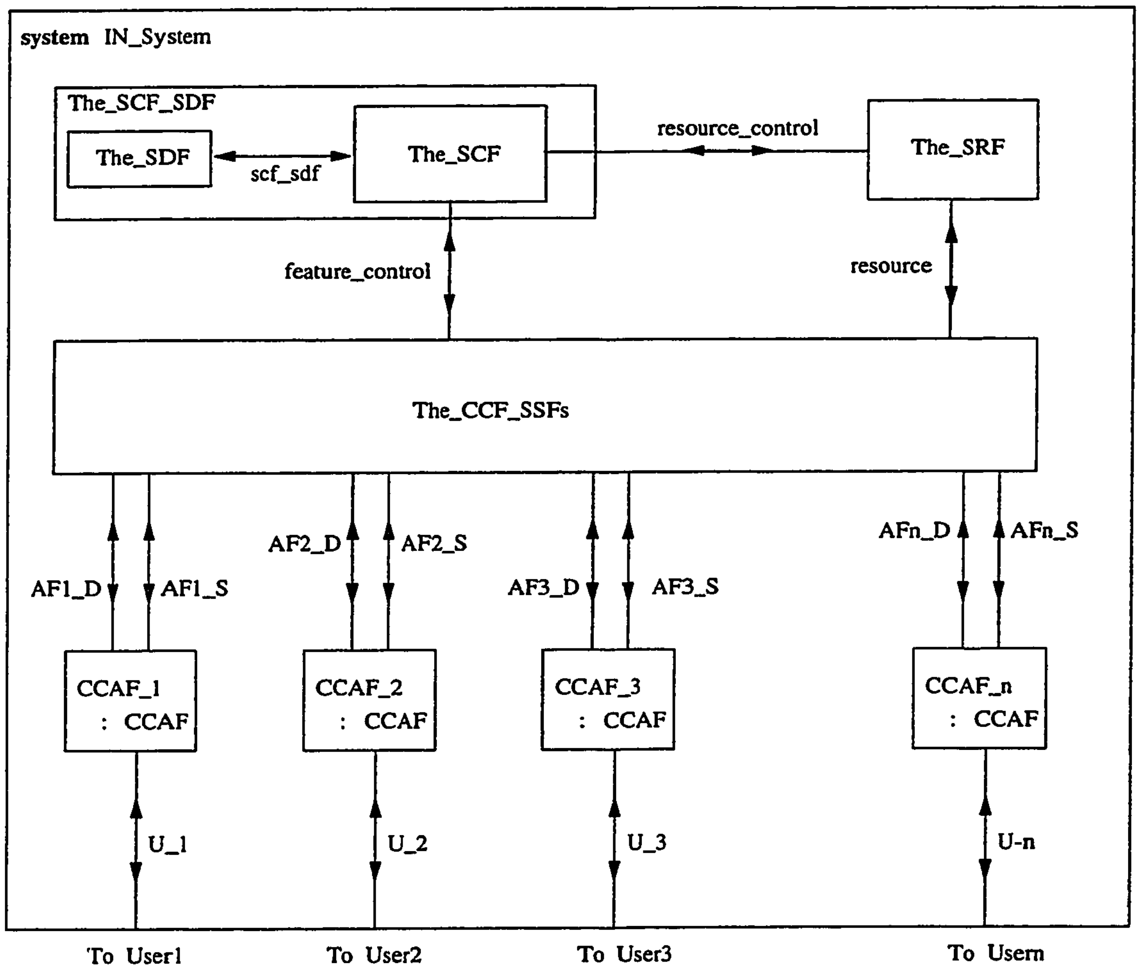


Figure 3-12: Modeling IN network as an SDL "system"

The communication **channels** among the **blocks** in the *IN_System* are summarized here: *feature_control*, *resource_control* and *AFx_S* (x = 1,2 .. n) are pure signaling routes. The *resource* and *AFx_D* (x = 1,2 .. n) are pure data transmission paths. The *U_x* (x = 1,2 .. n) channels are combined communication paths that carry both communication data and control signals.

We explain how all these IN entities and the communication routes among them are modeled in the following subsections.

3.2.2 CCAF entity

Call Control Agent Function (CCAF) entities (*CCAF_1*, *CCAF_2*, ...*CCAF_n* as shown in Figure 3-12) are the blocks responsible for data/signal interpretation and relay between the users and *The_CCF_SSFs*. They provide the users with interfacing and access to the IN system. The major functions of CCAF entities include:

- Interacting with the user to establish, maintain, modify and release, as required, a normal call or a call with certain feature(s).
- Accessing the feature-providing capabilities of CCF, using specific signaling mechanism such as pressing buttons for "transfer", "setup", "hold", etc.
- Receiving indications (prompts) relating to the call and features from the CCF and relaying them to the user as required.
- Maintaining call status as well as feature running state information.

In general, the CCAF is the unit that accepts user input and delivers the network prompts or signals or announcements to the user. In our model, each CCAF entity is modeled using a **block** of the **block type** *CCAF*.

The communications between a CCAF entity and the user are transmitted through an SDL bi-directional delaying channel (the *U_x* channel), which carries both the connection signals and the actual communication data, this is the concept "In-Band-Signaling". The communication data and control signals are all pre-defined as SDL **signals** to be allowed to transmit through these channels.

However there are two **channels** between the CCAF entities and *The_CCF_SSFs* block, one for control signaling, one for data transmission. This is the technology of "common-channel signaling" or "out-band signaling" explained in Chapter 2. Both of the two **channels** are delaying bi-directional since they connect two different **blocks** that may be geographically located in different sites in the IN system.

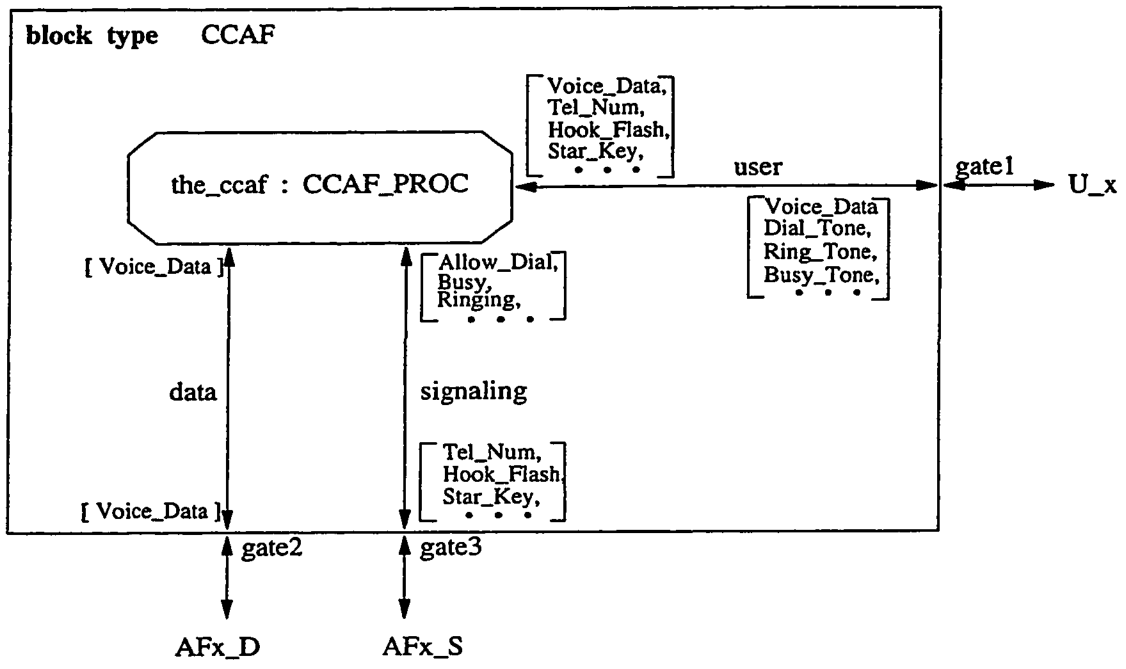

block type CCAF
the_ccaf : CCAF_PROC
Voice_Data, Tel_Num, Hook_Flash, Star_Key, • • •
user
Voice_Data Dial_Tone, Ring_Tone, Busy_Tone, • • •
gate1
U_x
[Voice_Data]
Allow_Dial, Busy, Ringing, • • •
data
signaling
Tel_Num, Hook_Flash, Star_Key, • • •
[Voice_Data]
gate2
gate3
AFx_D
AFx_S

Figure 3-13: Diagram of CCAF

Figure 3-13 indicates the structure of the **block type** *CCAF*. Inside each CCAF **block**, one **process** is designed to fulfill the task, acting as the facade between the user and the IN system. The **process** is of type "CCAF_PROC" and is created statically when the IN network is built. Every CCAF **block** has such a **process** (an instance of the "CCAF_PROC" **process type**). Each CCAF entity serves only one user.

3.2.3 CCF-SSF

The_CCF_SSFs **block** deals with basic call processing and feature-triggering checking. Normal calls are processed completely by *The_CCF_SSFs* without having to bother SCF and SDF entities. But if any feature-triggering condition is successfully detected during a call, *The_CCF_SSFs* will notify SCF, switch call control power to SCF and act as instructed by SCF until the control power is switched back from SCF after the feature's termination. So *The_CCF_SSFs* does not have to know much about how the feature works, it only helps to determine (during the processing of a call) whether one or more features need to be triggered. If yes, let the SCF and SDF deal with it. Otherwise it will handle the call normally (same as POTS).

As shown in Figure 3-14, *The_CCF_SSFs* block consists of one *Switching_Network* block and a set of CCF_SSF blocks. The *Switching_Network* carries out the actual data transmission and signaling among all the CCF_SSF blocks. There are two **channels** between the *Switching_Network* block and each CCF_SSF block because the Common-Channel Signaling mechanism is used. One channel *Dx* for data transmission and one channel *Sx* for call control signaling, where x = 1, 2, .., n. Because the *Switching_Network* is not a new IN entity, we will not elaborate this block.

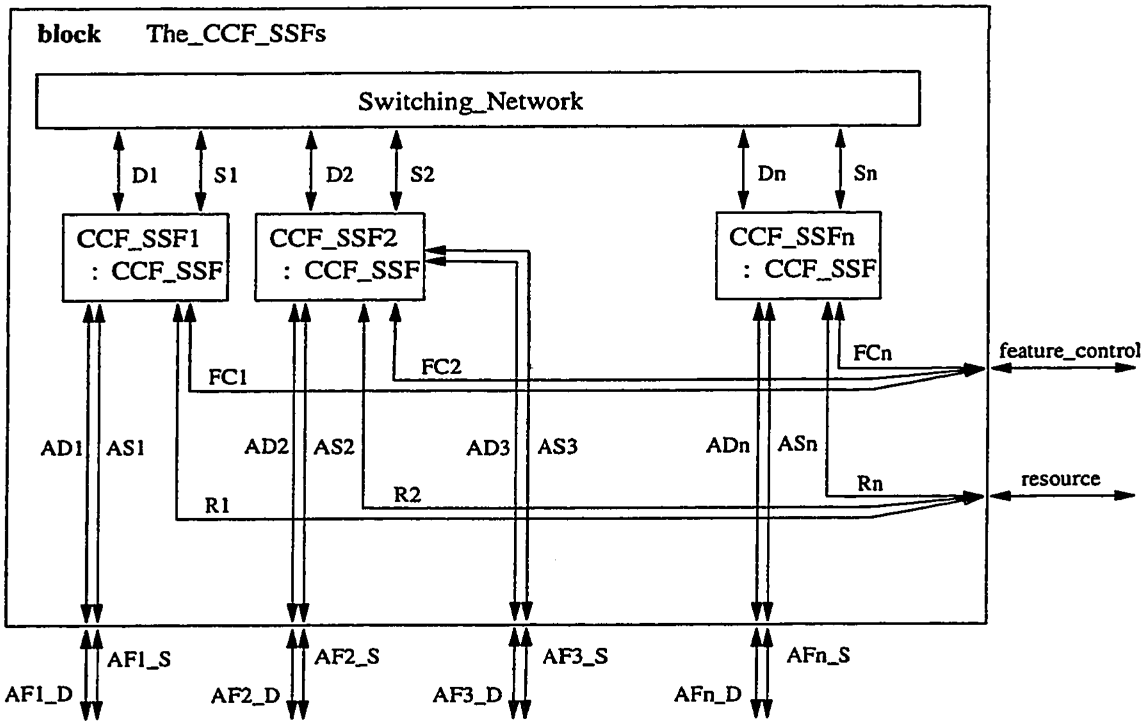


Figure 3-14: SDL diagram of CCF-SSF

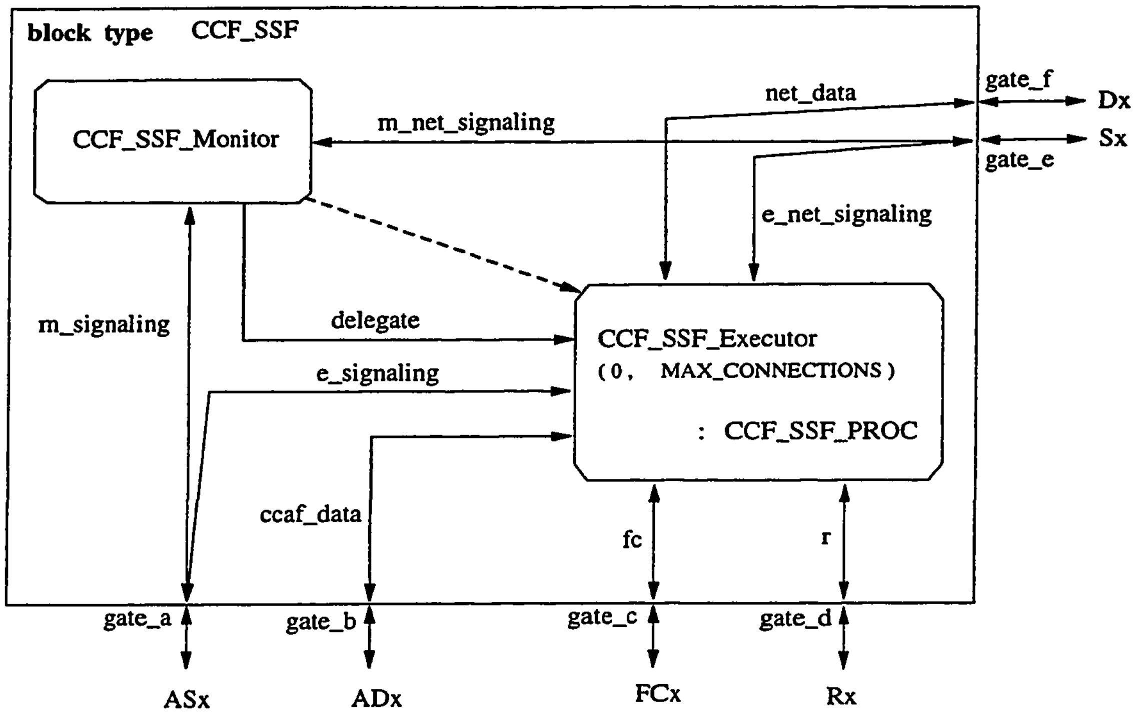
Each of the CCF_SSF blocks (CCF_SSF1, CCF_SSF2, .. CCF_SSFn) roughly represents a local switching node. It is connected with a set of CCAF entities that in turn connect users. A CCF_SSF block also has channels connected to SCF that serves as the intelligent unit in the network and supports various features, as well as to SRF that provides special call devices. We can see in Figure 3-14, channels *AD_x* are used for data transmission and *AS_x* for signaling between CCF_SSF blocks and CCAF entities; Channels *FC_x* are for signaling between CCF_SSFs and SCF entity; Channels *Rx* are used for data transmission between CCF_SSFs and the SRF entity (x = 1, 2, .. n).

Each CCF_SSF is modeled as a block with the **block type** *CCF_SSF*. The definition of *CCF_SSF* contains two sets of **processes**: the *CCF_SSF_Monitor* **process** and a number of actual *CCF_SSF_Executor* **processes** of the **process type** *CCF_SSF_PROC*. The *CCF_SSF_Monitor* is the master of this block. It is created statically when the IN is constructed. However the *CCF_SSF_Executor* **processes** are created dynamically by *CCF_SSF_Monitor* based on the call situations.

As shown in Figure 3-15, the number of *CCF_SSF_Executor* **processes** that can co-exist in the block range from zero to MAX_CONNECTIONS (maximum number of call connections a local switch can maintain simultaneously, it is a system capacity constant determined when the IN is constructed). *CCF_SSF_Monitor* creates a *CCF_SSF_Executor* **process** in two kinds of situations: (1) when it receives a notification from the CCAF that a user wants to initiate a call (when the user is idle) or establish one more call connection in addition to the existing ones (of course, with the help of a certain feature); (2) when a call connection request is coming from a peer CCF-SSF block (i.e., another local switch) to a user served by this local switch. However, only MAX_CONNECTIONS numbers of *CCF_SSF_Executor* **processes** are able to communicate with outside entities simultaneously.

In our model, one CCF-SSF block is able to handle more than one CCAF entities, which means it can serve a set of users. Further more, the CCF-SSF block supports one user having more than one call-connections (when the user subscribes to certain features). These two capabilities are very important because they must be provided in actual IN systems.

There are two IN entities included the process type *CCF_SSF_PROC*: CCF and SSF that are represented by *The_CCF* and *The_SSF*. CCF manages normal calls, and SSF monitors the processing of the call and checks for various feature-triggering conditions. In the cases when such conditions are satisfied, SSF will suspend the call processing of CCF and switch control power to SCF. However, because CCF and SSF are so tightly coupled together, we modeled them using a single **process** inside the **process type** *CCF_SSF_PROC*. But within this **process** they are distinct parts and only one of them can be active (in execution) at any specific time. We use two SDL **services** *The_SSF* and *The_CCF* to represent the SSF and CCF entities respectively in our model.



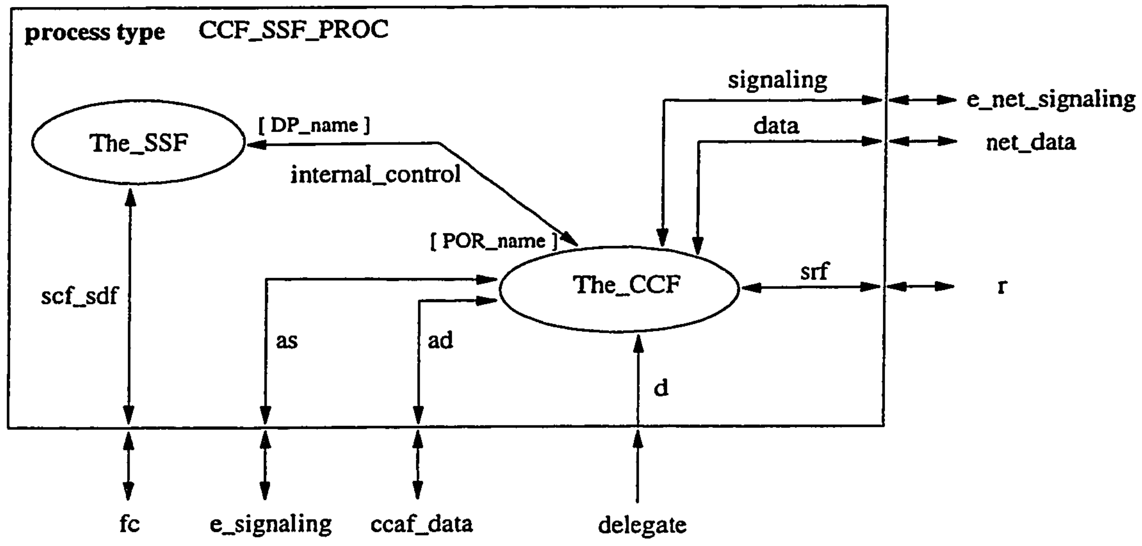


Figure 3-15: Structure of block type CCF_SSF

Referring to Figure 3-15, *The_CCF* controls the processing of a call when it is first initiated or accepted by a user served by this local switch. Whenever a DP is reached in the BCSM of *The_CCF*, it sends a *DP_name* **signal** to *The_SSF*, halts call processing and waits for *The_SSF*'s response. Upon receiving *DP_name* **signal**, *The_SSF* starts to perform feature- triggering check, and finally sends *The_CCF* a *POR_name* **signal** that contains a PIC. This PIC indicates at which call state that *The_CCF* should continue its call processing. After getting the *POR_name* **signal**, *The_CCF* resumes execution at this PIC. When another DP is reached, *The_CCF* will again communicate with *The_SSF* in the same way.

If no triggering criteria are found to be met, *The_SSF* will specify the PIC next to (in the BCSM of *The_CCF*) the DP given by *The_CCF* in the *DP_name* signal. This means *The_CCF* should continue call processing normally (same as POTS). On the other hand, if a feature is triggered, the feature will get the power of call processing. In this case, the feature will give out a POR to *The_SSF* and return the call control power to *The_SSF* when the feature stops. Then *The_SSF* will send this POR to *The_CCF* in the POR_name **signal.** *The_CCF* gets back the call control power from *The_SSF* and resumes processing the call at the specified POR.

3.2.4 SCF modeling

Service Control Function (SCF) is the core control unit of IN systems. It commands the service switch function (SSF) and call control functions (CCF) in processing the feature-related calls. Whenever a feature is triggered by the user, CCF-SSF will notify SCF of that, and SCF will take the control to perform the designated task(s) of this feature.

In our model, SCF is described using an SDL **block** *The_SCF* inside *The_SCF_SDF*. *The_SCF* consists of a few component blocks: *FEAM* (Functional Entity Access Manager), *SDAM* (SCF Data Access Manager), and *SLEM* (Service Logic Execution Manager). The *FEAM* provides the functionality needed by the other two blocks to exchange information with other outside entities (such as SDF, CCF-SSF, etc.) via a certain message passing mechanism (using **signals** in SDL). *SDAM* provides the ability of the storage, management and access of shared and persistent information (data that persists beyond the lifetime of an executing feature instance). *SDAM* is also responsible for accessing remote data that is managed by SDF. The *SLEM* handles and controls the complete course of feature logic execution. It also interacts with *SDAM* and *FEAM* to support the concurrent feature instances.

Service Logic Execution Manager

Refer to Figure 3-16, the SLEM block consists of three parts: *Feature Monitor* (FM), *Resource Manager* (RM) and *Running Feature Instance*s (RFIs). FM is a statically created **process** when the IN is built, and so is the RM. However, the RFI stands for a set of **processes**, each of which is called a feature instance that deals with the use of one feature by one user. The RFIs are created dynamically by FM.

FM is the monitor of all the concurrent feature instances. It first accepts the request from CCF-SSF that a certain feature was triggered by telephone users or certain call conditions. FM then creates an instance of this feature (as a process whose behavior is defined by the feature). FM also gathers necessary data for this feature instance to support its running. FM may access certain local data (inside SCF) or remote data (in SDF) with the help of SDAM, and FM also manages call resources through RM. *Path_FM_RFIs* are a set of non-delaying **signal routes** that allow the communications between RM and each of the feature instances. *Path_FM_SDAM* is a non-delaying **channel** through which FM can realize data access.

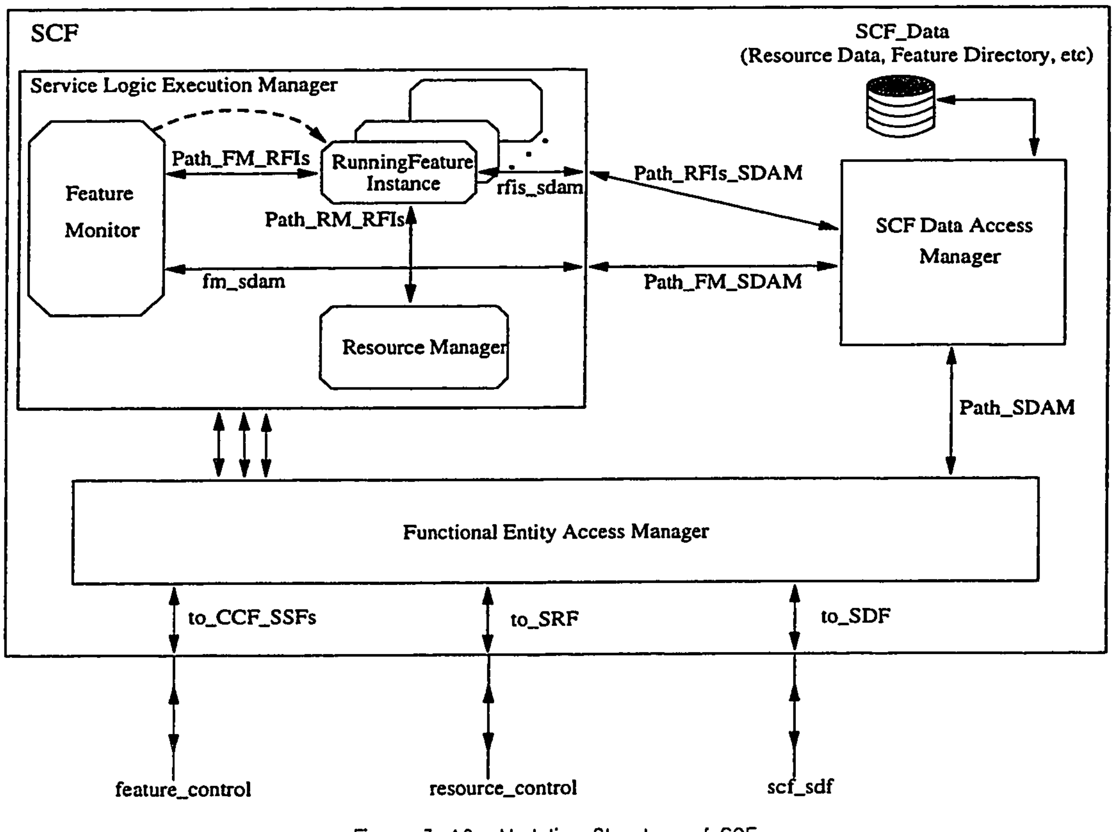


Figure 3-16: Modeling Structure of SCF

RM is responsible for providing special call resources to support the running of RFIs. *Path_RM_RFIs* represents a group of non-delaying **signal routes** between RM and each of the RFIs that allow the RFIs to control the allocation and use of special call resources during their execution. Although these resources are controlled by SCF, the CCF-SSF entities are the real users of them.

RFIs are actual call control programs that are running inside *SLEM* and under the monitoring of FM. They behave according to their own pre-defined feature logic individually without even knowing of the existence of other feature instances. They can request to allocate and control the use of call resources through RM. They control the call processing of CCF-SSFs. The communication between a RFI and its corresponding CCF-SSF entity is realized by a non-delaying **channel** linking each RFI to *FEAM* and a delaying **channel** from *FEAM* to CCF-SSF. Also a group of non-delaying **channels** *Path_RFIs_SDAM* enables the communication between each RFI and the *SDAM.*

SCF Data Access Manager

SDAM handles the data access of all elements in the *SLEM* block. *SDAM* consists of three processes that are *Data Integrator* (DI), *Remote Data Manager* (RDM) *and Local Data Manager* (LDM). Illustrated in Figure 3-17, DI accepts data access requests from *SLEM* and distributes corresponding data access action to RDM and/or LDM, assembles the results from the two and passes it back to *SLEM.*

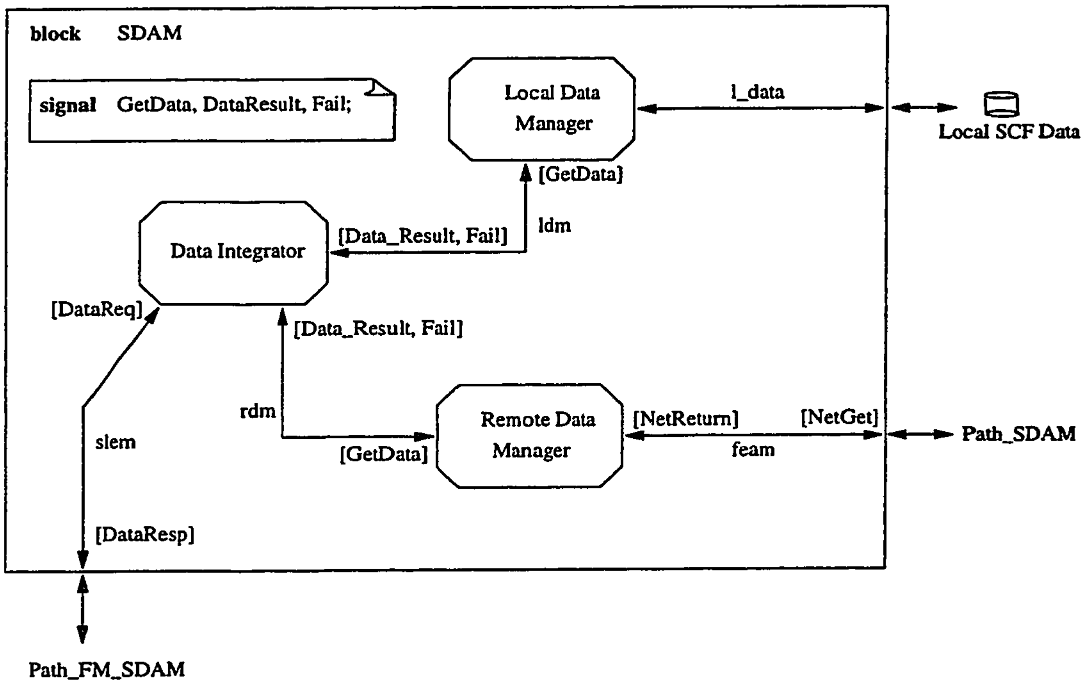


Figure 3-17 Construction of SCF Data Access Manager

Functional Entity Access Manager

As shown in Figure 3-16, *FEAM* serves as the interface of *SLEM* and *SDAM* to the outside world. It has been connected to CCF-SSF, SRF and SDF though different paths that are specified using SDL delaying channels. These channels are bi-directional and may attach a random time delay to each signal transmitted through them. However, the sequence of the signals transmitted is preserved. All channels connected to SCF are fundamentally signaling channels that are meant to transfer control information about how to continue processing the feature-related calls.

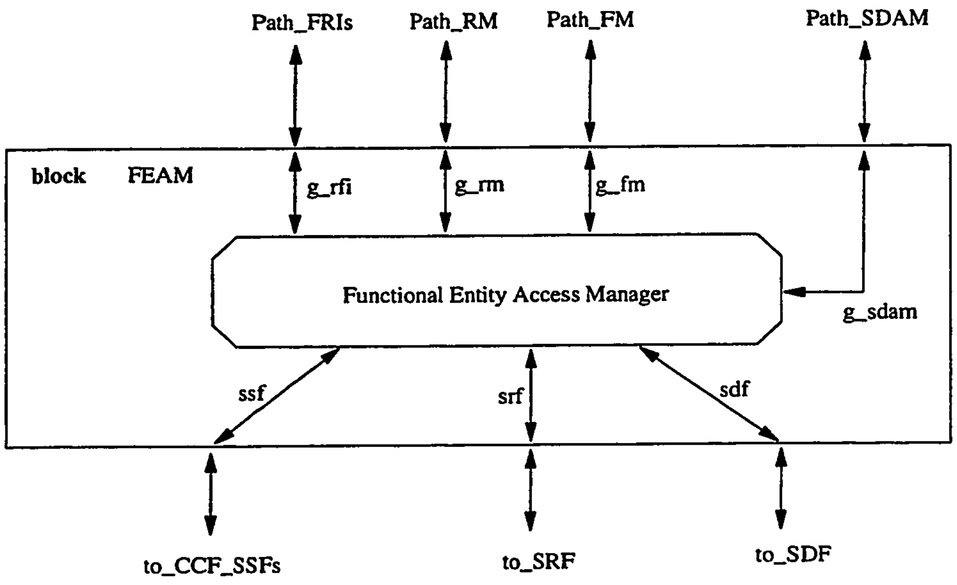


Figure 3-18: Facade of SCF -- Functional Entity Access Manager

FEAM contains a single process that deals with the above tasks, as demonstrated in Figure 3-18. *Path_SDAM* is a bi-directional non-delaying channel that connects *FEAM* and *SDAM*. Whenever *SDAM* needs to access data in SDF, this path is used to send the request and get the response. *Path_RM* is a bi-directional non-delaying channel that links *SLEM* with *FEAM*. It is mainly used for sending control messages from RM to SRF concerning the utilization of call resources. However, response and statistical information can be sent back along this channel. *Path_ FM* represents another non-delaying channel between *FEAM* and *SLEM*. This channel connects Feature Monitor and is used to transmit signals between FM and CCF-SSF. The feature-executing requests (when CCF-SSF receives a feature-triggering event) are transmitted via this channel.

There is another bundle of non-delaying channels connecting *SLEM* and *FEAM* that are used for communications between RFIs and the corresponding CCF-SSFs. These channels are also bi-directional and carry the signals that control the behavior of CCF-SSFs.

3.2.5 SDF Modeling

Service Data Function (SDF) is responsible for providing necessary Service Support Data (SSD) and Call Instance Data (CID) to support the running of all the feature instances in the SCF unit. Also it may accept to store some CID when requested by the SCF.

Inside the SDF block, shown in Figure 3-19, there are three **processes**: *SDF Data Manager* (SDF-DM), *Data Request Checker* (SDF-DRC) and *SDF Functional Entity Access Manager* (SDF-FEAM). SDF-DM provides the functionality needed for storing, managing and accessing information in the SDF. For example, if the data are physically structured as a database, the SDF-DM may also handle a database accessing language such as SQL. The SDF_DRC checks for the validity of data access requests. This includes access privilege verification and data availability screening. The SDF-FEAM provides necessary interfacing between SDF-DM and outside world. This design means to hide the unnecessary communication details with other IN entities.

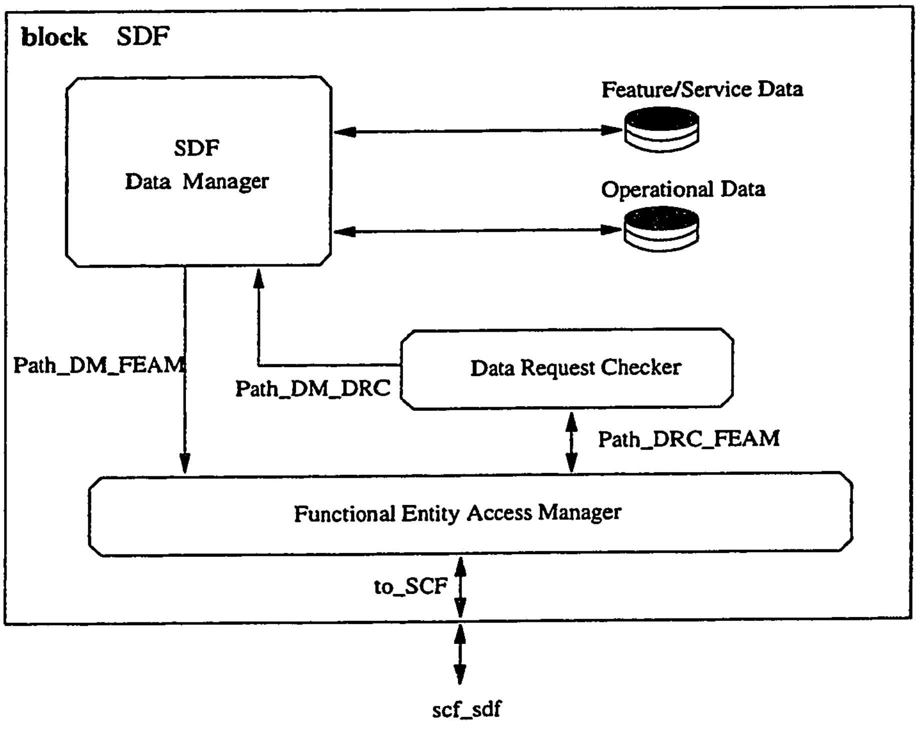


Figure 3-19: Illustration of SDF Model

The *Path_DRC_FEAM* is a bi-directional non-delaying **signal route**. However the *Path_DM_DRC* and *Path_DM_FEAM* are both unidirectional non-delaying **signal route** transferring verified data requests from SCF (through SDF-FEAM) and the response data to FEAM and then to SCF. FEAM is linked to SCF through a delaying **channel**.

Two kinds of data exist in the SDF entity: *Service/Feature Data* and *Operational Data.* However, we do not elaborate describing how these data are organized and accessed because we focus on feature interaction analysis rather than feature data access.

- Service/Feature Data -- Used for the provision of features. Usually accessed by SCF during the running of feature instances. Examples include a subscriber profile, pre-specified general feature parameters, etc.
- Operational Data -- These data are not used by RFIs in SCF but by the SDF-DM itself for operational and administrational purposes. Examples of such data are references to a data object and access control information.

3.2.6 SRF Modeling

Specialized Resource Function (SRF) is the entity that provides specialized call resources to support the execution of various feature instances. Utilization of the resources inside SRF is under the control of *SLEM* inside SCF and these resources are actually used by CCF-SSF to perform corresponding functions. Examples of special call resources include the following [Q.1214]:

- DTMF receiver: This resource receives Dual-Tone-Multi-Frequency message from a linked resource, and recognizes it as a standard signal input.
- Tone generator/announcements: TGA provides in-channel information to the specified resource.
- Message sender/receiver: Sending or receiving messages, such as electronic message, voice message. etc., to/from users.
- Speech recognizer: Receiving in-channel speech information from a linked resource and recognizing it as a standard input.
- Audio conference bridge: ACB receives in-channel audio information from any linked resources, mixes this information together and sends it to all the linked resources.

Shown in Figure 3-20, SRF is composed of three parts: *SRF Resource Manager* (SRF-RM), *SRF Functional Entity Access Manager* (SRF-FEAM) and *the Specialized Resources* (SR). SR represents the resources as a whole; SRF-FEAM provides the necessary functionality to exchange information with other IN entities and delivers the designated functions of SR to their corresponding clients. SRF-RM serves to offer the ability to manage and use resources contained in SRF. This includes the capability to allocate a resource, to obtain the status of a resource, and to control the action of a resource, etc.

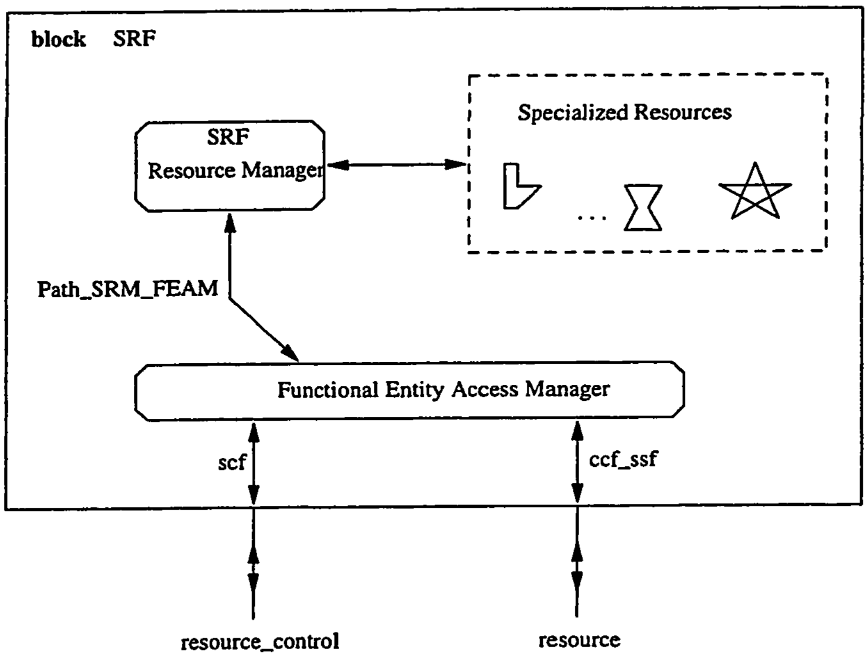


Figure 3-20: Structure of Specialized resource Function

3.2.7 Paths of Data and Signaling

After introduction of how we model the IN entities that are directly involved in feature-triggering and execution, we want to clarify the usage of the communication paths among these entities.

Please note that the data (voice, text, image,...) transmission is only performed among the users and CCF-SSFs as well as among the CCF-SSFs themselves. This means the data channels exist only between the CCAF and CCF-SSF, between different CCF-SSFs, and between CCAF and the its user. The channels between CCF-SSFs and SCF are signaling channels that carry control messages, these control messages determine how the data channel between peer CCF-SSFs works. The same is true between SCF and SRF as well as between SCF and SDF. The data transmission and signaling paths are shown in Figure 3-21.

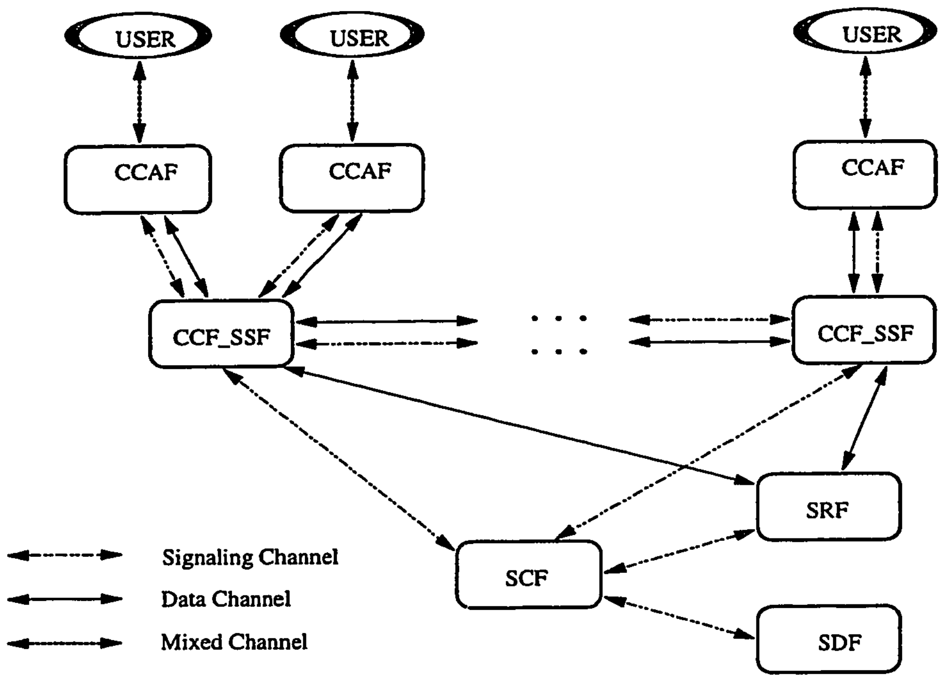


Figure 3-21: Data channels and Signaling channels

3.3 Features in IN

In this section we explain how features are modeled. Features in IN systems are extra added call functions provided to telecommunication users as separate units of commercial offerings on a subscribe-and-use basis. Technically speaking, they update or add certain call processing functions to the existing basic call processing functions in the IN systems. This allows a higher level of network utilization and achieves better communication ability and performance.

According to ITU-T's IN recommendations, different IN entities are responsible for different aspects of tasks in an IN system. SCEF (Service/feature Creation Environment Function) is in charge of feature creation and testing; SMF is responsible for feature deployment that allocates functional software of the features into corresponding network devices, and feature management that controls the activation/deactivation and some general context of the features; SDF provides and stores necessary data during the execution of features; SCF actually executes the triggered features and controls the behavior of CCF-SSF and SRF; SSF serves as switching unit between the basic call processing and feature execution; CCF performs basic call processing functions and triggering checking for the features. SRF will allow the usage of various specialized call resources; CCAF is the interface between the telecommunication user and the IN system.

Now that our focus is on feature interaction detection, some of the IN entities are not of interest (such as SCEF, SMF). We emphasize on the following issues:

- Location of Features and Feature Instances
- Modeling Feature, SIBs and GSL
- Modeling SSD and CID

3.3.1 Location of Features

Similar to the modeling for IN system itself, the major decision of modeling features inside IN systems is based on how we perceive the working mechanism of the features.

We have noticed that one of IN's major ideas is to separate the processing responsibility of newly introduced features from that of the basic call processing functions. Different network devices are used to deal with the two kinds of tasks. Using this mechanism, new features can be included into IN systems efficiently because no significant modifications need to be applied to large number of switching devices. Based on this idea, features need to reside in certain specialized intelligent unit(s) of the IN system rather than being "patched" on switching nodes.

We have chosen to allocate the IN features into SCF (service control function) entity that is responsible for feature control and execution according to the ITU-T recommendations. Please recall that there is a *Service Logic Execution Manager* (SLEM) block inside the SCF block in our IN system model. Also, the process *Feature Monitor* is responsible for controlling the selection and generation of the feature instances that actually carry out the tasks designated for the features. Figure 3-22 illustrates how we model the features inside the SCF entity.

Inside the *SLEM* block, there are two permanent processes: *Feature Monitor* (FM) and

Resource Manager (RM). Both the FM and the RM are created when the whole SDL block is constructed, i.e., when the IN system is implemented in real life. FM accepts feature execution requests from CCF-SSFs, and chooses the correct feature type to create a new feature instance process running this feature to serve the corresponding CCF-SSFs.

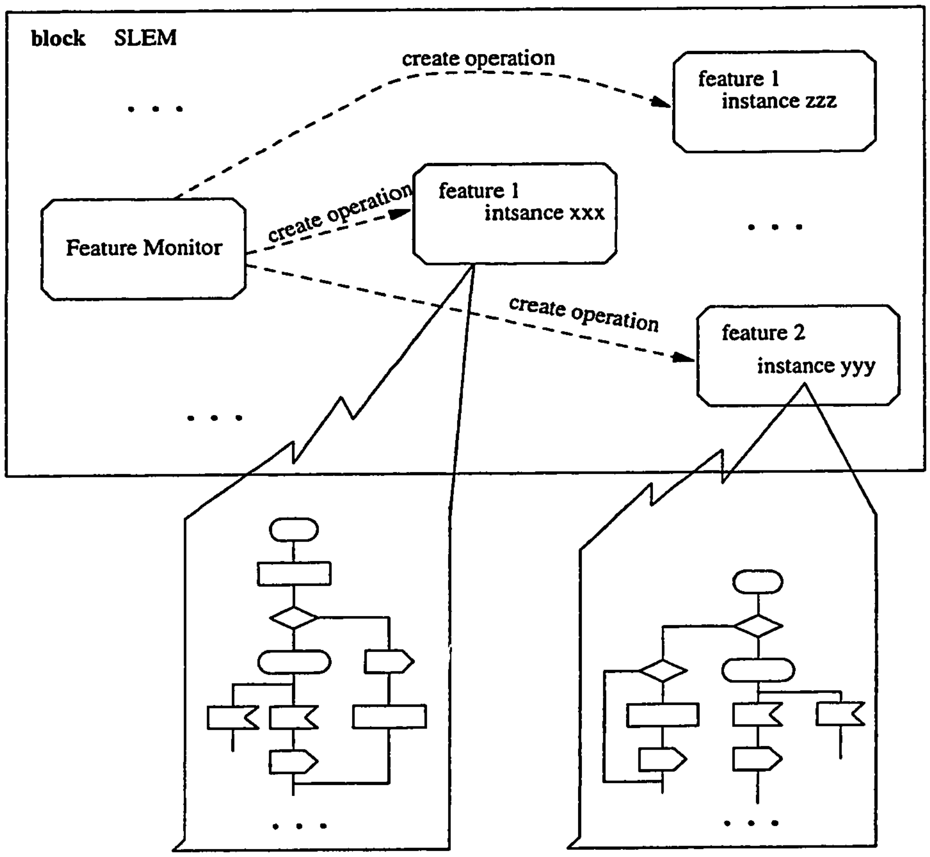


Figure 3-22: Modeling Features as SDL Processes

Before creating a feature instance, FM may need to prepare certain data. This is done with the help of SCF Data Access Manager (SDAM). After the creation of the new feature process, *Feature Monitor* returns to its idle state and continues waiting for other feature execution requests. The newly created RFIs (running feature instances) will actually deal with all the designated tasks of their corresponding IN features.

3.3.2 Modeling Feature, SIBs and GSL

Each kind of feature is specified as SDL **process type** in our IN system model. These process types are used to create actually running processes (i.e., feature instances) in SCF when users or certain call conditions have triggered the corresponding features. The fully specified **process type** definition is equivalent to the GSL in INCM.

Please recall that in Chapter 2, we have stated a feature can be designed based on a standard set of exiting SIBs. These SIBs are usually used to perform a variety of call processing actions, and parameters associated with these SIBs can be used to specify the actual context of the SIBs to customize them. A feature is mapped into a GSL in the GFP of INCM. The GSL specifies what SIBs are used to fulfill the tasks of this feature, how the SIBs are interconnected, and what are the Point Of Initiation (POI) and Point Of Return (POR).

The SIB-and-GSL mechanism of feature is implemented in our IN model as follow:

- Each kind of SIB is defined as an individual SDL **service type.** The service types are used to create SDL service instances that are parts of certain **process type.** The definition of process type indicates what SIBs are used and how they are chained together.

- The POI and POR are represented as in-coming and out-going SDL **signals** that are transferred by the **signaling route** connecting the **process type** and its environment (the *SLEM* in SCF). Both the POI and POR are of a standard PIC (Point In Call) **data type**, which represents different call states in the Basic Call System Model (BCSM) of the BCP.

3.3.3 SSD and CID

Two kinds of supporting data are needed during the creation and execution of a feature instance: Service Support Data (SSD) and Call Instance Data (CID). They must be provide to the feature instances in order for them to run correctly.

- SSD is data that has a static nature. SSD denotes the information that is used for a specific feature. It varies with different features but is common for all feature instances of the same feature.

- CID is the data that has a dynamic nature. CID represents the information that is not available when the feature is designed, it can only be obtained when a feature instance is actually triggered into execution. This type of data depends on each specific call that triggers the feature instance.

As shown in Figure 3-23, both the SSD and the CID are provided to the feature instances by the Feature Monitor (FM) inside the SLEM of SCF entity. However, FM obtains these two different data from two different sources. SSD is requested and received from the SCF Data Access Manager (SDAM) from local data source, while CID is obtained from the CCF-SSF entity that requested the execution of the feature.

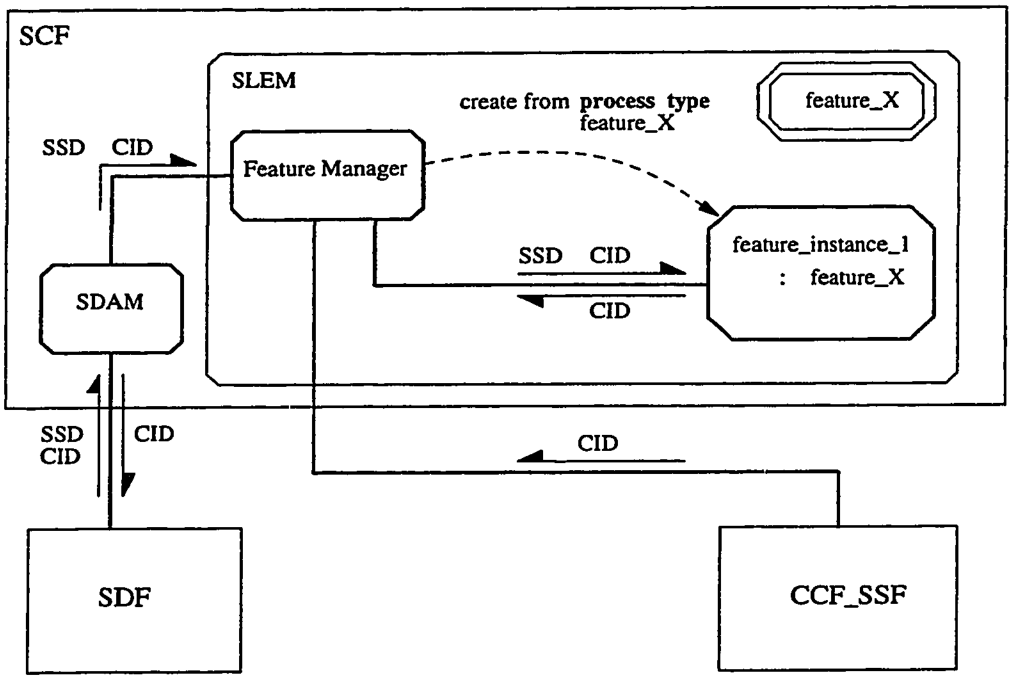


Figure 3-23: Provision of SSD and CID

3.4 Comments about SDL in IN Modeling

According to our own experience in using SDL, we feel that the following SDL characteristics are especially useful in representing the structure of IN system as well as the relationships and behaviors of IN entities:

- Unlimited data type definition ability: SDL provides strong support for user-defined signal and data types based on a set of built-in standard types. This promotes the design of flexible signaling and data processing, and greatly facilitates the description of various network communication protocols.
- Complete modeling capability: SDL is able to describe the target system thoroughly

and in both the static and dynamic aspects. The static aspect refers to the constructional information of the whole system and is captured using SDL structure concepts like **system, block** and **process**. The dynamic aspect covers the functional behavior of the entities inside the system and is expressed using extended finite state machine (EFSM) model.

- Non-deterministic behavior description: When modeling IN system, we sometimes need to describe non-deterministic behavior of certain entities, e.g., the possible (neither "never" nor "always") loss or damage of data packets during the transmission across the network. SDL provides random decision and spontaneous transition mechanisms to precisely capture such behavior.

- Transmission delay modeling: The time delay of data transmission across the network usually fluctuates from time to time. However, when data is transmitted among entities that are physically located together, almost no delay is perceived. SDL provides both randomly delaying data path and non-delaying data path to reflect these two different situations.

- Priority input signals: For certain network entities, some input signals may be more significant than others, e.g., network management signaling is more important than the signaling of setting up a data transmission channel. SDL offers the ability to specify priority input and normal input levels for different input signals. When multiple signals arrive at a network entity, they are queued according to their arriving order but the higher priority one will be processed first.

- Continuous enabling conditions: SDL provides the concept of enabling condition, which means any input or output or state transition can be guarded by a preset condition. When this condition is met, the action (input/output) or transition is allowed to be performed, otherwise the condition will be checked repetitively. This is especially useful to specify the conditional behaviors of some network entities.

- Signal saving mechanism: In a certain processing stage (represented as a "state" in EFSM) of a network entity, some input signals may be irrelevant. However, they may be very meaningful for another processing stage. SDL allows to describe the behavior of saving them aside for later processing while waiting for the currently interesting signals to arrive.

Generally, SDL is very powerful in describing a telecommunication system that is concurrent, real-time, interactive and geographically distributed. However, we did find one issue in SDL that are not good enough in specifying IN systems. This is the SDL "**service**" concept that is intended to represent different logic parts that constitute a **process**. Our criticism is based on the following considerations:

- SDL **service** is introduced as a constructional element of SDL process. However, they are restricted not to be active at the same time. Only one of the **services** in a **process** can be executed at any time. This is completely contradictory to the principal idea of SDL that each individual element has it own independent behavior and only interacts with other elements through sending signals via the communication paths that interconnect them.

- The connections of SDL **services** are not of logical meaning. Different **services** communicate via the inter-service **signal routes**, which is the same mechanism as that used for inter-process and inter-block communications. Thus the concept "**service**" cannot precisely express the situation where a **process**'s behavior is composed of a few logic components (not constructional components).

- We had difficulty in describing the how sub-features are defined using SIBs (service independent building-block) in the IN system because of the lack of logic component concept in SDL. Although the SDL **service** was reluctantly adopted to describe the SIBs that are essentially program modules linked together to form a sub-feature, it is not convenient and needs some tricky measures (such as sending signals among the **services** to imitate the logic connection relationship among them).

In general, we believe that SDL is very suitable for expressing telecommunication networks including the new IN-structured networks. Although it is not perfect, few other formal description techniques and tools can match SDL's description capability and performance in telecommunication domain. Also the CASE tool for SDL - ObjectGeode is very powerful, which is a strong support for choosing SDL as the tool for specification, design of telecommunication systems.

3.5 Chapter Summary

In this chapter, we first presented an introduction of the ITU-T standardized formal description tool: Specification & Description Language (SDL) including its major characteristics and concepts. We have shown how it can be used to effectively model target systems with regard to its construction, behavior, and inter-module communication.

We described how the Intelligent Network architecture and various Intelligent Network functional entities are modeled using different kinds of SDL concepts and constructs. Our modeling is focused on the Distributed Functional Plane (DFP) of Intelligent Network Conceptual Model (INCM). Major IN entities involved in feature-triggering, feature execution are covered, including CCAF, CCF-SSF, SCF, SDF and SRF. We introduced their structures as well as their relationships.

Also the mechanism of modeling features is presented in this chapter. We have described how a feature works inside the Intelligent Network system, including feature instance modeling, feature construction, and provision of feature supporting data.

Chapter 4 Describing Feature Properties using CCM and FCA

Research and development of features are becoming an important work in telecommunication community because adding new features to existing networks seems to be the only feasible approach of system functional enhancement. In order to study features effectively, we must first be able to describe them effectively. Although many features have been introduced and widely used, the methods and tools that can precisely describe the features for FI study purpose still need further improvements.

Inadequate feature description tools also have impact on the effective description of FIs and thus affect the efficiency of FI prevention, detection and resolution. As mentioned in the previous chapter, Intelligent Network framework has made it much easier to develop and deploy large number of features into the telecommunication networks. However, there is no guarantee that those features will work together correctly unless FI problem has been addressed to a satisfactory degree.

In this chapter, we present two key concepts that constitute the basis of our new FI detection approach. The *Call-Context Model* (CCM) is intended to characterize a feature by means of the call connections in which it is involved. The *Feature Context Assumption* (FCA) aims to reflect those essential requirements that a feature assumes from the running environment in order to perform its tasks successfully.

4.1 Features' Call Context Model

Although there are different methods to describe features, the purpose of them is the same: providing accurate and unambiguous descriptions of the features at an appropriate abstraction level. Different feature-description methods are suitable for different feature-development stages and aspects. CCM is meant to outline how a feature works globally in a precise way. In this section, we first introduce existing feature description methods and then demonstrate our idea of CCM and how to characterize features using CCM.

4.1.1 Existing Feature Description Methods

Natural Language (NL) has been a major tool to communicate the intention and working scenarios of the features. ITU-T has been using plain English text to describe features in its official recommendations, which are considered as international standards (see [Q.1211]). Natural language is also frequently used by researchers to explain call scenarios that produce feature interactions (e.g. see [Cameron 93]). However, there are a few drawbacks with NL description, including the ambiguity and incompleteness that are usually associated with it. The following italic-text segment gives an example of NL description of a feature Abbreviated Dialing (ABD).

> *Feature:* ***Abbreviated Dialing (ABD)*** *[Q.1211]*
>
> *This feature is an originating line feature that allows business subscribers to dial others in their company using, e.g., only four digits even if the calling user's line and the called user's line are served by different switches. This capability extends switch based intercom calling beyond the switch boundary. Typical (feature-using) scenarios <u>might include</u> :*
>
> *1) Caller A (location A) dials extension number of called B (location B) and the network connects the call.*
>
> *2) Caller A forwards his line to called B (different location) using B's extension number. Caller C calls A and is forwarded to B.*

The underlined "might include" seems to mean "may or may not include", which is an indication of ambiguity. How the network distinguishes a four-digit extension number from the first four digits of a normal line number is also NOT described in the above definition, which suggests its incompleteness. NL is usually used for describing features only at the requirement stage in order to make them easy to understand for feature users.

Message Sequence Chart (MSC) is another tool that can be used to show the working scenarios of features. MSC descriptions help to understand how features react to certain call conditions or call events and how the POTS call functions are modified by the features. [Gupta 96] explains how features can be expressed in MSC and how feature interactions can be reflected using MSC descriptions. His result shows MSC is good at describing the messages/signals exchanged between network entities or between network entities and users. However, MSC is weak in showing the control logic inside these network entities.

Figure 4-1 and 4-2 (adopted from [Gupta 96]) illustrate two MSCs that depict the features Call Distribution (CD) and Call Forwarding (CF). We can see that the two MSCs look alike (only three places where they are different), especially when we only compare the exchanged signals between network and the users. However, they are indeed quite different features as their text descriptions indicate.

Feature: ***Call Distribution (CD)*** *[Q.1211]*

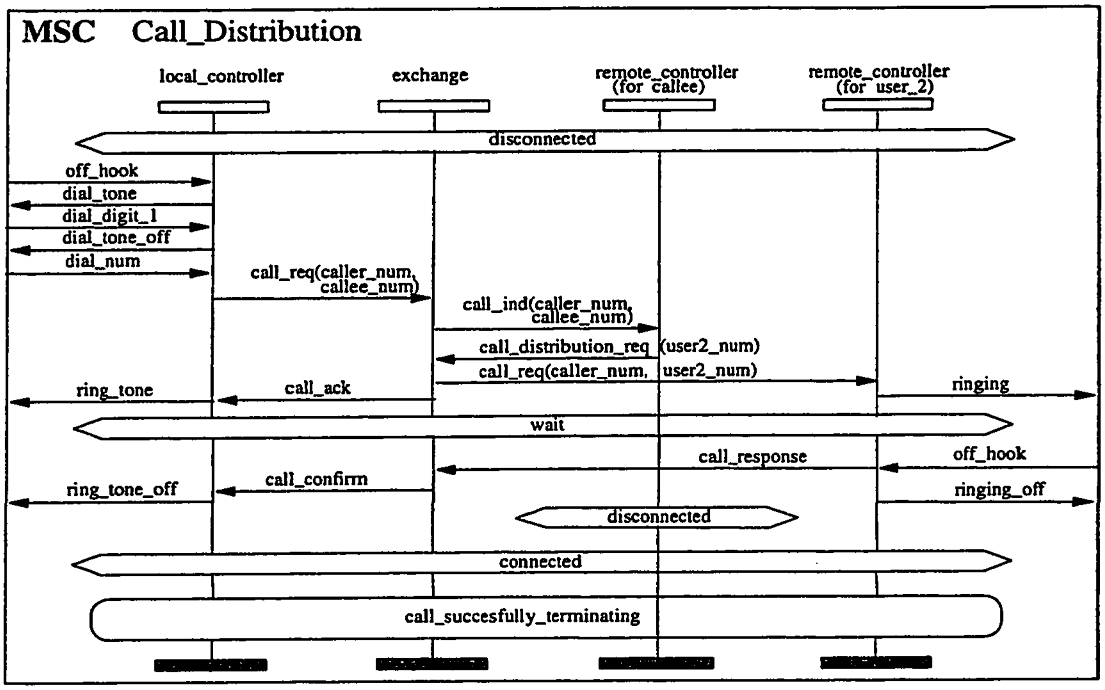
Call Distribution allows a subscriber to have incoming calls routed to different destinations, according to an allocation law that may be real-time managed by the subscriber. Three types of law may exist:

- *circular distribution: calls are routed to different locations with a uniform load;*
- *percentage distribution: calls are routed according to a percentage setting;*
- *hierarchical distribution: destinations are chosen based on the priority list.*

In addition, congestion at one location may cause overflow calls to be routed to an alternate location. [Q.1211]

Feature: ***Call Forwarding (CD)*** *[Q.1211]*

Call Forwarding enables the called user (a CF subscriber) to forward in-coming calls to another telephone number when this feature is activated. All calls destined to the subscriber's number are redirected to a new number designated by the subscriber no matter what the subscriber's line status is. However, the originating function of the subscriber is not affected.



MSC Call_Distribution
local_controller
exchange
remote_controller (for callee)
remote_controller (for user_2)
disconnected
off_hook
dial_tone
dial_digit_1
dial_tone_off
dial_num
call_req(caller_num, callee_num)
call_ind(caller_num, callee_num)
call_distribution_req (user2_num)
call_req(caller_num, user2_num)
ring_tone
call_ack
ringing
wait
call_response
off_hook
ring_tone_off
call_confirm
ringing_off
disconnected
connected
call_succesfully_terminating

Figure 4-1: MSC description of feature Call Distribution

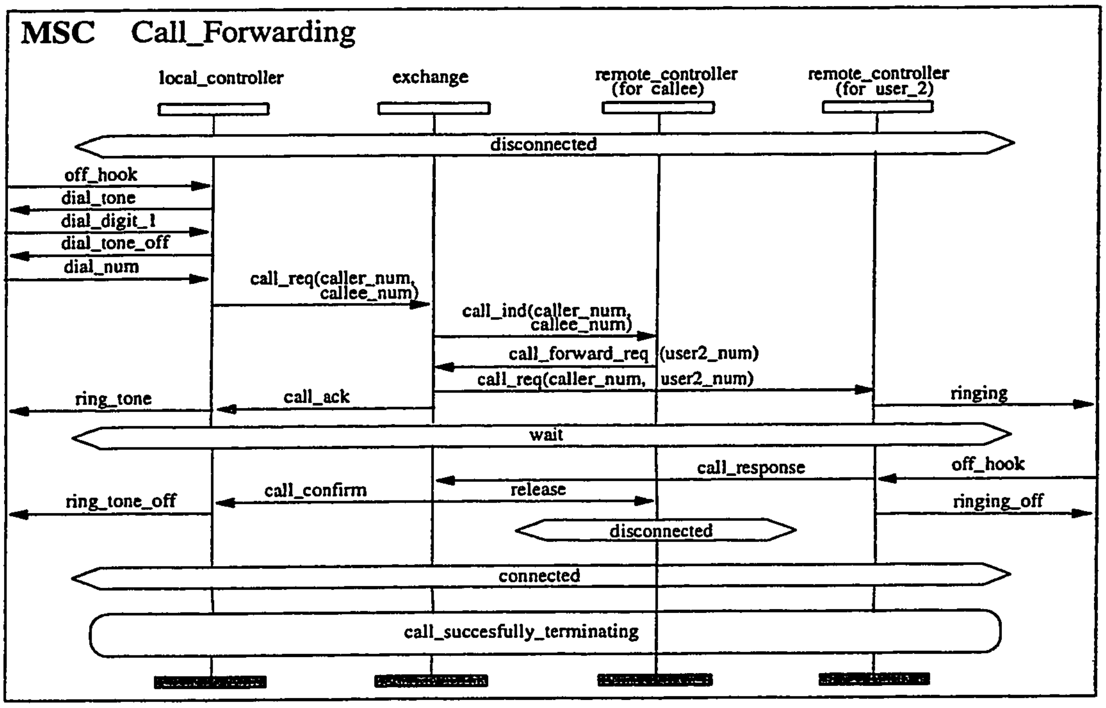


Figure 4-2: MSC description of feature Call Forwarding

Because the focus of the thesis is feature interaction detection, we cannot endure the ambiguity and incompleteness in feature description associated with natural language method. We need to analyze how a feature works as well as the signal exchanges that are considered as external characteristics. Consequently, both Natural Language and MSC are not suitable for our study.

Readers may ask "how about the SDL you used to model the IN ? ". Yes, SDL is a very good tool to describe IN including the features running inside it. However, such a description provides us with the whole volume of information about the features, and is often of large size. We think this is not good enough for FI detection purposes.

4.1.2 Principles of Call-Context Model (CCM)

Many existing feature-describing methods are focused on network-and-user type models. These models are essentially dedicated to represent and analyze the relationship and signal exchanges between network users and network entities (see [Bryce 94], [Ohta 94], [Bergs 97], [Turner 97]). The emphasis is on what the network does, what the user does and what other call users do before, during and after a feature is triggered into running. In such models, the behavior of the whole network (with respect to processing of this feature-related call) is viewed to be controlled by this feature, as shown in Figure 4-3.

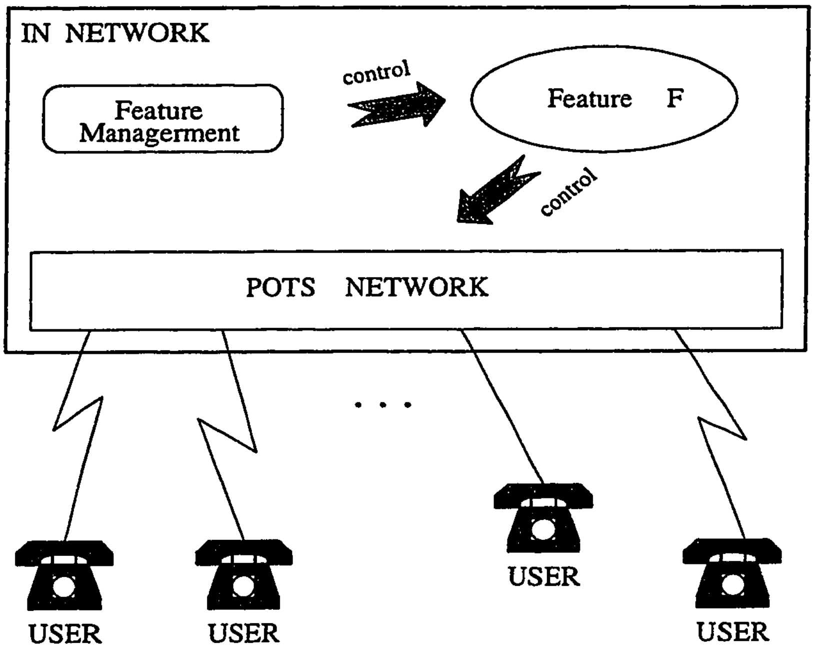


Figure 4-3: A network system (with feature F) as a whole

We think there are two major drawbacks with such models:

- They are not precise enough because actually only part of the network is under the control of a feature instance. So they are not suitable for analyzing multiple features simultaneously in such models.
- These models have insufficient capability to describe features that involve more than one call connection. With this kind of feature, one call party may play different roles in different call connections. However, the network-and-user type models cannot distinguish these roles because the network is viewed as a whole and the different call connections inside the network are invisible.

Although a telecommunication feature is always tightly related to its subscriber, it is more important to study features with regard to specific call-contexts that are essentially call connection(s) initiated or accepted by feature users. This is because the feature's tasks are done by altering the normal call processing functions inside each of the call connections. The feature applies its impact of providing extra call processing functions to feature users indirectly. Users can play a role in the call connections only after these connections have been established.

Call-Context Model is presented in order to characterize the inherent relationships between features and the call connections. Each call connection is defined as a call-context that enables the communication between two parties (a multiple-party call can be viewed as a set of two-party calls that are connected at the party who subscribes and invokes this multiple-party call).

A feature, by nature, can be involved in any number of call-contexts, although few features have more than three call-contexts. From the system's point of view, a running feature (a feature instance) will affect the execution of the BCSMs (Basic Call System Model) in each call-context that the feature is involved in, and thus control the processing of these call connections. The behavior of the feature also partially depends on the conditions (such as call-state, call data, signals/events) in all these call-contexts that represent the overall call situation. Figure 4-4 indicates the context-oriented idea.

Each of the call-contexts can be viewed as a separate POTS call processing process behaving based on BCSM. The feature can be viewed as a supervisory process that controls all the call-contexts. It has the power to modify the normal POTS functions. Thus the users' communication ability and style are decided by the features based on the features' designated tasks.

CCM allows us to express a feature's call-connection(s) clearly, which makes it easier to study and compare different features and also facilitates the detection of interactions among them. Also quite important, CCM allows different levels of abstraction. You can outline the feature or the feature instance at the most abstract level that only specifies the subscriber, call parties and call-contexts as well as at very concrete level where features' triggering and detailed behaviors are fully specified without ambiguity.

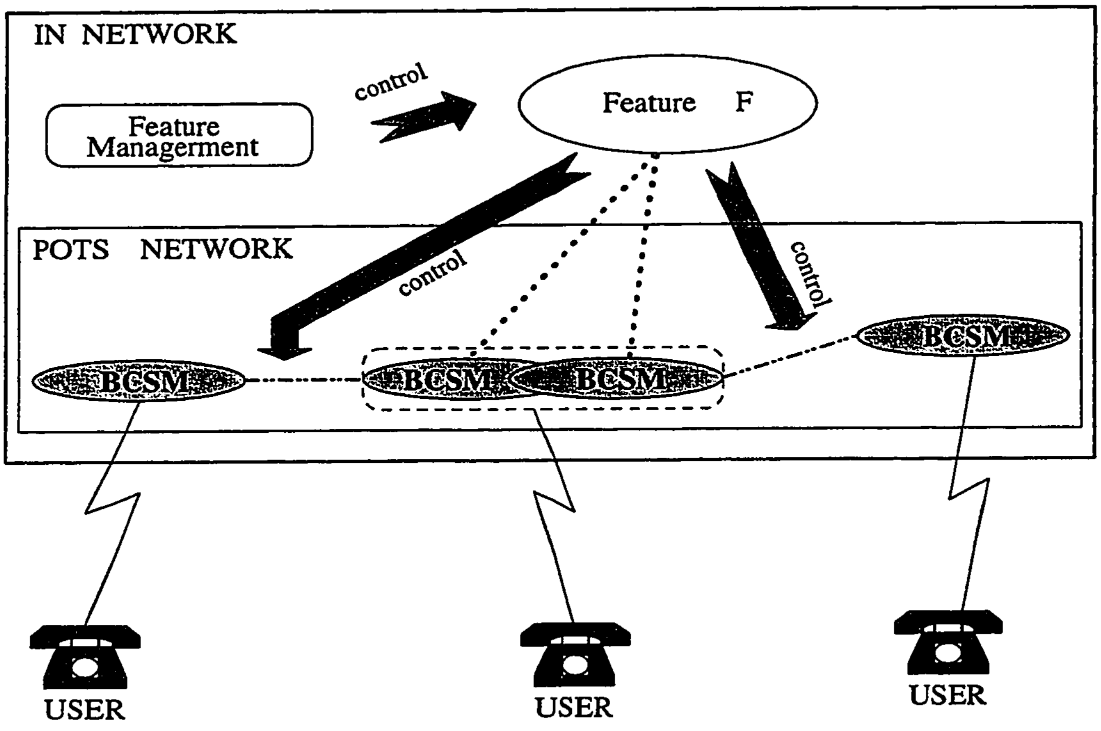


Figure 4-4: Network with feature-controlled call-contexts

4.1.3 Principle of CCM

It is important to distinguish between a "feature" and a "feature instance". The former means one kind of extra telecommunication functions that are offered to users by the network operator in a single commercial unit. The latter refers to an actual running process that performs the functions defined in the former. Intuitively, a feature to its feature instance(s) is like a program on the hard disk to a process running this program.

CCM of Features

In CCM, a feature is a static network control logic that, once triggered into running as a feature instance, works partially based on the call situations in each of its call-contexts. The most significant data about a feature is the call-contexts associated with it, to each of which the feature can apply its impact.

A feature is represented as a *tuple* in CCM:

> ***Feature_Name (S, {caller_1 -> called_1}, {caller_2 -> called_2), ...{caller_n -> called_n})***

Obviously **Feature_Name** denotes the name of the feature. All the symbols inside the pair of parenthesis further describe the feature. The first element **S** stands for the subscriber or authorized user of the feature, i.e., whom should the feature serve for?

The second element ***{caller_1 -> called_1}*** represents the first call-context of the feature. This element is always there since every feature will at least work in one call-context. The remaining elements starting from ***{caller_2 -> called_2}*** are other relevant call-contexts that may be established later during the running of this feature. They are optional because the number of call-contexts that a feature may be involved in depends on the tasks of this feature.

For each of the call-contexts, there must be one caller user who initiates the call and one called user who accepts it. Thus in the call-context ***{caller -> called}***, the "**->**" denotes the relationship from the caller user to the called user.

Please note that the subscriber or authorized user **S** of this feature always takes part in each of the call-contexts and acts as either caller or called party.

Let us look at an example feature: **Call Waiting** (CW) [Cameron93]

> *"When a call attempts to reach a busy line (and user of this line subscribed CW), Call Waiting generates a call-waiting tone to alert the called party, ..,(the subscriber can use flash-hook signal) to accept the connection attempt from the new caller while putting the current conversation (together with the other party in this conversation) on hold."*

With CW feature, there are two call-contexts: the first is the existing call between CW subscriber **x** and another user **y**, the second is a new call from user **z** to **x** when **x** is talking with **y**. We represent CW's call-contexts as follow. There are two forms because CW can work for its subscriber in either of the two cases.

CW (x, { x->y }, { z->x }) AND *CW (x, { y->x }, { z->x })* where

CW	Name of the feature "Call Waiting".
x	The subscriber of this feature.
y	The user that acts as the other party in the first call context of **x**.
z	The partner of **x** in the second call-context.

x, *y* and *z* are of type "telephone number", "*x->y*" means a call-connection made by *x* to *y*. "First call-context" means the first call-context established for a subscriber (here is *x->y or y->x*). "Second call-context" is the one set up next, and so on.

In the first case, *x* is the caller party in the first call-context. Also **x** can be a called party in this call-context as indicated in the second case. However **x** can only act as a called party in the second call-context.

By expressing a feature like this, the relationship between subscriber and the feature is quite clear. Also, other parties involved in this feature and their roles are specified in CCM without ambiguity.

CCM of Feature Instances

A feature instance is an invocation of the feature. In order to outline a feature instance, the CCM of the feature is concretized by replacing the "formal parameters" with "actual parameters" that are specific users' telephone numbers.

In the above example of Call Waiting feature, we assume *x* whose line number is **222-3344** subscribes CW, when *x* is talking with user *y* (**333-4455**) in a call connection initiated by *x* to *y*, user *z* (**444-5566**) calls *x*. Thus a CW feature instance is triggered to serve *x*, it can be represented as follow:

CW (222-3344, { 222-3344 -> 333-4455 }, { 444-5566 -> 222-3344 }) (F1)

where

{ 222-3344 -> 333-4455 }	is the first established call-context.
{ 444-5566 -> 222-3344 }	is the second call-context.

Another example of possible invocations of CW feature that *x* (222-3344) subscribes is shown below:

CW (222-3344, { 555-6677 -> 222-3344 }, { 666-7788 -> 222-3344 }) (F2)

Let us assume that *y* also subscribes CW, then one invocation of *y*'s CW could be:

CW (333-4455, { 222-3344 -> 333-4455 }, { 777-8899 -> 333-4455 }) (F3)

A feature in a telecommunication network may have arbitrary number of feature instances at any specific time. However, they have no relationship at all with one another as long as there is no common call-context (defined below) among them.

> *A call-context is a* ***common call-context*** *of two feature instances if they contain at least one pair of call-contexts that have same call parties and the call parties play the same roles (caller or called) in this pair of call-contexts.*
>
> *e.g.. The two CW feature instances (F1) and (F2) have no common call-context. However (F1) and (F3) have one common call-context:{ 222-3344 -> 333-4455 }.*

We will show in Chapter 5 of this thesis that having at least one common call-context is a prerequisite for two feature instances to produce FI during their running in the telecommunication network.

Behavior Description

If the above descriptions about the features are not detailed enough, we can continue specifying the behaviors of the features, which is called the feature behavior description. This is done by refining the CCM of the features.

In this case, a feature is considered to be a control logic that supervises one or more call-context(s) and decides how to process each of the calls that corresponding to each call-context. We have used CCM to characterize all the call-context(s) that may be involved in each kind of feature, and the call users in these call-context(s). This allows us to describe feature behaviors by means of specific elements (such as call states, signals, user input) inside these call-context(s).

There are different levels at which an element can be addressed in a feature property description. The highest level element in CCM is the feature, then call-context(s), call parties and call data, BCSM call states, DPs and call actions, the lowest elements are user input signals and events. Figure 4-5 depicts the relationships among them. A lower level element belongs to one and only one higher level element.

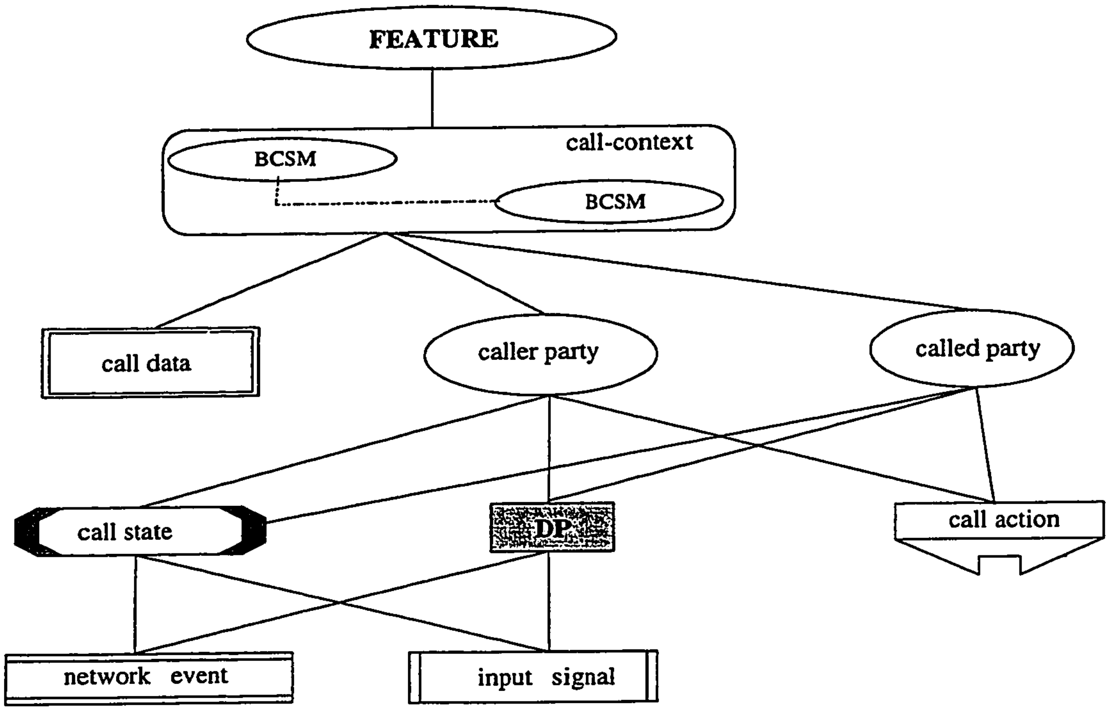


Figure 4-5: Different Levels of Elements in Property Description

When we address an element, all its ancestor elements along the path to the root element (feature) must also be specified. The symbol "::" is used to separate elements at different levels. The element at the right side of a "::" is said to be in the scope of the element at its left side. For example the phrase

feature_f :: call-context2 :: caller x :: O_Active :: Star_Key

refers to the event of

> *"*" key pressed by the user x who is a caller in feature_f's second call context when the basic call processing of x is in the O_Active call state of its BCSM model.*

Using such CCM concepts, almost every entity (caller or called party), data, call condition, input signal, event can be addressed precisely. Thus the features can be described at a more detailed level. By using behavior description, we specify features' behavior at different abstraction levels that can satisfy the needs of different study goals.

As another example, again take a look at the Call Waiting feature, we will use our CCM style description to express part of the CW behavior stated in natural language as:

> *"When a call attempts to reach a busy line (and user of this line subscribed CW), Call Waiting generates a call-waiting tone to alert the called party ..."*

As before, we assume x is a CW subscriber, x called y and established a call connection. While x is talking to y, z called x and triggered x's CW feature. The above piece of behavior can be expressed in CCM as follow:

IF

$x :: \{x \rightarrow y\} :: x$ IN ***O_Active***

AND

$x :: \{z \rightarrow x\} :: x$ IN ***DP_Term_Auth***

THEN

CW_Tone TO $CW(x, \{x \rightarrow y\}, \{z \rightarrow x\}) :: \{x \rightarrow y\} :: x :: User$

The two call connections of the Call Waiting feature and the relevant call states concerning the above behavior in the two contexts are shown in Figure 4-6.

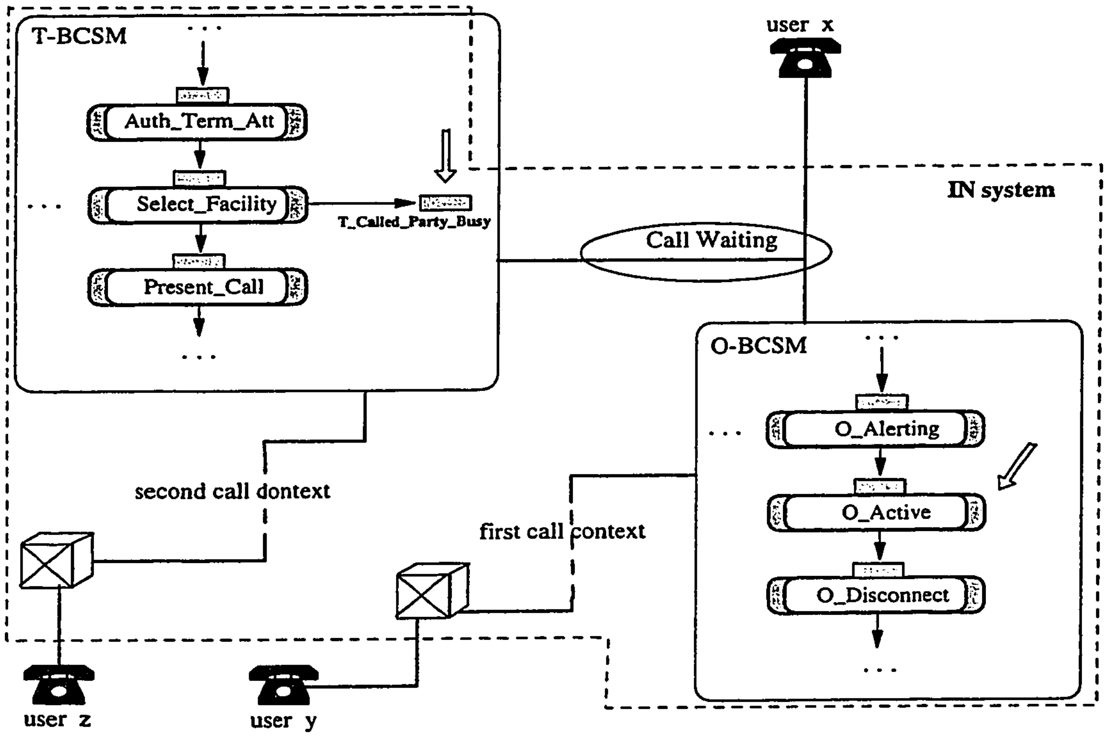


Figure 4-6: Illustration of Describing feature CW

4.2 Indicating FI using CCM

CCM is a useful model to describe features and feature instances as introduced above. Furthermore, it is also capable of indicating the existence of some FIs. This indication is derived simply from the context definitions of the features even without having to care about their functions or behaviors.

Known feature interactions are mostly described using natural language, with respect to what features are involved, what call party and subscriber take part in this interaction, and under what situation the interaction occurs, etc. For example [Cameron93] describes the FI between Call Waiting and Three-Way Calling as follow:

> *"Suppose during a phone conversation between* ***A*** *and* ***B****, an in-coming call from* ***C*** *has arrived at the switching element for* ***A****'s line and triggered the Call Waiting feature that* ***A*** *subscribes to. However, before being alerted by the call-waiting tone,* ***A*** *has flashed the hook, intending to initiate a three-way call (to include* ***D*** *into conversation). Should the flash-hook be considered as the response for Call Waiting, or an initiation signal for 3-Way Calling?"*

To have a concise idea about this interaction, we can use CCM to express the two features at the most abstract level (assume ***A***'s number 222, ***B***'s 333, ***C***'s 444, ***D***'s 555):

Call Waiting:	***CW*** *(* ***x****, {* ***x*** *->* ***y*** *}, {* ***z*** *->* ***x*** *})*
Three-Way Calling:	***3WC*** *(* ***r****, {* ***r*** *->* ***s*** *}, {* ***r*** *->* ***t*** *})*

where ***x, y, z, r, s, t*** are all users, x subscribes CW and r subscribes 3WC.

Feature interactions are actually feature instance interactions because it is the feature instances that run according to the feature definitions. We need to concretize the above two expressions. The mentioned feature interaction occurs when A subscribes both CW and 3WC. Both ***x*** and ***r*** in the two expressions will be replaced to ***A*** (222). Because the first call connection (i.e., the first call-context) is always shared by all the features, both feature instances have the same first call-context { 222 -> 333 }. Thus we have:

Call Waiting:	***CW*** *(222, { 222 -> 333 }, { 444 -> 222 })*
Three-Way Calling:	***3WC*** *(222, { 222 -> 333 }, { 222 -> 555 })*

FI can be seen between the two features because their CCMs indicate that, once triggered, *CW* would allow the establishment of a second call-context where its subscriber is the called party, whereas the *3WC* would set up a second call-context where its subscriber acts as calling party. Thus they just cannot be triggered at the same time because they will perform conflicting actions (i.e., setting up the second call-context in contradictory manners). One of them will not be able to perform its task; this is an FI.

To further explore the exact reason that causes the interaction between them, behavior descriptions can be performed to reveal more details of the two features and how they would interfere each other when both are triggered. For example, we can see that the triggering conditions that the two features use are similar:

222	*::*	*{222 -> 333}*	*::*	*222*	*::*	***O_Active***	*::*	***Hook_Flash***
subscriber		*call-context*		*call party*		*call state*		*user input*

Only that CW needs another condition:

*222 :: { 444 -> 222 } :: 222 :: **DP_Term_Auth***

However, this is a condition that may become satisfied by external factor (if there is an in-coming call). So we are sure FI exists between the two features when one subscriber has both of them.

4.3 Features' Context Assumptions (FCA)

After years of study, some researchers recently recognized that despite the existence of a group of interaction detection methodologies, we are still far from solving the feature interaction problem. In the latest international feature interaction study conference Feature Interaction Workshop '97, [Kimbler97] urged to address FI problem at the enterprise level suggesting newly developed methodologies should keep in mind a few important questions so that these research results can be really valuable. We have made our efforts trying to answer the following questions that are among those presented by [Kimbler97]:

- How to detect interaction on a large scale ?
- How to make feature interaction detection efficient ?
- What information about features is needed to detect feature interactions?

Practical FI detection and resolution methods are needed for the telecommunication industry, given the fact that even the existence of a general solution for this problem is yet to be determined and proved.

We explained how to use CCM to outline a feature and its instances, even to indicate some feature interactions. However, we noticed that the most abstract level CCM is too rough to describe a feature and its instances, whereas fully detailed behavior descriptions are also hard to achieve because every piece of data, every small action as well as every user input have to be specified. A significant computational complexity will be encountered if we try to analyze features using automated software tools at this level, especially when the behaviors of the features are not trivial. Thus, we need to use some notations that are between the most abstract CCM and detailed behavior description to help us detect FIs more efficiently.

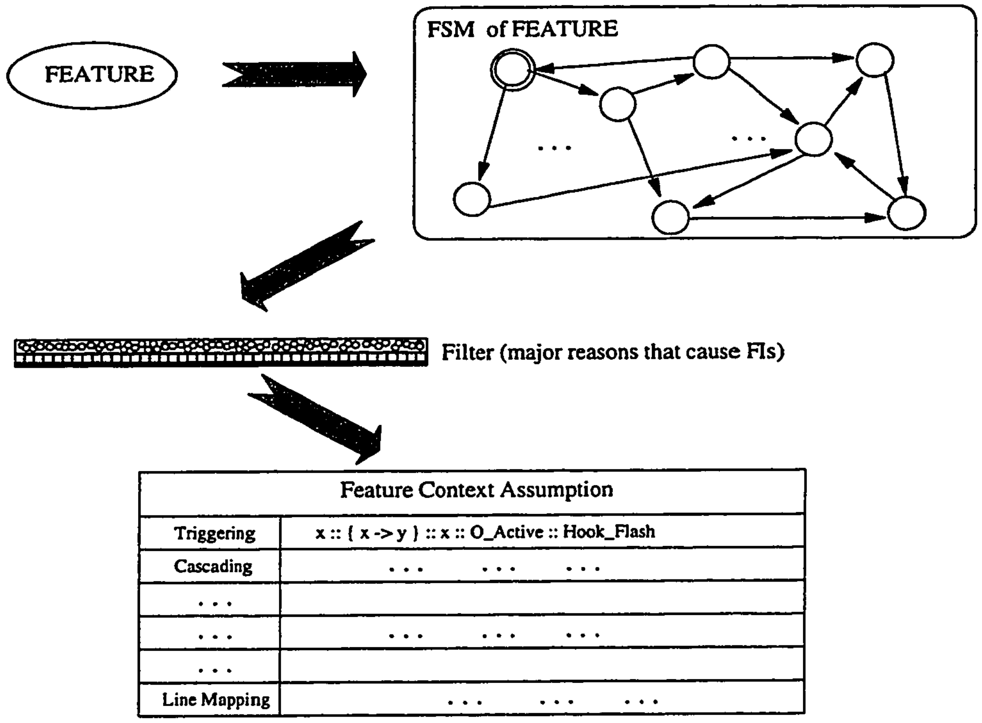
4.3.1 Principle of the FCA

FCA is introduced under the belief that not all information about a feature is needed for FI detection purpose. By focusing on information that directly related to FI, we can greatly reduce the problem size and complexity. This will allow us not only be able to detect FIs but also do it efficiently.

Due to the complexity of the networks and features, human analysis cannot achieve high efficiency and its is easy to neglect some infrequently met situations. Even the automated software tools based on formal methods still may meet the computational complexity problem when the size of the system becomes larger and larger, even if they can work thousands of times faster than human beings. In order to detect as many as possible FIs without encountering computational complexity problems, we must find out what aspects of a feature need to be studied.

After studying various telecommunication features and known feature interactions, we believe that a major attention should be paid to exploring the reasons that cause FIs. Because those reasons will suggest what aspects of the features deserve to be carefully analyzed, it can serve as a tool to reduce the amount of information about a feature that we need to analyze. Figure 4-7 shows the idea that feature-behavior description contains too much information about the feature for FI detection purpose. Some of the information is unnecessary and can only decrease our efficiency in FI detection. Possible reasons that lead to FIs should be used as a "filter" to reduce the volume of information about features. The filtered information will form the FCA of the feature.

From the viewpoint of a feature, in order to perform its task(s) correctly, it must make some important assumptions about its running environment (i.e., the networks). So we focus our study on the fundamental requirements that a feature demands from the environment in which it runs. We noticed that when all the assumptions can be satisfied, the feature can get its work done without FI. On the other hand, if some assumptions cannot be satisfied due to the existence of other feature(s), then FIs occur and this feature cannot behave correctly.



Feature Context Assumption	
Triggering	x :: { x -> y } :: x :: O_Active :: Hook_Flash
Cascading	. . . .
. . .	
. . .	. . . .
. . .	
Line Mapping	. . . .

Figure 4-7: Filtering Information that vital for FI

These important assumptions of each feature need to be carefully specified so that we can determine when they can be satisfied and when cannot. This is the idea of summarizing the Feature Context Assumptions (FCA). A feature's assumptions about the networks rest in many different aspects such as call data, signals, user input, resources, etc. Also these assumptions may be in different call connections that related to the same feature. Furthermore, combined conditions can be involved.

4.3.2 Causes of Feature Interaction

In order to study what kind of factors should be taken into account to form the FCA of a feature, we must first have a look at the possible reasons that lead to feature interactions. Through the study of known interactions, we have noticed several generic causes that can result in different kinds of feature interactions. These causes may not be the only ones, but they are surely the most frequent ones.

- *Triggering Condition Conflict:* A feature is active only under certain situations. During the basic call processing procedure, when certain conditions are satisfied, the system will start executing the feature. Such conditions are called "triggering conditions". When two features have the same triggering conditions, they are destined to interfere with each other if no priority-order has been pre-determined, since the system has no way to decide which feature should be triggered, which one should not. Figure 4-8 indicates such an FI between the features Call Waiting and Answering Call.

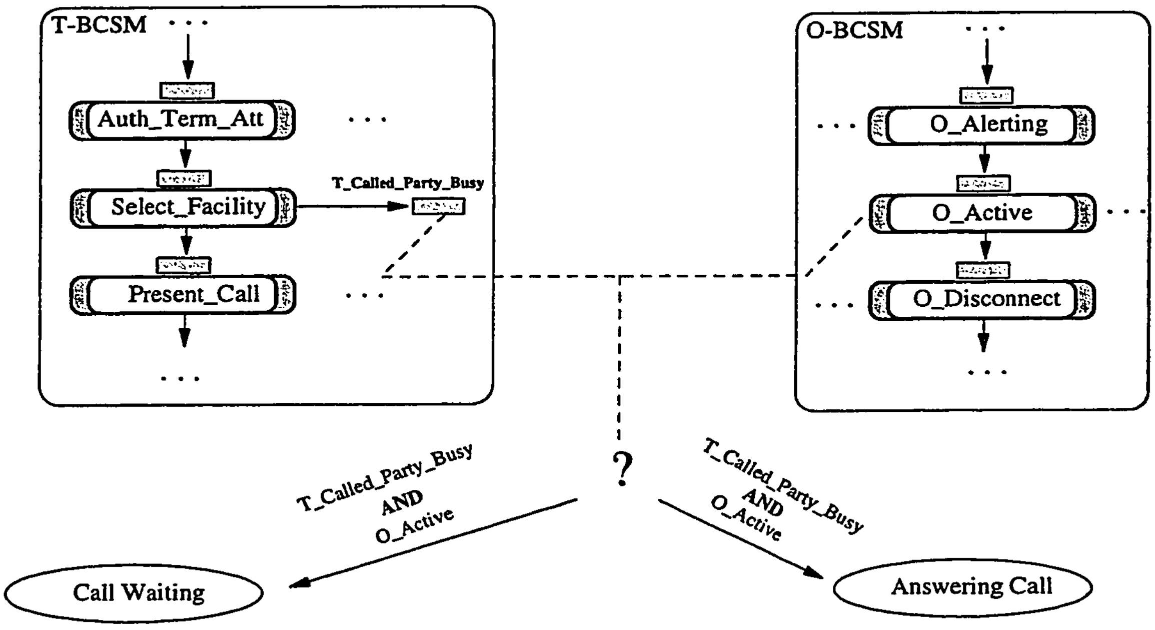
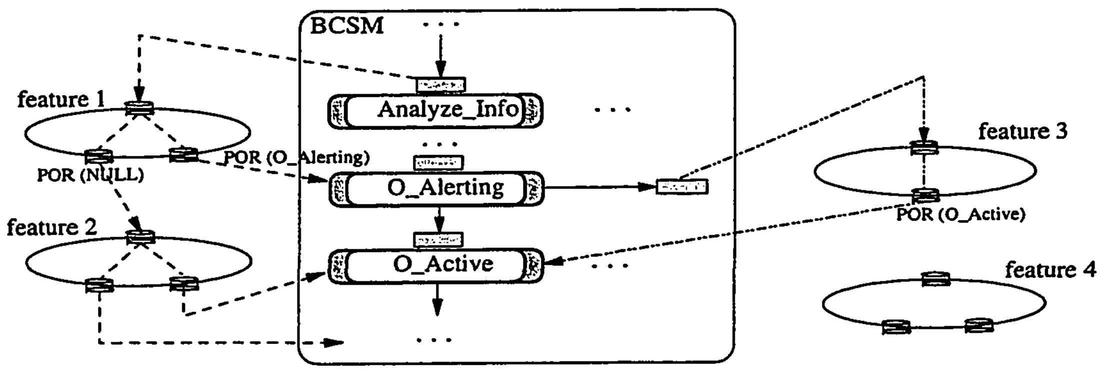


Figure 4-8: Triggering Conflict between feature CW and AC

- *Priority / Cascading Conflict:* Priority means a pre-arranged triggering order when two or more features have the same triggering conditions. If no priority sequence has been provided, or the priority sequences given by two features are not compatible, then interaction will surely occur among these features. Cascading denotes the connection of entry and exit ports of more than one feature that are triggered in the same detection point (DP) and same triggering conditions in the BCSM. If two features have conflicting cascading conditions, interaction between them is inevitable. An example of cascading conflict is shown in Figure 4-9.



Case 1 :
feature 2 cascades after feature 1 because feature 1has a "null" POR which means continue with where the feature was triggered.

Case 2 :
feature 4 can't cascade after feature 3 since feature 3 has only one explicitly specified POR that directly jumps to a PIC.

Figure 4-9: Priority/Cascading Conflicts Illustration

- ***Data Manipulation Conflict:*** During the execution of a feature, it needs to perform certain operations on certain system data, call data, etc. These are called "data manipulation". If the features are not allowed to manipulate those data, they will not be able to work correctly. If one feature is prevented from accessing the data it needs to, due to the existence of another feature, then FI between them is obvious. Suppose two features ***Fa*** and ***Fb*** will work together, feature ***Fa*** must read system data ***D*** to work well, feature ***Fb*** must hide ***D*** to fulfill ***Fb***'s task, then we conclude that interaction definitely exists between them since their data manipulation requirements collide.

- ***Call Resources Conflict:*** To get its work done, a feature may request to utilize certain number and certain type of system resources. If more than one feature requires the same type of resource and there are fewer resources available in the system than requested by the features, then interaction is again sure to occur among these features.

- ***Call Operations Conflict:*** In order to perform its own task(s), a feature needs to perform certain call operations successfully such as "put a party on hold", "add a joint leg", "initiate a new call", etc. If two features have requested some contradictory operations to be done, feature interaction cannot be avoided between them.

- ***Signal Claiming Conflict:*** When a feature is active, it may claim certain signals/events at certain call-states and relies on them to convey special meanings. When such a signal/event is met, this feature will consume it exclusively. If two features have claimed the same signal/event in the same call-state of the same call-context, the system cannot decide which feature should be given this signal/event. So the two features definitely will interfere with each other when running together.

- ***Basic Line Assumption:*** This refers to the fundamental assumption about the mapping between users and physical telephone lines. The possible mappings in the system are: ***1-Line-1-User***, ***1-Line-n-User*** , ***n-Line-1-User*** and ***n-Line-n-User***. If one feature presumes to work with a 1-Line-1-User line while another needs a 1-Line-n-User line, then they just cannot work together on the same line without interaction, unless the system provides some mechanisms to identify different users registered on the same line and different lines registered for the same user.

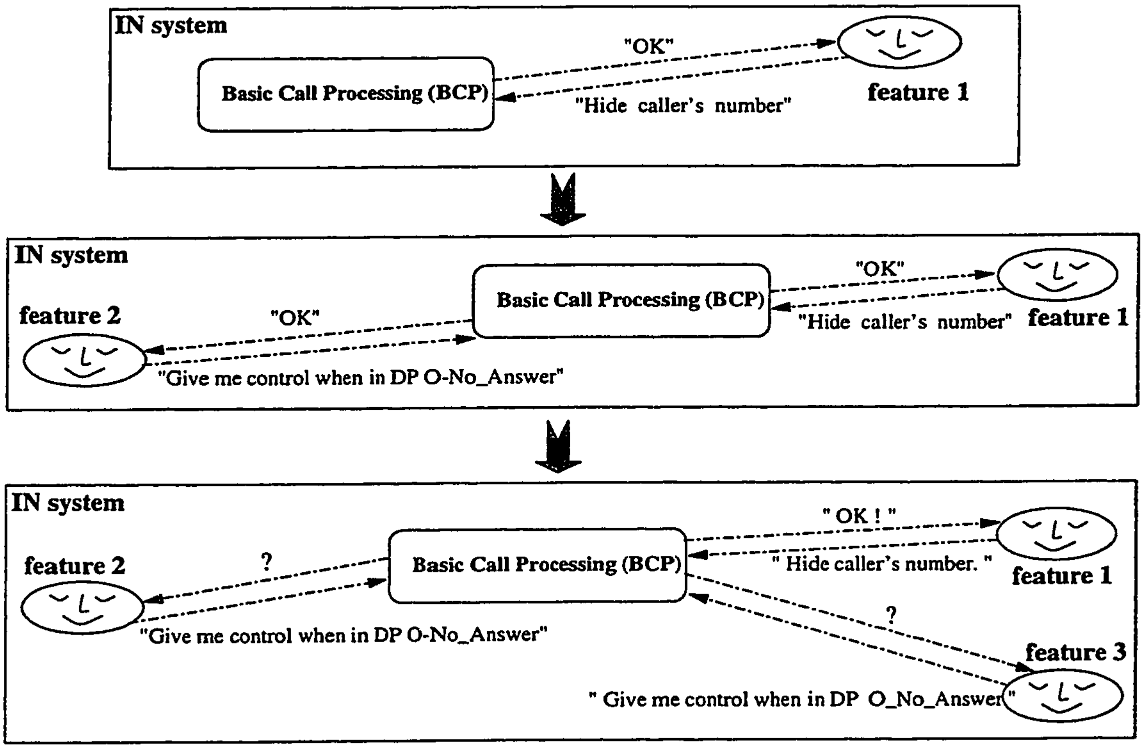

IN system
"OK"
Basic Call Processing (BCP)
"Hide caller's number"
feature 1
IN system
"OK"
Basic Call Processing (BCP)
feature 2
"OK"
"Hide caller's number"
feature 1
"Give me control when in DP O-No_Answer"
IN system
"OK ! "
feature 2
?
Basic Call Processing (BCP)
" Hide caller's number. "
feature 1
?
"Give me control when in DP O-No_Answer"
feature 3
" Give me control when in DP O_No_Answer"

Figure 4-10: FI resulted from unsatisfied assumptions

All these types of assumptions are determined when the features are designed and implemented. Due to the fact that multiple feature vendors exist, the developer of one feature may not even know the existence of another feature manufactured by another company. Features are developed independently and are all based on the basic call processing functions specified by BCSM. Hence, it is very possible that different features have conflicting environmental assumptions. That is why FIs exist.

FIs are mostly resulted from unsatisfied feature assumptions. When a feature works alone in the network, its assumption can always be satisfied because the feature designers had carefully chosen the assumptions based on the behavior of basic call processing. However, when more than one feature are running concurrently, the behavior of a feature ***Fa*** may have updated the behavior of the basic call processing, thus another feature ***Fb***'s assumptions that were supposed can be satisfied based on normal basic call processing behavior now becomes unsatisfiable. Consequently, the latter ***Fb*** cannot work correctly because of the existence of feature ***Fa***. Figure 4-10 shows that FIs occur when the system cannot satisfy all the requirements of features.

4.3.3 Proposed FCA Contents

After studying the above seven feature interaction causes, we conclude that information about a feature's assumptions in these aspects is quite crucial if we want a feature to be able to run correctly without interaction with other features. FCA should include information of all those aspects. The above frequent causes of feature interaction indicate that features are very likely to affect one another due to their demands from the networks in these aspects. Given this result, we believe the following contents need to be included in a feature's FCA:

Symbol-and-Context:	Specify the Call-Context(s) of this feature.
Triggering:	Specify the triggering conditions of this feature.
Cascading:	Give out the priority and cascading assumptions.
Data-Manipulation:	List all the vital data manipulation operations needed.
Resources:	List all kinds of system resources necessary.
Call-Operations:	List call operations need to be performed.

Signal-Claiming:	Specify the signals that convey special information throughout the execution of this feature, including the call states in which the signal should be recognized.
Line-Assumption:	Specify the fundamental user-line mapping assumption.

Table 4-1: Proposed Contents of Feature Context Assumption

By using FCA to represent various features, the complexity of analyzing them is significantly simplified. Specifying what needs to be guaranteed and finding out if it is satisfied is much easier than tracing the execution of features and looking for evidences of feature interactions.

4.4 Discussion of CCM-FCA

After having introduced the Call Context Model (CCM) and Feature Context Assumption (FCA), we give the advantages and shortcomings.

Advantages

A few important points for feature-property description using our CCM-FCA model:

- **Clearness:** Different features' and feature instances' behaviors are much more distinguishable because of the use of BCSM call states to show call situations in each call-context that ultimately have impact on how these features continue processing these call connections.
- **Preciseness:** Specifying a feature's behavior using specific elements such as call data, call conditions, signals, and call actions greatly improves the possibility of mutual technical understanding of a feature. Ambiguity can be reduced to a much lower level.
- **Unique Addressing:** All the call data, call conditions, signals call states are referred to uniquely with regard to the call connection(s) and BCSM concepts.
- **Versatility:** Such descriptions not only can be used to analyze different features but also capable of studying different instances of the same feature.

Disadvantage

- The person who prepares the CCM-FCA for a feature needs to have thorough knowledge about the feature. Usually only the feature designer can do that.
- There may exist features that are hard to express using CCM-FCA.

According to the FI detection framework that will be introduced in Chapter 5, the feature designers will prepare the CCM-FCAs. The FI detectors will only need to be familiar with CCM-FCA mechanism and possible situations of utilizing a specific feature.

4.5 Chapter Summary

The main content of this chapter is the introduction of two key concepts that are specifically used for feature description and feature interaction detection.

Call Context Model (CCM) is based on the idea that a feature's behavior consists of the feature's partial behavior in each of the call contexts in which it is involved. CCM outlines features' running context and behavior and can characterize a feature at different abstraction levels using specific elements (states, events, data, etc.).

The Feature Context Assumption (FCA) reflects features' important environment assumptions that must be satisfied for them to run correctly. It is used to reduce the complexity of describing features for feature interaction detection purpose. After the analysis of known feature interactions, we summarized a few kinds of causes that can lead to such interactions. The content of features' FCA is based on these causes.

Chapter 5 Feature Interaction Detection

Telecommunication features are packages of enhanced call functions that provide more powerful and customized communication capabilities. However unlike the POTS functions, they serve the users on a subscribe-and-use or pay-per-use basis. Features reside in the network quietly without even being noticed when they are not active. But once certain triggering conditions are satisfied, an instance of the feature is created to provide its user with the designated "feature function". Feature Interaction (FI) is defined as all kinds of interference among multiple concurrent feature instances that prevent at least one of them from performing their tasks correctly.

FI problem has been hindering the feature development since early 90's. Due to inherent complexity of this problem, no general solution has been found yet. However strong user demands for more features in telecommunication networks require practical FI detection and resolution methodologies. This chapter presents an approach for FI detection that aims to detect FI effectively and on large scale. The basis of our detection scheme is the Call-Context Model (CCM) that summarizes the major characteristics of features, and the Feature Context Assumptions (FCA) that outlines the important environmental requirements of the features.

5.1 FI Detection Framework

Features are essentially complex computer programs that control the operation of network devices. Developing features also conforms to the rules of normal software manufacturing. Software engineering theory has suggested that eliminating any deficiency in earlier stage of development is much cheaper than at a later stage. As an effort to detect FI as early as possible in the feature development life cycle and make it cost-effective, we present our FI detection framework concerning what constitutes a feature's life cycle, when to detect FI in the life cycle and who will actually perform the task of detecting FI.

5.1.1 Life Cycle of Features

The major stages of feature development are depicted in Figure 5-1. From starting to end, they are: *User Demand Analysis, Feature Contriving, Feature Specification, Feature Design, Feature Implementation, Feature Testing, Feature Deployment, Operation and Maintenance.*

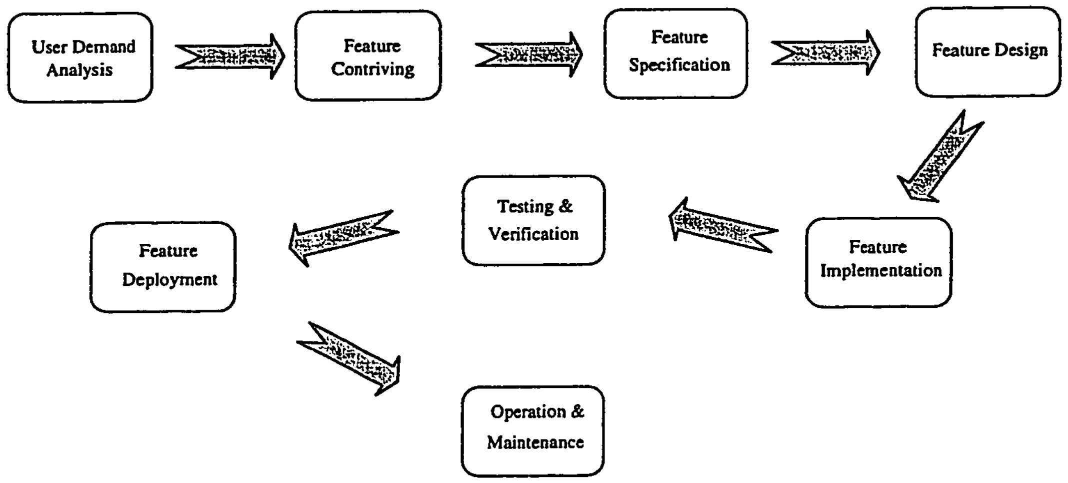


Figure 5-1: Life cycle of telecommunication features

Although most stages are identical to that of normal software development, the Feature Deployment (FD) is specifically for feature creation. FD refers to the inclusion of features into appropriate equipment of the telecommunication networks and making the feature network-wide (or even broader, any reachable inter-network domain) available to serve its subscribers.

5.1.2 When to Detect FI ?

Facing this problem, different people have considerably different opinions that in turn affect their FI study directions. Currently there are three major groups of approaches in feature interaction research area (see [Cameron 93]):

(1) *Enhancing Network Infrastructure for Feature Deployment*

(2) *Eliminating Feature Interactions from Feature Design*

(3) *Run-time Feature Interaction Resolution*

The first group believes that FI problem is a system problem rather than a feature development problem. Thus by dealing with system design issues that are related to FIs when features are being deployed would be able to address this problem to a very large extent. For example, a new naming scheme that can uniquely identify among directory numbers, line and users could dissolve FI stemming from these issues; a richer set of functional signals could help resolving some ambiguities caused by limited signaling set that easily results in FI when two features use the same signal to convey different meanings; a good distributed system platform could manage problems due to non-atomic operation and the distributed nature of telecommunication networks.

The second group of people consider FI problem as feature designing and implementation deficiencies that can be avoided if enough attention is paid when the features are being built. They typically focus on the detection and resolution of FI during the design phase of the feature development life cycle. Please note that "design phase" refers not only to the feature design stage but also to the feature specification stage and the feature implementation stage. For example, FIs between two features may be caused by conflict issues in their specifications, which are avoidable if we appropriately adjust the two specifications. In addition, decisions made at a feature's design stage about certain signal utilization can be re-evaluated so that different features will not choose same input signals that cause conflict.

The last group of approaches are advocated by people who do not believe that all FIs can be discovered and resolved in their early stages. Because of diverse user preferences, no single policy that governs the availability of call data could be mutually satisfactory, nor could a set of rigid precedence relations exist for resolving all kinds of conflicting call controls. Researchers who work towards this direction concentrate on adopting reasonable actions to dissolve the encountered FI by trying to compromise less important behavior of the features and allow them to fulfill their major tasks. Figure 5-2 shows which stage in feature life cycle that the three groups work on.

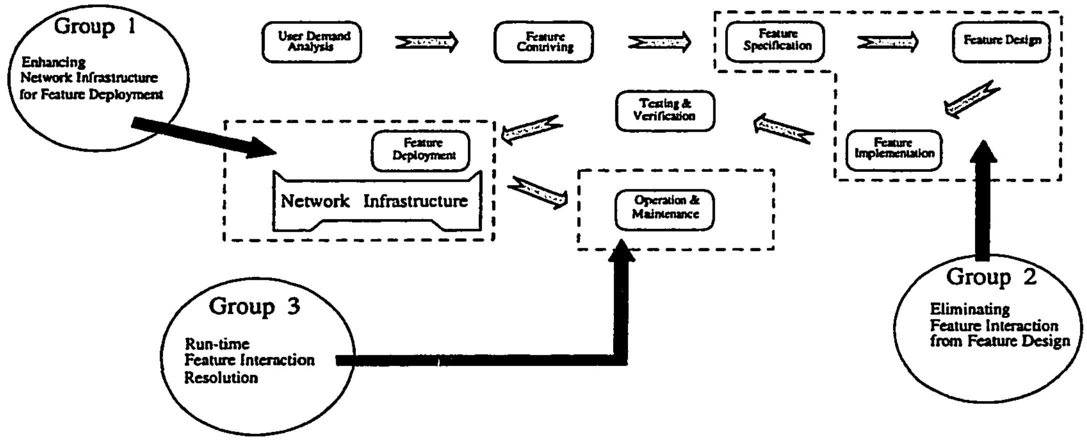


Figure 5-2: FI study groups working on different stages

Our scheme of FI detection belongs to the second group that work on improving the specification, design and implementation of features so that as much as possible FIs can be detected before the features are actually deployed into the networks. However inside this group, there are three different stages in which researchers can work to detect FIs:

- Discovering FIs based on feature specifications that describe features' functions.
- Screening out FIs by analyzing feature designs that realize feature functions.
- Checking for FIs when features are being implemented into computer code.

After carefully analyzing the advantages and drawbacks of focusing one each of the above three stages, we came to a conclusion that simply concentrating on any single stage will not yield satisfactory FI detection result. For example, although some people are working on specification stage and did make some achievements (e.g. [Ohta 94], [StepL 95], [JalelL 96] and [Frapp 97]), many kinds of FI cannot be discovered by examining feature specifications simply because they only occur at later stages due to specific design or programming decisions made at these stages.

In order to overcome the limitations associated with working on a single development stage, we have adopted an approach that will collect information about a feature throughout its specification, design and implementation stages so that decisions made in all these stages that may contribute to FIs are taken into consideration. This approach allows us to span three stages when summarizing and to specify (to a certain abstract level) each feature's necessary information so that the FIs can be detected no matter at which stage they are brought in.

Moreover, detecting and solving FIs in the design phase will avoid the expensive costs of fixing them after the features are deployed into the networks. The feature vendor will also enjoy better reputation of their products among their customers.

5.1.3 Who should Detect FI ?

Believe it or not, few papers in FI research explicitly state who should be responsible for FI detection despite the fact that many people are working on this topic. Researchers are only responsible for inventing FI detection methodologies. They are not the ultimate FI detectors who carry out the on-going FI detection tasks. From various types of approaches that aim to address FI problem, we can have some hints about what kinds of people are supposed to detect possible FI using their approaches by the approach inventors.

The approaches of "Enhancing Network Infrastructure for Feature Deployment" pay more attention to the network structure when deploying the features. Inventors of such methods seem to put the responsibility of FI detection on the feature deployment personnel and the network infrastructure designers. Researchers who insist on "Eliminating Feature Interactions from Feature Design" are apparently counting on the feature designers to detect possible FIs during the feature's design phase. Feature designers are also responsible for resolving these detected FIs. The "Run-time Feature Interaction Resolution" is advocated by people who wish the network operators to discover and resolve FIs by developing supervisory FI control software into the networks.

However, we did not adopt the idea of using the above kinds of people as ultimate FI detectors. The reason is that none of the three approaches designate appropriate FI detectors, which may affect the ability of detecting FIs efficiently and on a large scale.

- Putting the FI detection task on feature deployment persons will unreasonably increase their burdens and is not very feasible because their knowledge is limited about how these features' behaviors could lead to FIs.
- Network infrastructure designer may be able to improve existing network design to resolve already-known FIs. But given the fact that no functional limitations exist for

features, no network infrastructure designer can have the foresight to arrive at a system that can accommodate both the current and future features.

- Assuming the feature designers to detect FIs and resolve them by improving their design seems plausible but actually it is unrealistic because of the existence of multiple feature-vendors. A feature designer in one company may not even aware of the features that are being built in another company, let alone to consider the FIs between his feature and theirs.

- "Run-time FI Resolution" counts on network operator or feature users to discover FIs during their use of features. This is the last resort we want to take and it is also not satisfactory. A user can discover that certain feature is not working well but usually unable to identify how this is caused (e.g., by FIs or other network problems). Even the experienced network operators are usually unable to trace the problems and determine whether it is caused by FIs because they are not FI experts. Even we put an FI expert there will be still not enough because it is very difficult to have in-depth knowledge about the behaviors of every feature deployed in the network. Even worse, new features are pouring in probably every day!

We suggest that FI detection should be performed by a group of people that are independent of any feature-manufacturing company. This group is specifically responsible for detecting FIs that may occur among various features built by different companies. Feature developing companies submit partial information (enough for FI detection purpose) about their features to this FI detection group. The feedback from this group will be sent back to feature developing companies to help the feature-developers to improve their specification, designs or implementation. The same kind of information about the revised features can be re-submitted to this group for further FI detection until no FI can be found. The framework of our FI detection framework is shown in Figure 5-3.

Under this framework, the feature developers only need to submit the absolutely necessary information about their features in order to allow this group to detect possible FIs. One advantage is the reduced complexity and size of representing the features that allow members of this FI detection group to have sufficiently knowledge about most existing features and features under development, which enables them to detect FIs effectively and efficiently. Another advantage is that the patent issue can be addressed, which is always a major concern of the feature developing companies.

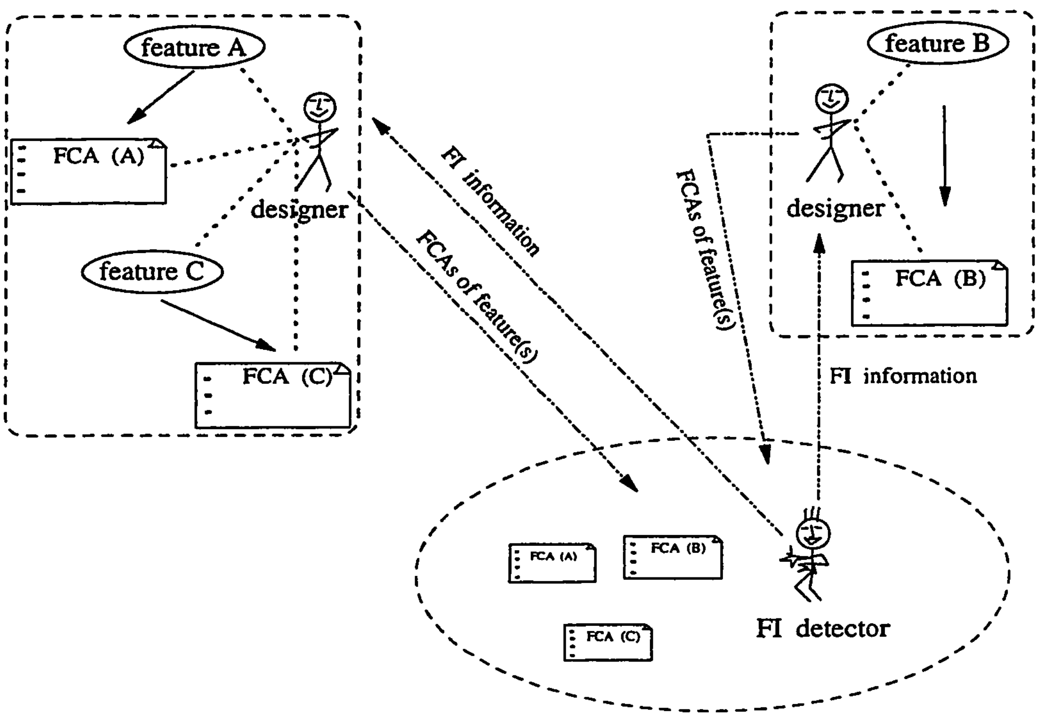


Figure 5-3: Framework of our FI detection approach

In addition, the ability to determine if a feature has possible interactions with other existing features through independent FI detection group helps feature developers to improve their result to make sure the features are FI-free before the feature design/ implementation is finalized and delivered to next development stage.

5.2 FI Detection Approach

Under our FI detection framework, we have proposed a new FI detection approach that is based on the CCM and FCA of the features. The principle and specific steps of our FI detection approach are explained in this section.

5.2.1 Basis of our approach

What kind of information about the features is need in order to detect FIs is an important issue to decide. If the information is too rough, actually existing FIs may get through the detection process without being discovered; if the information is too detailed, patent-related and proprietary issues may rise because feature-developers are concerned about the possible leakage of their technical secrets. Moreover, too detailed information easily leads to unmanageable computational complexity that would dramatically reduce the efficiency of FI detection.

Unlike some researchers who devoted their FI study on specifying and analyzing the features' detailed behaviors, we do not deal with the minute behavior of the features. We are focusing on analyzing environmental factors that are crucial for a feature. These factors are considered to be crucial when a feature cannot perform its designated call functions if these factors fail to satisfy the features' needs.

The basis of our FI detection is the Call-Context Model (CCM) and Feature Context Assumptions (FCA) introduced in Chapter 4 of this thesis. CCM allows us to describe the features with regard to specific call context(s). FCA outlines the important requirements of the network environment in which the features are running.

- The *Call-Context Model* (CCM) describes the related call-connections of a feature in a clear manner. In this model, a feature is considered to have one or more call contexts, where each call context is essentially a communication connection between two parties. A feature's behavior depends on the states and events in the call context(s) that the feature is involved in. The feature also exerts effects on the call context(s) by altering the basic call processing functions.

- The *Feature Context Assumption* (FCA) helps to describe the important needs of a feature. FCA represents those assumptions about the system that must be satisfied in order for the feature to work correctly. In other words, FIs will be encountered if any of those assumptions becomes invalid due to some reason (e.g., because of the existence of another feature).

As FI detection specialists, the persons need not know much about the detailed behavior of each feature. Instead, they are familiar with the general working mechanisms of features and how features utilize the network resources including switching, signaling and data. Also, they have good knowledge about various kinds of situations, causes and logical problems that may result in FIs.

It is CCM and FCA that allow FI detectors to have good understanding to all the features that have been built by different companies without having to know much about these features' detailed behaviors. Compared with their detailed behaviors, features' CCM and FCA are much less complex and usually do not cause computational problem when being analyzed. FI detectors can achieve good efficiency even when trying to detect FIs among a relatively large number of features.

5.2.2 Technical Principle

After analysis, we believe that FIs only occur at a specific time when multiple features (or multiple instances of the same feature) have been triggered and the networks cannot satisfy all the requirements demanded by these features. Thus we need to specify and study such feature requirements in order to understand and detect the FIs among them. The CCM and FCA of a feature exactly reflect the important environmental requirements that must be satisfied in order for the features to work well.

The major technical principle of our FI detection approach is to put together all the environmental requirements of the features under study, and then analyze under what situation they can not be fully satisfied. Such a situation is considered to be equivalent to an occurrence of an FI. In practice, once we have got the CCM-FCAs of two or more features, we perform a combining analysis of them by "joining" them at one or more common call contexts (CCC, described in Chapter 4). If the networks can satisfy all the assumptions inside those CCM-FCAs, no FI exist among these features. Otherwise the evidence of the existence of FIs is found. Some measures need to be taken, either to update the features or to enhance the network system, to solve the found interaction(s). Figure 5-4 illustrates the mechanism of "joining" two features at a common call context.

The designer of a feature is the one who knows the feature best. We suggest he/she gives out a list of essential assumptions that must be satisfied for the feature to work properly. According to these lists submitted from different feature developers, an FI detector can discover inherent FIs among these features without actually running them. This makes it possible to discover FIs even before a feature's design is finalized.

A FI detector does not need to be very familiar with every feature, he/she only needs to be proficient with the CCM-FCAs. The CCM-FCA can be made to have a unique format and independent of the feature vendors and underlying hardware. This will enable effective and efficient FI detection.

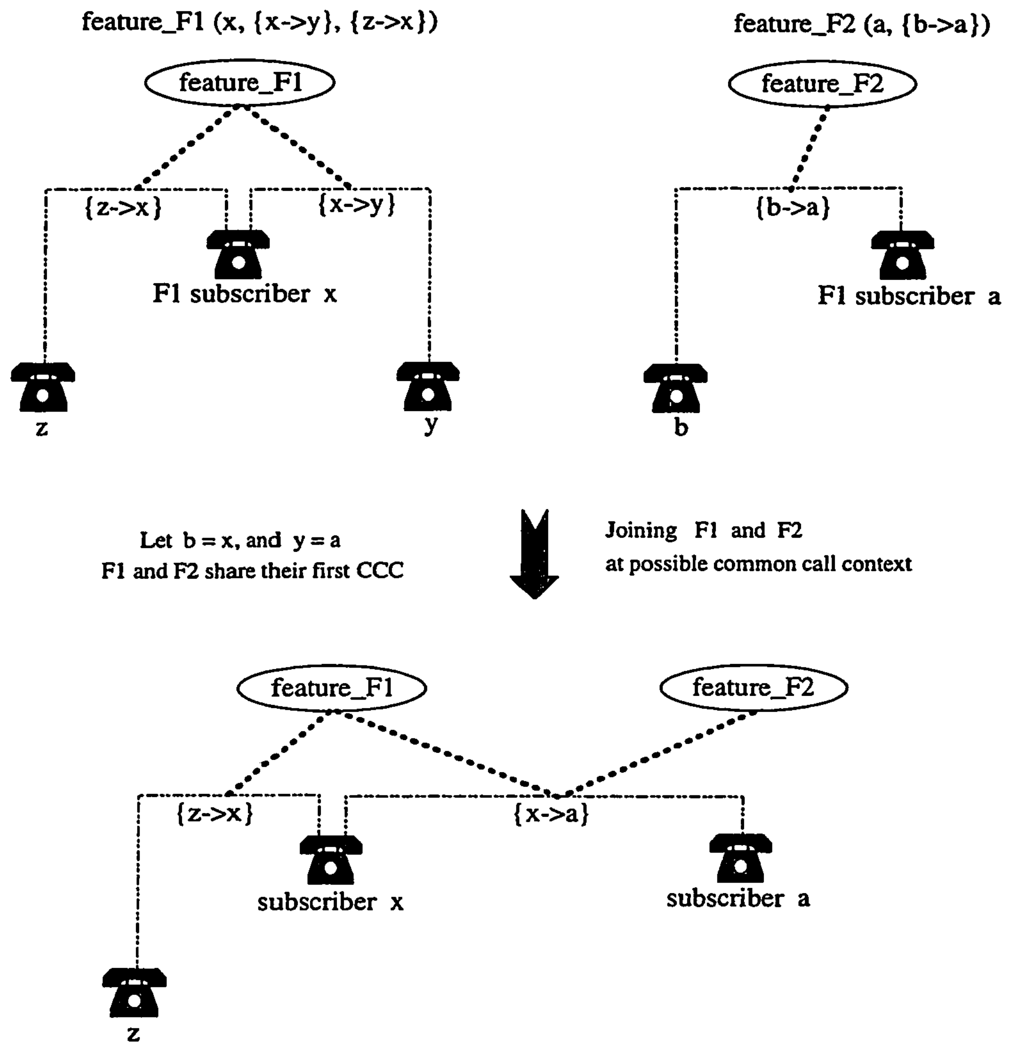


Figure 5-4: Joining two features at a Common Call Context

5.2.3 Detection Steps

Our FI detecting approach is based on the conflict checking in the CCM-FCAs of the features under study. These conflicts are indications of FIs. Once an assumption-conflict is found, an indication that the network cannot satisfy all the features' requirements is found. This fact reveals that at least one of them will not be able to fulfill their tasks. Thus there is FI.

If we have found conflicts in the features' CCM-FCAs, equivalently we have found FIs among these features.

Given the CCM and FCA descriptions in Chapter 4, the following steps are suggested when using our FI detection approach:

Step 1: Describing features in CCM. Use CCM to express the features, including analysis of all the call-context(s) involved, and the roles of subscriber and other call parties. However, behavior descriptions of features are NOT needed.

Step 2: Listing out the FCAs of the features and put them together with the CCM. The CCM-FCAs should represent each feature's important assumptions about the networks. These assumptions are based on the call data, network data, BCSM states, signals, events, and so on, with regard to the each call context of the features.

Step 3: Joining the features on common call-context(s). That is: analyze various feature-using situations in which different feature subscribers play different roles, and list those that allow features to have common call context(s). This is because two features must have at least one common call-context in order to produce any feature interaction. There are two kinds of situations where two features may have common call context and thus may interfere each other.

> **I.** One user subscribes to both features. So the two features will have the same first call context with this subscriber, if the two features' first call contexts are compatible (see below). They also may share other context(s) established later.
>
> **II.** The two features are subscribed by different users. But they have at least one common call-context(s) due to the establishment of a call in which one feature subscriber is the caller party and the other is the called party.
>
> *Note: Two call contexts that belong to two features are* ***compatible*** *if their subscribers' roles (caller or called party) in these two call contexts do not conflict.*

Step 4: Analyzing feature-using cases. Pick up one feature-using case and replace formal parameters in CCM-FCAs with actual values so that this case is realized. These parameters are chosen to reflect this specific feature-using case. After being given actual values, a CCM-FCA represents the description of an instance of a certain feature.

Step 5: Searching for conflicts in features CCM-FCAs. This step is to scan for conflicts that indicate the existence of feature interaction. A conflict is found when any of the following conditions is met:

> ***"Triggering" conflict:*** The triggering conditions are identical for more than one features AND no priority sequences are specified for them in the "Cascading" fields of these features' CCM-FCAs.
>
> ***"Cascading" conflict:*** The priority sequences provided in the CCM-FCAs of the features are not compatible. For example, feature F1 requests to be the first triggered while feature F2 asks to be triggered before F1.
>
> ***"Data Manipulation" conflict:*** Two or more features requested incompatible data accessing operations. For example, feature F1 asks to "write" the call data "caller's number" while feature F2 demand to protect "caller's number" from being changed. Then a conflict is found here.
>
> ***"Resources Requested" conflict:*** System cannot satisfy both features' resource demands at the same time due to insufficient system resources.
>
> ***"Call Operations" conflict:*** Two features requested contradictory call operations to be performed. Thus at least one operation will fail, which prevents at least one feature from fulfilling the tasks.
>
> ***"Signal Claiming" conflict:*** When more than one feature claims the same call input signal in the same call state of BCSM of a call party in the same call context. Only one feature will be able to consume this signal while others cannot perform their task because of this.
>
> ***"Basic-Assumption" field conflict:*** A conflict is found when two or more features specified different User-Line mapping assumption.

Step 6: If possible feature-using cases are NOT exhausted, continue with **Step 4.** Otherwise, output FI detection result with respect to feature-using case and conflict type.

From the above description of our FI detection method, we can see that it is not only able to detect FIs but also can indicate the possible reason(s) of the detected FIs. How exactly each step is performed in our FI detection approach is further illustrated in section 5.3 with application examples.

5.3 Application Examples

In this section, four examples are given to illustrate how our FI detection method successfully detects both previously known FIs as well as unknown ones. The first two are known FIs while the last two were detected when experimenting our method and have not yet been found in published FI technical papers.

5.3.1 Detecting FI between AC and CW

We first introduce the two features, then describe the FI between them and how our method was applied to detect it.

Feature: Answering Call (AC)

When a call attempts to reach an AC subscriber's line and this line is busy, the feature Answering Call will connect the caller to an automatic answering device, play a piece of recording information and ask the caller to leave a message.

Figure 5-5 depicts how AC serves its subscriber. Please recall that the Basic Call Processing (BCP) of a call is performed by running a BCP process, which is expressed by the BCSM (Basic Call State Model), which consists of PICs (Point In Call), DPs (Detection Point) and transitions.

In this figure, user *X* is an AC subscriber and is talking with user *Y*. Technically speaking, its BCSM (O-BCSM) is in the PIC of "**O_Active**". This means the call conversation between user *X* and user *Y* was initiated by user *X*, and currently in the "talk" state. When another user *Z* calls *X* at this time, a second BCMS (T-BCSM, because *X* is the called party) is launched to process the in-coming call. From the viewpoint of user *Z*, a busy tone will indicate *X*'s line is busy. However the feature AC that *X* subscribed jumps out at the moment when the second BCSM is in the PIC "**DP__T_Called_Party_Busy**". AC connects Z to an answering device as if *X*'s line were not busy but nobody is available to take the phone.

Feature: Call Waiting (CW)

As introduced early in Chapter 1, if you subscribe Call Waiting, it will send you a "call waiting tone" if a third party calls when you are talking with someone on the phone. If you wish, CW allows you to accept the new call and keep both calls connected to you simultaneously. You can talk to one of them at a time and put the other one on hold, and you can switch back and forth between the two calls as you like until one of them is finished. The functional illustration of CW is shown in Figure 1-1.

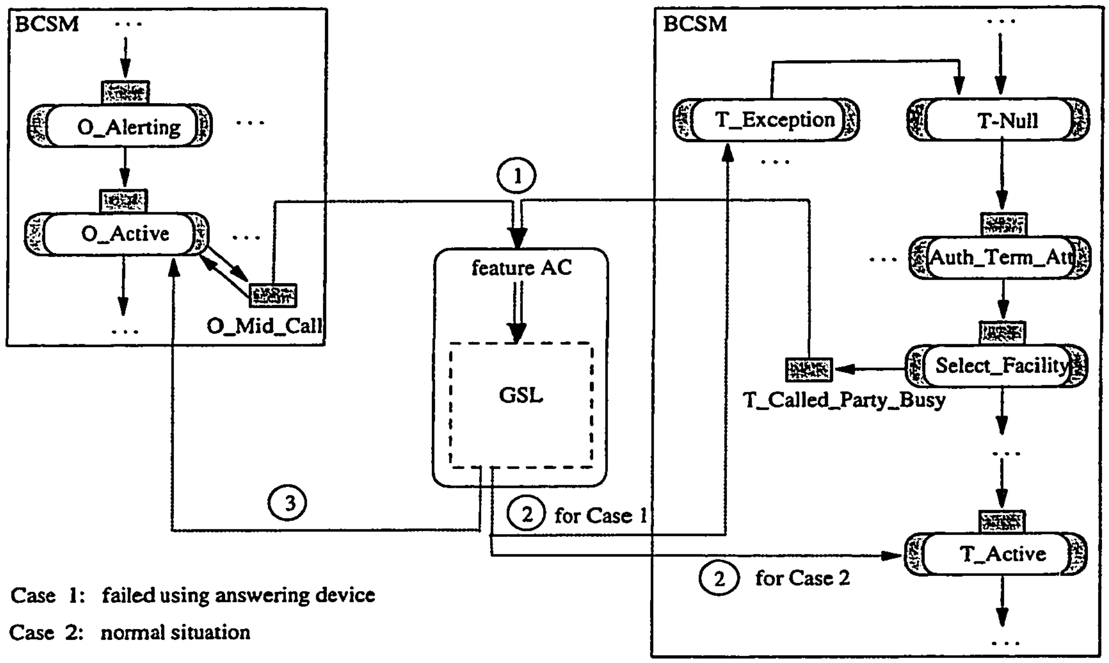


Figure 5-5: Invocation of feature Answering Call

FI between them and Detection

Call Waiting (CW) & Answering Call (AC) [Cameron 93]

> *"When a call attempts to reach a busy line,* CW *generates a call-waiting tone to alert a called party, whereas* AC *connects the calling party to an answering service. Suppose that **a** is a subscriber of both features. If **a** is already on the line when the second call comes in, should **a** receive a call-waiting tone or should the second call be directed to the answering service? "*

We apply our method by following the steps specified in section 5.2. However for simplicity of illustration, not every detail is covered. But this should not affect the understanding of how the specific FI can be detected using our approach.

Step 1: Representing both features with Call-Context Model. There are two call situations in which the feature Answering Call may be used:

$$AC(x, \{x \text{->} y\}, \{z \text{->} x\}) \text{ and } AC(x, \{y \text{->} x\}, \{z \text{->} x\})$$

And also two situations with Call Waiting:

$$CW(a, \{a \text{->} b\}, \{c \text{->} a\}) \text{ and } CW(a, \{b \text{->} a\}, \{c \text{->} a\})$$

We choose $AC(x, \{x \text{->} y\}, \{z \text{->} x\})$ and $CW(a, \{a \text{->} b\}, \{c \text{->} a\})$ in the following steps to demonstrate our approach. The other situations can be studied in a similar way.

Step 2: Listing out the two features' CCM-FCAs. Contents of the fields are explained below. (Note: *1st_CC* denotes the first call-context, *2nd_CC* the second, and so on.)

Feature Name	Answering Call	Call Waiting
Description	(omitted)	(omitted)
CCM	*AC* (*x*, {*x*->*y*}, {*z*->*x*})	*CW* (*a*, {*a*->*b* }, {*c*->*a*})
Triggering	*1st_CC :: x :: O_Active* AND *2nd_CC::x ::DP__T_Called_Party_Busy*	*1st_CC :: a :: O_Active* AND *2nd_CC::a::DP__T_Called_Party_Busy*
Cascading		
Data Manipulation		
Resource Requested	VoiceAnnouncer 1 VoiceRecorder 1	ToneGenerator 1 Timer 1 2-leg call switch 1
Operation Requested		*1st_CC :: a ::* OnHold(*b*) AND *2nd_CC :: a ::* OnHold(*c*)
Signal Claiming		*1st_CC::a::O_Active::Hook_Flash* AND *2nd_CC::a::T_Active::Hook_Flash*
Line Assumption	1-Line-1-User	1-Line-1-User

"Triggering" field

AC is supposed to work for its subscriber when an in-coming call has passed preliminary processing and discovered that the subscriber is busy when trying to present the call. The triggering criteria specified here "*1st_CC :: x :: O_Active* AND *2nd_CC:: x ::DP__T_Called_Party_Busy*" means the subscriber is busy in the "talk" state of first call, and the second caller is going to be sent a busy tone. CW is invoked at essentially the same kind of call situation as AC.

"Cascading" field

No priority sequence or cascading specification was given when FI between these two features are described and classified in [Cameron 93].

"Data Manipulation" field

Both the AC and the CW do not modify the call data of existing call context.

"Resource Requested" field

AC seems to require two resource devices to announce and record voice messages. CW needs a call-waiting tone generator to send its subscriber this special tone. And a timer is needed to determine if the subscriber wants to take the second call (if the subscriber did not respond to the call-waiting tone for a few seconds, CW will do nothing further and let the second caller to drop the call himself). The 2-leg call switch is used when the subscriber wants both calls to be connected and switches between them to talk to both partners.

"Operation Requested" field

CW needs to put the partner of first call on hold to take the second call, and also to put the caller of second call on hold to re-talk to the first call partner. Thus the operation requirement "*1st_CC :: a ::* OnHold(*b*) **AND** *2nd_CC :: a ::* OnHold(*c*)" is specified here.

"Signal Claiming" field

AC claims no user input signal. CW needs to claim the "Hook_Flash" signal in the "talk" state of both call contexts in order to convey the user intentions of accepting the second call and switching between the two calls. That is why the phrase "*1st_CC::a::O_Active::Hook_Flash* **AND** *2nd_CC::a::T_Active::Hook_Flash*" is here.

"Line Assumption" field

Both features require that the line that connects its subscriber is the only line that connects him/her and only connects to him/her. In other words, the mapping between the physical line and directory number of the subscriber is one-to-one .

Step 3: Joining them at the common call-context, i.e. find out possible feature-using situations that allow the two feature to have common call context.

Case 1: ***AC** :: 1st_CC* == ***CW** :: 1st_CC* (Pcssible Case)

This means one user subscribes both AC and CW. The first call contexts of both features are compatible because their subscribers both act as the caller party in the two call contexts. Thus under this situation, AC and CW share the first call context at any time.

Case 2: ***AC** :: 1st_CC* == ***CW** :: 2nd_CC* (Possible Case)

This situation happens when the AC subscriber ***x*** first calls the CW subscriber ***a*** and then gets another call from ***b*** .

Case 3: ***AC** :: 2nd_CC* == ***CW** :: 1st_CC* (Possible Case)

This situation happens when the AC subscriber ***x*** is talking with a third party when the CW subscriber ***a*** calls ***x*** .

Case 4: ***AC** :: 2nd_CC* == ***CW** :: 2nd_CC* (Impossible Case)

According to the CCMs, both AC and CW subscribers act as called parties in the second call context. But in this case, one needs to be a caller party.

So Case 4 is discarded. In Case 1, CW and AC share the same subscriber and thus share the first call-context. They also share the second call-context since theirs are compatible, which means their second call contexts constitute another common call context. Figure 5-5 intuitively expresses such a feature-using case.

Step 4 (Loop 1): Pick up one case and based on this feature-using case, we replace formal parameters with actual values so that this case can be realized. In this example we pick Case 1. But if more than one case exists, we can pick up any. Others will be processed later in **Step 4** (Lopp2) and **Step 4** (Loop 3) when going back from **Step 6.**

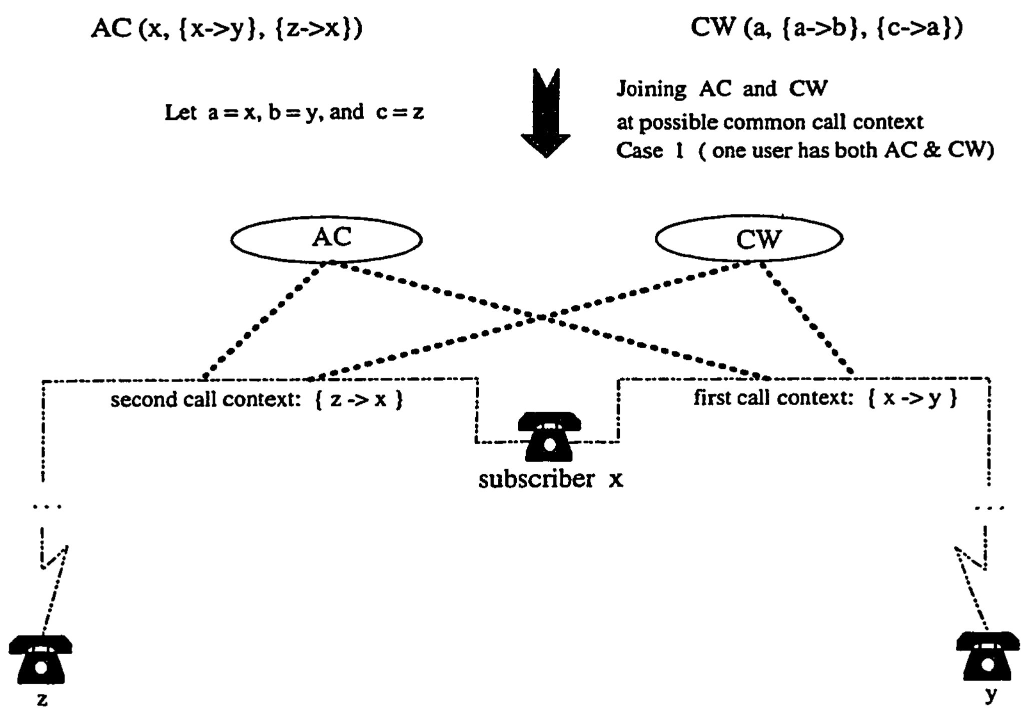

AC (x, {x->y}, {z->x})
CW (a, {a->b}, {c->a})
Let a = x, b = y, and c = z
Joining AC and CW
at possible common call context
Case 1 (one user has both AC & CW)
AC
CW
second call context: { z -> x }
first call context: { x -> y }
subscriber x
z
y

Figure 5-6: AC and CW having the same subscriber

(Loop 1 execution): Let us presume that *111* subscribes both features AC and CW, their partner in the first call-context is *222*, partner in second call-context is *333*. Then the two features can be represented as follow:

Feature Name	**Answering Call**	**Call Waiting**
Description	(omitted)	(omitted)
CCM	*AC (111, { 111->222 }, { 333->111 })*	*CW (111, { 111->222 }, { 333->111 })*
Triggering	*{111->222}::111::O_Active* AND *{333>111}::111:: DP__T_Called_Party_Busy*	*{111->222}::111::O_Active* AND *{333>111}::111::DP__T_Called_Party_Busy*
Cascading		
Data Manip.		
Resource Requested	VoiceAnnouncer 1 VoiceRecorder 1	ToneGenerator 1 Timer 1 2-leg call switch 1
Operation Requested		*{111->222}::111::*OnHold*(222)* AND *{333->111}::111::*OnHold*(333)*

Signal Claiming		*{111->222}::111::O_Active::Hook_Flash* **AND** *{333->111}::111::T_Active::Hook_Flash*
Line Assum.	1-Line-1-User	1-Line-1-User

(Loop 2 execution): Pick up the feature-using situation of Case 2. Assume *111* is the AC subscriber and *333* is the CW subscriber. We have:

Feature Name	**Answering Call**	**Call Waiting**
Description	(omitted)	(omitted)
CCM	*AC (111, { 111->222 }, { 333->111 })*	*CW (333, { 333->111}, { 444->333 })*
Triggering	*{111->222}::111::O_Active* **AND** *{333->111}::111::DP__T_Called_Party_Busy*	*{333->111}::333::O_Active* **AND** *{444->333}::333::DP__T_Called_Party_Busy*
Cascading		
Data Manip.	-	
Resource Requested	VoiceAnnouncer 1 VoiceRecorder 1	ToneGenerator 1 Timer 1 2-leg call switch 1
Operation Requested		*{333->111} :: 333 ::* OnHold*(111)* **AND** *{444->333} :: 333 ::* OnHold*(444)*
Signal Claiming		*{333->111}::333::O_Active::Hook_Flash* **AND** *{444->333}::333::T_Active::Hook_Flash*
Line Assum.	1-Line-1-User	1-Line-1-User

(Loop 3 execution): Pick up the feature-using situation of Case 3. Assume *111* is the AC subscriber and *222* is the CW subscriber. We have:

Feature Name	**Answering Call**	**Call Waiting**
Description	(omitted)	(omitted)
CCM	*AC (111, { 111->222 }, { 333->111 })*	*CW (222, { 222->444}, { 111->222 })*
Triggering	*{111->222}::111::O_Active* AND *{333->111}::111::DP__T_Called_Party_Busy*	*{222->444}::222::O_Active* AND *{111->222}::222::DP__T_Called_Party_Busy*
Cascading		
Data Manip.		
Resource Requested	VoiceAnnouncer 1 VoiceRecorder 1	ToneGenerator 1 Timer 1 2-leg call switch 1
Operation Requested		*{222->444}::222::*OnHold*(444)* AND *{111->222}::222::*OnHold*(111)*
Signal Claiming		*{222->444}::222::O_Active::Hook_Flash* AND *{111->222}::222::T_Active::Hook_Flash*
Line Assum.	1-Line-1-User	1-Line-1-User

Step 5 : Searching for conflict in the "concretized" CCM-FCAs.

(Loop 1 execution): Conflict is found in the "**Triggering**" field, this is revealed by the fact that the triggering criteria of both features are identical: "*{111->222}::111::O_Active* **AND** *{333>111}::111::DP__T_Called_Party_Busy*". No priority information is specified in "**Cascading**" field. Due to this conflict, we can conclude that FI exist between AC and CW.

(Loop 2 execution): No conflict is found which means no FI is found between AC and CW under this feature-using case.

(Loop 3 execution): No conflict is found which means no FI is found between AC and CW under this feature-using case.

Step 6: Output detection result and exit because we have no other feature-using case. When there are still feature-using cases not analyzed, go back to **Step 4**.

We have detected an FI in the case when one user subscribes both AC and CW. The main reason of this FI is that the two features have conflict in their **Triggering** field of CCM-FCAs, which indicates their triggering criteria may not compatible.

5.3.2 Detecting FI between TCS and 3WC

Now we consider another pair of features and the FI between them. Again the two features are explained first and then the FI and detection.

Feature: Terminating Call Screening (TCS)

This feature provides the capability of screening in-coming (terminating) calls against certain criteria that reflects subscriber preferences. TCS allows its subscriber to specify what kinds of in-coming calls are permitted to reach him/her. The screening criteria are usually a list of call numbers and optionally can be combined with day and/or time, etc. As long as a caller's number is NOT on the list, this TCS subscriber should not be connected to that caller.

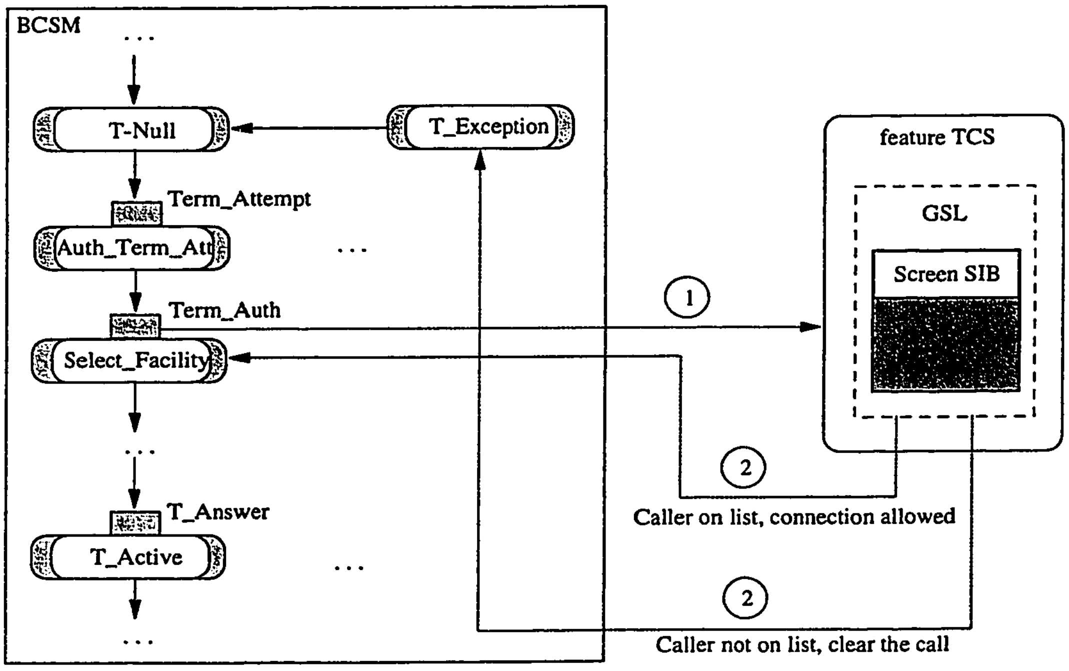


Figure 5-7: Invocation of feature Terminating Call Screening

Figure 5-7 indicates how feature TCS works for its subscriber. Similar to section 5.3.1, we use BCSMs to represent the basic call processing of the network. Because TCS only involves in one call-context, a BCSM is used to represent the first (and only) call context of TCS subscriber x. Suppose network user y is calling x, a T-BCSM will be launched in network switch that serves x to process this call. When it comes to the DP "**Term_Auth**", feature *TCS* jumps out and screens the in-coming call's caller number against the pregiven screening list and criteria. If the caller's number is on the list, TCS takes no further action and return to normal BCSM with POR (Point Of Return) "**Select_Facility**". Otherwise TCS will clear this call by returning to BCSM using a POR of "**T_Exception**" that indicates the call is aborted abnormally.

Feature: Three-Way Calling *(3WC)*

The Three-Way Calling feature allows its subscriber to establish a connection among three parties instead of a normal two-party call. Please note that, in order to set up a three-way call, as least as one of the three parties subscribes to 3WC would be sufficient.

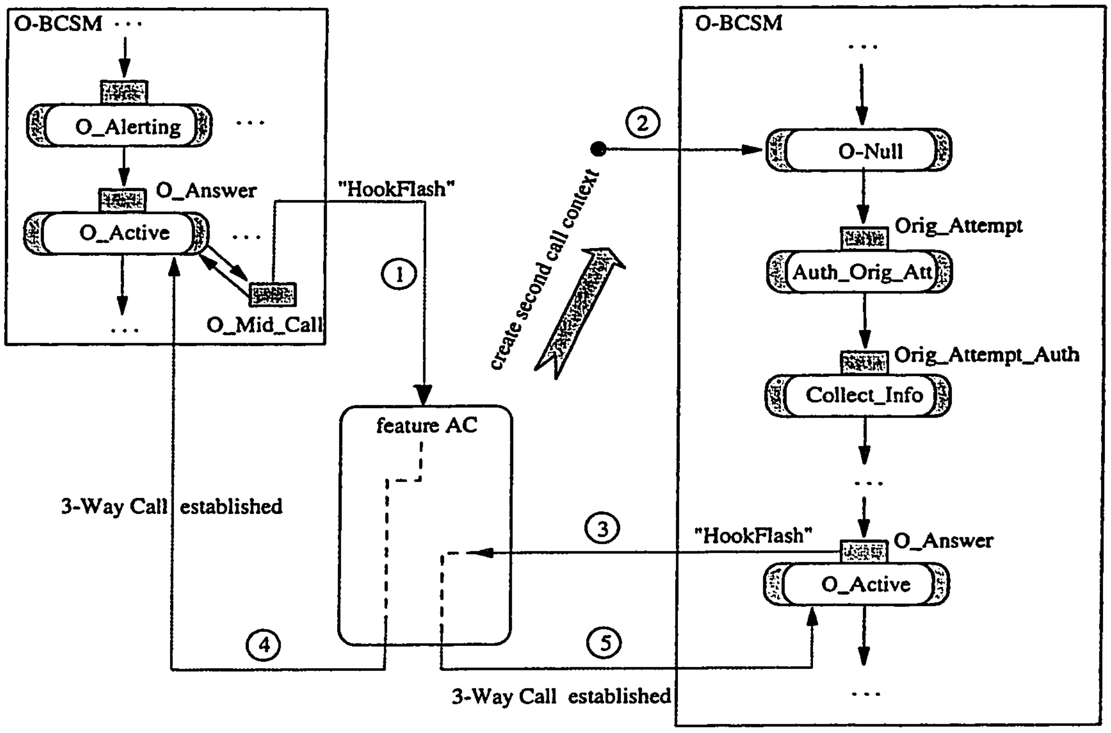


Figure 5-8: Invocation of the feature Three-Way Calling

Suppose x is a 3WC subscriber, y and z are two other parties. The using procedure of 3WC starts with x's calling y, and upon being connected, put y on hold. Then dial to call z and, upon successful connection, bring y back to form a three-party call.

Again we use BCSMs to illustrate how 3WC works for its subscribers in Figure 5-8. x first dial y's number to connect y. This is a normal two-party call without having to bother 3WC. After the call connection x-y is established, the BCSM (an O-BCSM, since x

initiated the call) is in PIC "**O_Active**". Then x uses an input signal "*Hook_Flash*" to ask for 3WC's service. 3WC suspends the execution of this O-BCSM and creates another BCSM (again an O-BCSM). In the second O-BCSM, if a "*Hook_Flash*" is received from x when this O-BCSM is in PIC "**O_Active**", 3WC will use a call bridge to connect all three parties into a single conversation.

FI between them and Detection

Terminate Call Screening (*TCS*) vs. Three-Way Calling (*3WC*)

> *Let us assume that x is a TCS subscriber, who puts y's number on his screening list to refuse the connection with y, z is a* 3WC *subscriber. Consider this situation: z calls y first, then they decide to bring x into their conversation. So z puts y on hold, calls x and brings x into a three-way connection upon x's answering. Now y is connected with x without being checked against the screening list of x. The* TCS *feature that x subscribes failed to work for x !*

Let us apply our FI detection approach step by step and see how the FI between TCS and 3WC was caught.

Step 1: Represent them in CCM. TCS has only one call context in which its subscriber acts as a called party. However 3WC involves two call contexts. In both contexts its subscriber serves as a caller party.

TCS (*x*, { *y->x* }) and *3WC* (*a*, { *a->b* }, { *a->c* })

Step 2: Give out CCM-FCAs of the two features. Here are the partial FCAs for the purpose of detecting FI between TCS and 3WC. Contents in the FCAs are explained below.

Feature Name	**Terminating Call Screening**	**Three-Way Calling**
Description	(omitted)	(omitted)
CCM	*TCS (x, { y->x })*	*3WC* (a, { a->b }, { a->c })
Triggering	*1st_CC::x::DP__T_Term_Auth*	*1st_CC::a::O_Active::Flash_Hook*
Cascading		
Data Manipulation	NOT Write (*1st_CC::CallData__Connected_Party*)	Write(*1st_CC::CallData__Connected_Party*) **AND** Write(*2nd_CC::CallData__Connected_Party*)

Resource Requested		Call Bridge 1
Operation Requested		*1st_CC::*OnHold*(b)*
Signal Claiming		*1st_CC::a::O_Active::Hook_Flash* AND *2nd_CC::a::O_Active::Hook_Flash*
Line Assumption	1-Line-1-User	1-Line-1-User

"Triggering" field

TCS needs to be triggered when an in-coming call has passed other terminating restraints such as bearer capability check. Technically, the triggering criteria are the in-coming call's reaching the DP (detection point) "**DP__T_Term_Auth**". *3WC* is invocated when its subscriber indicates the intention of a three-way call by using "*Hook_Flash*" when (s)he is in the "talk" state of the first normal two-way call. Thus we have "*1st_CC::a::O_Active::Flash_Hook*" in this field.

"Data Manipulation" field

TCS means to guarantee that only screened and authorized callers will be connected to its subscriber. In order to achieve this, TCS first performs screening on the caller, and once the caller is allowed to connect, protects this data from being modified during this call (to avoid the situation of caller being replaced or new party being added after the screening process). That is why we have the operation request "**NOT** Write *(1st_CC :: CallData__Connected_Party)*" in the field. 3WC needs to setup a three-way call by combining two normal calls. It must overwrite the call data "*CallData__Connected_Party*" in both normal two-way calls to contain all three parties' numbers so that a connection among three parties can be established. That is why " *Write (1st_CC :: CallData__Connected_Party)* **AND** *Write (2nd_CC:: CallData__Connected_Party)* " is included here.

"Resource Requested" field

TCS seems to require no special call resources while 3WC needs a call bridge that can connect the three parties into a single conversation.

"Operation Requested" field

3WC needs to put the partner of first call on hold and dial to the third party in order to setup a three-way call. So the operation of "*1st_CC::*OnHold*(b)*" is here.

"Signal Claiming" field

TCS executes automatically without asking subscriber's opinion when a call is coming in. So no user signal is claimed. 3WC must claim the "*Hook_Flash*" signal in the "talk" state of the first call as an indication of user's desiring a three-way call. It also claims the same signal in the "talk" state of the second call as subscriber indication of merging the two calls.

Step 3: Joining them at the common call-context, i.e. find out possible feature-using situations that allow the two feature to have common call context.

Case 1: One user subscribes both ***TCS*** and ***3WC*** (Possible Case)

However, under this situation, it is for sure that no FI exists between TCS and 3WC because two features must share at least one common call context in order to produce any FI but TCS and 3WC cannot share any call context in this situation. TCS needs its subscriber to be a called party but 3WC requires its subscriber to act as caller party in both call contexts.

Case 2: ***TCS*** *:: 1st_CC ==* ***3WC*** *:: 1st_CC* (Possible Case)

This means the subscriber of 3WC called the TCS subscriber first and then calls another user. Under this situation, TCS and 3WC share the first call context.

Case 3: ***TCS*** *:: 1st_CC ==* ***3WC*** *:: 2nd_CC* (Possible Case)

This corresponds to the situation where the 3WC subscriber called another user first and then calls the TCS subscriber. Under this situation, TCS and 3WC share the second call context.

Please note that TCS only intervenes in a call when its subscriber is the called party. 3WC needs its subscriber to act as caller party in both call contexts. So the call context of TCS is compatible with both call contexts of 3WC. The first case is discarded and the later two cases will be analyzed in order to check for possible FI.

Step 4 (Loop 1 execution): We pick up Case 2 and assign actual user numbers to realize such a case. Let us assume that user 111 subscribes TCS, user 222 subscribes 3WC and the third party's number is 333. So we have: [after this step go to **Step 5** (Loop 1)]

Feature Name	Terminating Call Screening	Three-Way Calling
Description	(omitted)	(omitted)
CCM	*TCS (111, { 222->111 })*	*3WC (222, { 222->111 }, { 222->333 })*
Triggering	*{222->111}::111::DP_T_Term_Auth*	*{222->111}::222::O_Active::Flash_Hook*
Cascading		
Data Manip.	NOT Write (*{ 222->111}::CallData__Connected_Party*)	Write (*{ 222->111 } :: CallData__Connected_Party*) AND Write (*{ 222->333 }:: CallData__Connected_Party*)
Resource		Call Bridge 1
Operation		*{ 222->111 }* :: 222 :: OnHold(*111*)
Signal Claiming		*{ 222->111 }::222::O_Active::Hook_Flash* AND *{ 222->333 }::222::O_Active::Hook_Flash*
Line Assum.	1-Line-1-User	1-Line-1-User

Step 4 (Loop 2 execution): We pick up Case 3 and assign actual user numbers to realize such a case. So we have: [after this step go to **Step 5** (Loop 2)]

Feature Name	Terminating Call Screening	Three-Way Calling
Description	(omitted)	(omitted)
CCM	*TCS (111, { 222->111 })*	*3WC (222, { 222->333 }, { 222->111 })*
Triggering	*{222->111}::111::DP_T_Term_Auth*	*{222->333}::222::O_Active::Flash_Hook*
Cascading		
Data Manip.	NOT Write (*{ 222->111} :: CallData__Connected_Party*)	Write (*{ 222->111 } :: CallData__Connected_Party*) AND Write (*{ 222->333 }:: CallData__Connected_Party*)
Resource		Call Bridge 1
Operation		*{ 222->333 }* :: 222 :: OnHold(*333*)
Signal Claiming		*{ 222->111 }::222::O_Active::Hook_Flash* AND *{ 222->333 }::222::O_Active::Hook_Flash*
Line Assum.	1-Line-1-User	1-Line-1-User

Step 5 (Loop 1 execution)**:** Looking for conflict in the CCM-FCAs to determine if there are indications of FI between the two features under this case [go to **Step 6** after this step].

From the table we can see the conflict between their **"Data-Manipulation"** fields.

The data operation "**NOT** Write *({222->111} :: CallData__Connected_Party)*" required by TCS conflicts with the "Write *({ 222->111 } :: CallData__Connected_Party)*" requested by 3WC. This indicates that TCS protects *"CallData__Connected_Party"* from being overwritten but 3WC needs to overwrite it in order to bring one more party into the connection.

Step 5 (Loop 2 execution)**:** Looking for conflict in the CCM-FCAs to determine if there are indications of FI between these two features under Case 3.

Same as in Loop 1, the conflict between their **"Data Manipulation"** fields is found, which indicates the FI between TCS and 3WC under this call case.

Step 6: Here we output the detection results. In Loop 1 execution continues with Step 4 (Loop2), and in the Loop 2 execution exits.

From the above detection practice, we recognized that there are actually two situations associated with the two features TCS and 3WC that can produce FI. Both when the TCS subscriber acts act the first call partner and the second call partner of the 3WC subscriber.

Only now that we realize the natural language description presented earlier in this section about the FI between TCS and 3WC is *INCOMPLETE* because only Case 3 is mentioned. The Case 2 can also produce FI but was overlooked. That is why so many researchers chose Formal Description Techniques (FDT) in their study of FI problem, because FDT helps to reveal every possible call situation that human-beings may neglect.

5.3.3 Detecting FI between ACC and POTS

The feature Account Card Calling may have FI with even an automated normal POTS (Plain Old Telephone System) user line. We give the details below.

Feature: Account Card Calling

The ACC feature allows a user to make a call from almost any telephone (that can read the ACC card) and have the charges for the call automatically debited to a telephone account number stored in the card that was pre-subscribed with the network operator. The user usually is given a PIN (personal identification number) as cardholder in case of a missing or stolen card.

If the cardholder makes only one call, he just inserts the card, inputs the PIN, dials the number, talks and hangs up with no problem. However, when multiple calls need to be made, it would be tedious to repeat this procedure call after call. Hence the ACC allows the cardholder to press "#" key for placing another call instead of hanging up, re-applying the card and re-inputting the PIN.

POTS : Basic Call Processing

Here we are talking about a normal POTS line that is connected to an automated machine that answers all the in-coming calls. There are plenty of such examples such as SAAQ (Quebec Automobile & Driver Administration), telephone banking, info-line of Revenue Canada, user-service line of investment companies, etc.

These types of automated machine-processing are actually not features provided by network operator but normal usage of POTS by certain telephone users. Only that the ways they use the POTS are somewhat different from normal calls that simple involve two persons' talking. In such machine-answered call services, callers are instructed step by step to input certain information by pressing the number keys in order to perform some specific tasks (such as getting information, register car-tests, etc.). Quite often, the "*" and "#" keys are also needed in these interactive procedures.

As an example, the mutual fund investment service-line of Bank of Montreal uses automated machines to serve its customers. When a customer wants to get up-to-date information about his/her mutual fund account, the machine instructs him/her to input the account number followed by the "#" key. This operation usually succeeds without any problem, but we will see there is exception.

FI between them and Detection

Account Card Calling (*ACC*) vs. Machine-Answered POTS Line **(POTS)**

*Suppose user **a** is using the* ACC *feature to place a call to the Bank of Montreal mutual fund service line. **a** applies his card, inputs PIN and dials the number. Once connected successfully, **a** wants to know some detail information about his account. He listens to the machine's instruction and inputs his account number and at last press the "#" key. Surprisingly, **a** was disconnected instead of getting the information he needs. No matter how many times he tried, none succeed !*

What is the problem? No, the bank's service line and machine are perfect. The reason behind this is that the ACC *feature, which **a** was using to call the bank service, undesirably interfered the functioning of the normal POTS line of the bank service.* ACC *captured the "#" key and interpreted it as an intention of **a** to place another call under the* ACC *feature. So it disconnected **a** from the current call and was waiting **a** to dial another number.*

The interaction between ACC and POTS is depicted in Figure 5-9. When user ***a*** is in the process of a normal call, ACC unexpectedly interrupted him because of misunderstanding of the "#" key pressed by ***a***.

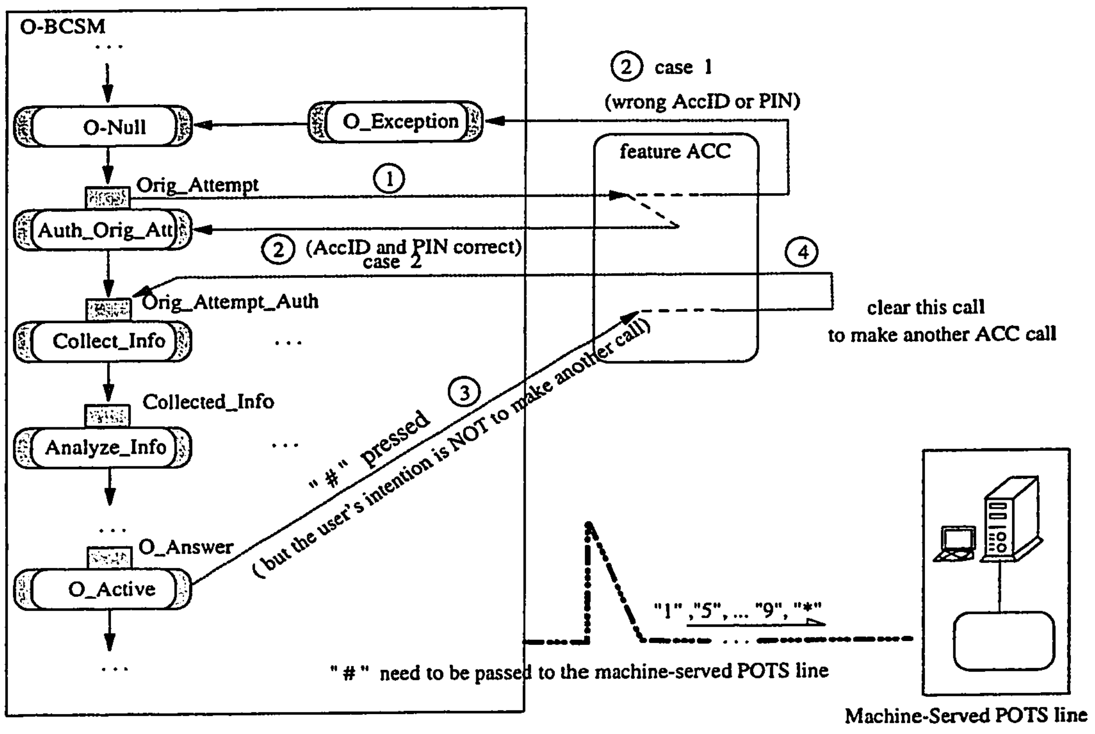


Figure 5-9: Interaction between feature ACC and normal POTS line

Let us apply our FI detection approach step by step and see how the FI between ACC and machine-answered POTS was caught.

Step 1: Represent them in CCM. Both the ACC and POTS have only one call-context.

Account Card Calling:	*ACC (x, { x -> y })*
Machine-Answered POTS:	**POTS** *(a, { b -> a })*

Step 2: Give out CCM-FCAs of the two features. Here are partial CCM-FCAs for the purpose of able to detect FI between them. We only explain the fields that are useful with detection of this FI.

Feature Name	**Account Card Calling**	**Machine-Answered POTS**
Description	(omitted)	(omitted)
CCM	*ACC (x, { x -> y })*	**POTS** *(a, { b -> a })*
Triggering	*1st_CC::x::Orig_Attempt*	*1st_CC::a::T_Active*
Cascading		
Data Manipulation		
Resource Requested		
Operation Requested		
Signal Claiming	*1st_CC::x::O_Active::Pound_Key*	*1st_CC::b::O_Active::Pound_Key*
Line Assumption	1-Line-1-User	1-Line-1-User

"Triggering" field

ACC is supposed to begin its work once a user picks up the phone that indicates the intention of making an out-going call. Thus the triggering criteria of ACC are "*1st_CC::x::Orig_Attempt*" in this field.

The Machine-Answered POTS begins its task when a user has successfully connected to this line. Technically, when this BCSM (a T-BCSM because this line accepts a call) is in the PIC "**T_Active**" that means the two parties are in the state of ready to "talk".

"Signal Claiming" field

ACC claims the "#" key when the caller is in "talk" state to convey the meaning of placing another ACC call. Machine-Answered POTS has claimed the caller's "#" key in the "talk" state to signify the end of a mutual fund account number.

Step 3: Find out possible feature-using situations that allow the two features to have common call context. Here we have only one case where the user utilizes ACC to call the Machine-Answered POTS line.

Case 1: *ACC :: 1st_CC* == **POTS** *:: 1st_CC* (Possible Case)

Step 4: Use actual values to realize the above call case. Let us assume the caller uses the phone *111* to call the bank service line *999*. Now we have:

Feature Name	**Account Card Calling**	**Machine-Answered POTS**
Description	(omitted)	(omitted)
CCM	*ACC (111, { 111 -> 999 })*	**POTS** *(999, { 111 -> 999 })*
Triggering	*{ 111 -> 999 }::111::Orig_Attempt*	*{ 111 -> 999 }::999::T_Active*
Cascading		
Data Manip.		
Resource		
Operation		
Signal Claiming	*{ 111 -> 999 }::111::O_Active::Pound_Key*	*{ 111 -> 999 } ::111::O_Active::Pound_Key*
Line Assum.	1-Line-1-User	1-Line-1-User

Step 5: Searching for conflict in the CCM-FCAs. Easily we discovered that both of them have claimed the same user input signals in the same PIC of the caller's BCSM that is *"{111 -> 999}::111::O_Active::Pound_Key"*. This means only one will be able to consume this signal, and thus one of them will be prevented from fulfilling its designated task. The FI between ACC and Machine-Answered POTS has been successfully detected.

Step 6: There is no other call case, output result and exit.

5.3.4 Detecting FI between MCI and UN

Here are the explanations of the features Malicious Call Identification (MCI) and Unlisted Number (UN), followed by the introduction of FI between them and how the FI is detected using our approach.

Feature: Malicious Call Identification (MCI)

MCI enables a user to request that the source of an incoming call, which is considered to be of malicious nature, to be identified and registered.

Feature: Unlisted Number (UN)

The UN request at no time should its subscribers' number be released when a call is made from any of its subscribers' line.

FI between them and Detection

Malicious Call Identification (*MCI*) vs. Unlisted Number (*UN*)

*Say **x** subscribes* MCI, *and **y** subscribes* UN. *Let us presume that **y** called **x**, **y**'s number is not known to **x**. However, if **x** declares that this was a malicious call, then **x** will get the number of **y** by pressing a trigger button (such as "*" key) and utilize the* MCI *feature. Thus **y**'s* UN *feature is now nullified. And furthermore, if **y** denies that it was of malicious nature, how can a network operator knows if this was really a malicious call or **x** was lying in order to get **y**'s number?*

How did we detect this interaction?

Similar to the previous examples, we performed each of the steps and we found a conflict in their "**Data Manipulation**" fields. MCI needs to be able to perform "Deliver *({222->111}::CallData_Conneted_Party::222)*" but UN insisted that "**NOT** Deliver *({ 222->111 } :: CallData__Connected_Party::222)*" should be maintained. This conflict indicates inevitable feature interaction between the two features MCI and UN.

Feature Name	Malicious Call Identification	Unlisted Number
Description	(omitted)	(omitted)
CCM	MCI (111, { 222->111 })	3WC (222, { 222->111 })
Triggering	{222->111}::111::T_Active::Star_Key	{222->111}::222::DP__Orig_Auth
Cascading		
Data Manipulation	Deliver ({222->111}::CallData_Conneted_Party::222)	NOT Deliver ({ 222->111 } :: CallData__Connected_Party::222)
Resource Requested		
Operation Requested		
Signal Claiming	{222->111}::111::T_Active::Star_Key	
Line Assumption	1-Line-1-User	1-Line-1-User

As shown above, MCI needs to deliver the caller's number to the MCI subscriber to fulfill its designated task when its subscriber requests so. However UN inhibits such a delivery. Thus this is a conflict, and only one of them is allowed to function well.

5.4 Discussion of our Approach

After introducing our FI detection approach and application examples, we discuss and evaluate its performance and completeness below.

5.4.1 Performance

Our feature interaction detection method has been applied to the Bellcore (U.S.A) Feature Interaction Benchmark that systematically classifies various known feature interactions. Currently we are dealing with the FIs among the user features that provide enhanced call functions to the end users of telecommunication networks. Features used for network management purpose (see [Cameron 93]) are out of the scope of the thesis and thus not covered.

By using our approach, 16 out of 18 user-level FIs listed in the Bellcore FI Benchmark can be detected. Here is a list of the feature interactions in the benchmark and what kind of conflict we found when detecting them.

	Feature Interaction	Detectable	What Conflicts	Comments
Ex1.	CW vs. AC	Yes	Triggering	Same triggering criteria.
Ex2.	CW vs. 3WC	Yes	Triggering	Non-disjoint triggering criteria.
Ex3.	911 vs. 3WC	Yes	Operation Requested	3WC needs to put 911 operator on hold, but 911 cannot be put on hold.
Ex4.	TCS vs. ARC	Yes	Cascading	TCS to be triggered first but FI exist when ARC is processed before TCS.
Ex5.	OCS vs. ANC	Yes	Resource Requested	Both features requested a query but the network only allow one query.
Ex6.	OPS vs. OCS	Yes	Data Manipulation	OPS needs to modify call data while OCS protect call data from being overwritten after screening.
Ex7.	CCC vs. VM	Yes	Signal Claiming	CCC and VM both claim "#" key.
Ex8.	MBS-ED vs. CENTREX	Yes	Signal Claiming	Both claim four-digit number keys.
Ex9.	CF vs. OCS	Yes	Data Manipulation	OCS protect call data, CF modifies.
Ex10	CW vs. PCS	Yes	Line Assumption	CW assumes 1-to-1 mapping between directory number and physical line, but PCS wants n-to-1.
Ex11	OCS vs. MDNL-DR	Yes	Line Assumption	OCS needs 1-to-1,but MDNL-DR not.
Ex12	OCS vs. CF	Yes	Data Manipulation	OCS protect call data, CF modifies.
Ex13	CW vs. ACB	Yes	Data Manipulation	CW hide real call state, ACB needs it.
Ex14	CW vs. CW	Yes	Signal Claiming	Two instances of CW react to the same user input signal "Hook_Flash".
Ex15	CW vs. 3WC	Yes	Signal Claiming	Both features claim "Hook_Flash".
Ex16	CND vs. UN	Yes	Operation Requested	CND asks for caller number delivery, but UN prevents it being delivered.
Ex17	CF vs. CF	No		Not real FI (see below)
Ex18	ACB vs. ARC	No		Not real FI (see below)

Table 5-1: Benchmark performance of our FI detection approach

The reasons why I think the last two FIs are "*Not Real FI*" are as follows:

- About Ex17. FI between Call Forward (***CW***) vs. Call Forward (***CW***)

  *"A customer with **CF** can redirect calls to any number. As a result, anyone can 'accidentally' forward all incoming calls to his/her own number and create an infinite loop. Moreover, calls for **a** can be 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. "*

  I think under such situations, the feature ***CF*** is doing its work without being interfered. The task of ***CF*** is to forward calls to another directory number. ***CF*** assumes no guarantee on the consequences that may result from this forwarding action. As long as ***CF*** succeeds in forwarding all incoming calls to the given destination, no FI occurred with ***CF***. The overall consequence is not the responsibility of the feature ***CF*** !

- About Ex18. Auto Call Back (***ACB***) vs. Auto ReCall (***ARC***)

  The author described a situation when ***ACB*** and ***ARC*** perform their action (call the other party) exactly at the same time repeatedly, thus prevent either of them to fulfill their tasks. I do not think this is a FI due to two reasons:

  a. *During the procedure, both features are doing what they are supposed to do. Both features are NOT interfered, they will continue trying until one of them succeed and then the other will succeed later.*

  b. *Even without the two features, user **a** and **b** may still meet above problem if **a** and **b** call each other exactly at the same time again and again. Is that right ?*

Beside already known feature interactions, we have detected a few unpublished (neither listed on Bellcore Benchmark [Cameron 93] nor on EuresCom Benchmark [Kimbler 95]) feature interactions with our method. This revealed the detection power of our approach.

Last-Caller Number Display	*vs.*	*Unlisted Number*
Account Card Calling	*vs.*	*Machine-Answered POTS*
Originated Call Screening	*vs.*	*1-800 Free Call*
Malicious Call Identification	*vs.*	*Unlisted Number*

5.4.2 Completeness

Our method can detect many kinds of FIs including most known ones and a few unknown ones as shown above. However, we need to notice that:

- Although we cannot find more at this time, the reasons that can lead to FIs may not be limited to those that we have listed out. Other issues could also lead to new types of FIs.
- The FI detection condition we use is **sufficient condition** but may not be the **necessary condition** of causing such FIs.
- If we have detected FIs among features, we can find at least one call situation that reproduces this FI. However, these FIs may not happen under other feature-using situations.

This FI detection approach presents a unique view of the feature (Call Context Model) and feature interaction cause analysis (environmental assumption violation). It is considered to be effective and efficient for practical feature interaction detection at the present stage of FI research.

5.5 Chapter Summary

In this chapter, we first introduced the basis and general principle of our feature interaction detection approach. Secondly, description and step by step explanation of our method are given to illustrate the detail of how it is applied to detect feature interactions. Then four application examples that successfully detect different kinds of feature interactions have been presented to help understand our idea and scheme. After that, we analyzed the performance and completeness of our approach.

Our detection method is based on two important concepts: Call Context Model (CCM) and Feature Context Assumptions (FCA). The CCM describes a feature's involvement of call connections as well as outlining feature's behavior with respect to specific elements inside each call context. FCA contains information about various aspects of environmental assumptions that must be satisfied so that the feature can run and perform its task correctly.

After some steps of information processing, our FI detection is realized by searching for conflict in the corresponding fields of the CCM-FCAs of the "concretized" features. A conflict in any of the seven fields is considered as an indication of feature interaction between or among the features.

Our detection method has achieved good performance on Bellcore (U.S.A) FI Benchmark. Sixteen out of eighteen known feature interactions can be detected. In addition, a few unpublished feature interactions were also discovered by this method. Our detection criteria are a sufficient condition of the existence of FIs but may not be the necessary condition.

Chapter 6 Summary and Future Work

Providing telecommunication features that offer various added functions is the most feasible approach of enhancing the functionality and user-customizability of existing networks. Intelligent Network (IN) is proposed by ITU-T as a new infrastructure for future telecommunication networks in order to allow fast and cost-effective introduction of new features into the networks. IN aims to separate the Basic Call Processing (BCP) functions of the normal switches from the "network intelligence" that provides powerful and customizable communication capabilities. This makes it feasible to develop and deploy large number of features in a relatively short period of time.

Feature Interaction (FI) problem refers to all kinds of interference among multiple concurrent features that prevent at least one of them from performing designated tasks correctly. This problem has been a major hindrance of feature development and utilization for years and has been proved to be tough to solve. No general solution is available despite the fact that it was first discovered as early as in the late 80's.

6.1 Summary

In this thesis, we mainly have been focusing on Feature Interaction problem in the environment of Intelligent Network systems.

In chapter 1, we explained the concepts of telecommunication features, IN systems, and Feature Interaction problem among multiple features. General knowledge about the Formal Description Technique (FDT) was also presented. In addition, related work in this area and a brief overview of our work are given thereafter.

Chapter 2 first introduces the origin and motivation of IN, the major functional requirements of IN, and the two key technologies (Common-Channel Signaling and Non-Switching Node) that have made IN technically feasible. The ITU-T proposed Intelligent Network Conceptual Model (INCM) is introduced as important background knowledge to understand this thesis. The four planes defined in INCM are service plane (SP), global functional plane (GFP), distributed functional plane (DFP), and physical plane (PP). They represent distinctive descriptions of the same IN system from different perspectives and at different abstraction levels.

Chapter 3 consists of three major parts: the main characteristics of the language SDL, the IN system model that we worked out using SDL, and our comments about SDL in IN system modeling.

We have worked out a study model of IN-structured systems based on ITU-T Q.12xx series Intelligent Network recommendations. The modeling tool we use is the formal description language SDL that has been widely used in telecommunication area. We have surveyed and compared different mapping methods adopted by other researchers. Based on our study goal, we proposed our own scheme of modeling IN architecture, each functional entity and their relationships using appropriate SDL concepts. This model can be used to study the recommended IN architecture as well as analyze the telecommunication features and possible FIs among them.

Chapter 4 presents two important concepts we used to outline features for FI detection purpose. One of the important issue in FI detection is how to describe the features concisely and yet precisely, and what kind of information of features is really needed for the purpose of FI detection.

We proposed the Call Context Model (CCM) that can describe the features in a manner suitable for FI detection. This model expresses the features with respect to each call connection involved in. Feature behaviors are described using specific elements (such as call data, call states, user input) inside these call connections. The Feature Context Assumption (FCA) captures the most important assumptions that the features made about its running environment. These assumptions indicate the conditions that must be met to guarantee features' correct behaviors. FCA is specifically designed for FI detection and, complex information about features can be summarized into much less complex feature assumptions using FCA.

Chapter 5 demonstrates our new FI detection approach. Based on the CCM-FCAs of the features under study, the essential technical principle of our approach is conflict searching in "concretized" CCM-FCAs.

In order to concretize the features' CCM-FCAs, we need to find out all possible feature-using cases of the features and exclude those that can never produce FI. Then we assign actual user telephone numbers to the formal parameters in the CCM to realize each of the feature-using cases. Applying these numbers to each assumption field of the FCAs will get the "concretized" CCM-FCAs that actually represent instances of this feature. The conflicts found in the CCM-FCAs are considered as indications of FIs under certain feature-using cases. The type of conflict(s) found will give possible reason(s) of such feature interactions.

Our FI detection method has been applied to Bellcore Feature Interaction Benchmark and achieved very good results. In addition, a few unpublished new feature interactions were also detected using our method.

Chapter 6 gives a summary about the background of FI problem and our work in IN modeling and FI detection approach development. Major contributions and directions of future work are also presented.

6.2 Main Contributions of the Thesis

The major contributions of this thesis are modeling of IN system architecture and entities using formal description language SDL, and the development of a novel method for Feature Interaction detection in IN systems.

6.2.1 Contribution 1: Modeling of Intelligent Networks using SDL

Under the context of ITU-T defined Intelligent Network Conceptual Model, we worked out a model of IN systems using a formal description language SDL. This model characterized the architecture of IN system and the major entities in the system, including CCAF (Call Control Access Function), CCF/SSF (Call Control Function / Service Switching Function), SCF (Service Control Function) and SDF (Service Data Function), by using appropriate SDL concepts such as "**system**", "**block**", "**process**", "**process type**", "**procedure**", etc. The communications among these entities are realized using SDL "**signal route**" and "**channel**" that precisely simulate the actual data and control signal exchange among distributed network components.

We have seen significance of this model lies in the following points:

- Help to analyze the behavior of various IN entities and their relationships with emphasis on how they cooperate to fulfill BCP and feature functions.
- Facilitate the study of features, from their triggering checking, execution to the returning. In addition, we can be easier to trace the occurrences and reasons of feature interactions.
- Supported by automated software tool, this model can be used to verify the correctness and evaluate the detection ability of various feature interaction detection methods.

6.2.2 Contribution 2: A New Method for Feature Interaction Detection

The key point of feature interaction problem is the efficient and early detection. We have developed a new FI detection method that works on the feature development phase including specification, design and implementation stages. Our method is based on analyzing feature assumptions rather than feature behaviors. This will ensure our detection method is practically usable when we want to address FI problems among large number of features. Moreover, by considering and collecting feature assumptions throughout each phase of development stage, we are able to discover some kinds of FIs that only occur after specification stage.

The core idea of the method is based on two important concepts we presented: Call-Context Model (CCM) and Feature Context Assumptions (FCAs). CCM enables the feature to be specified with regard to call connections it is involved in. FCAs describes all the important environmental assumptions that must be met for a feature to work correctly. FIs can be detected by searching for conflicts in CCM-FCAs of multiple feature instances that share common call connection(s). Our FI detection method has been used to verify the previously known FIs as well as to discover unknown ones, both indicate satisfactory detection results.

6.2.3 Connection Between the Two Contributions

Our IN system model in SDL provides a usable platform on which we can analyze the behavior of IN functional entities and features. This model also helped us in performing FI cause analysis that contributed to the FI detection study.

Moreover, the IN model can be used to verify the detection result of our FI detection method. The correctness of FI detection result is proved if we can reproduce the corresponding FI in a specific feature running situation in our IN model (in SDL) with the help of the software tool - ObjectGeode.

6.3 Directions of Future Work

Feature Interaction problem in IN systems is an interesting but tough research topic. No general solution currently exists. We merely propose a practical new method for FI detection but cannot guarantee it covers all kinds of FIs. Based on the work we have carried out and described in this thesis, future study can be proceeded towards the following directions:

- ***Verification of FI Detection Results***

  An important continuation of our work will be the verification of FI detection results achieved by our detection method, especially for unknown feature interactions. This can be carried out by trying to produce a call situation where multiple feature instances are running and feature interaction(s) do occur among them as suggested by the results of our detection method.

  Based on the our Intelligent Network system model and the support of CASE tool ObjectGeode, we can perform simulations of various call situations with the feature instances running to serve different users. Normally, this simulation will be a "guided" simulation, which is commanded by human beings, towards the direction(s) that most likely to reproduce the suspected FIs. This person may get some hints on how to produce such FIs based on the type of conflict(s) detected when using our FI detection method.

- ***Extending from CS1 to higher Capability Sets***

  Currently our work is mainly based on ITU-T standardized IN CS1 and can deal with some features beyond CS1 such as "Automatic CallBack", "Caller Number Delivery" and "Multi-Dir. Number with Distinctive Ringing". With the standardization of CS2 and high Capability Sets, more IN functionality will be introduced and new processing and signaling capabilities will be appended. In addition, the IN architecture itself and the mapping between adjacent planes may evolve. It would be useful to extend the results described in this thesis so that it can be applied to CS2, CS3, ... CSn.

- ***Standardization of Description Sets***

  In order to be used widely in the industry, a series of complete and standardized sets that allow the assumptions of features to be specified uniquely are needed. These sets include Standard Operation Set, Standard Resource Set, Standard Call-Data Set, etc. Improving our work by representing feature assumptions using industry recognized standards would be a worthwhile future work.

- ***Modeling New Features***

  More features will be developed and introduced in the future. Trying to analyze them and detect possible unknown feature interactions among them will be a very interesting task and can further verify the detection power of our method.

Bibliography

[Aloni 97]	A. Alonistioti et al., "SDL-based Modeling and Design of IN/UMTS Hand-over Functionality", SDL '97: TIME FOR TESTING - SDL, MSC and Trends, A.Cavalli and A. Sarma (Edition) , 1997, Elsevier Science B.V.
[Astrid 93]	Astrid Nyeng and Birger M.P., "Approaches to the Specification of Intelligent Network Services in SDL-92", SDL '93: Using Objects, (Eds.) O. Faegemand and A. Sarma, Elsevier Science Publishers B.V., 1993.
[Bergs 97]	Jan Bergstra & Wiet Bouma, "Models for Feature Description and Interaction", Feature Interactions in Telecommunication Networks IV, edited by Peter Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 31-42, 1997 (FIW97, Montreal, Canada)
[Blom 95]	J. Blom, R. Bol, and L.Kempe, "Automatic Detection of Feature Interactions in Temporal Logic", Feature Interactions in Telecommunication Systems III, 1995, IOS Press.
[Bostr 94]	M. Bostrom and M.Engstedt, "Feature Interaction Detection and Resolution in the Delphi framework", Feature Interactions in Telecommunication Systems, 1994, IOS Press.
[Bowen 89]	T.F. Bowen et al., "The Feature Interaction Problem in Telecommunication Systems", 7th International Conference on Software Engineering for Telecommunication Switching Systems., July, 1989
[Bryce 94]	Bryce Kelly et al., "Feature Interaction Detection Using SDL Models", IEEE GlobeCom 1994, pp. 1857 - 1861

[Cameron 93] E. Cameron et al., "A Feature Interaction Benchmark for IN and Beyond", IEEE Communications Magazine, March, 1993, pp. 64-67

[Chen 97] Y.L. Chen, S. Lafortune and F. Lin, "Resolving Feature Interaction Using Modular Supervisory Control with Priorities", Feature Interactions in Telecommunication Networks IV, edited by Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 31-42, 1997 (FIW97, Montreal, Canada)

[Comb 94] P.Combes, and S.Pickin, "Formalization of a User View of Network and Service for Feature Interaction Detection", Feature Interactions in Telecommunication Systems, 1994, IOS Press.

[Comb 95] P. Combes et al., "MSCs to express Service Requirements as Properties on a SDL Model: Applications to Service Interaction Detection", 7th SDL Forum, Oslo, September 1995.

[Conor 95] Conor Morris and John Nelson, "An SDL Based Realization of an IN Service Development Environment", IS & N '95

[Csurgay 95] P.Csurgay & F.A Agesen, "A Distributed SDL-based Intelligent Network Laboratory", 7th SDL Forum Conference Proceedings, Oslo, September 1995.

[FIW 92] The First International Workshop on Feature Interactions in Telecommunication Software Systems, Florida, 1992.

[FIW 94] Second International Workshop on Feature Interactions in Telecommunication Software Systems, (Editions) L.G.Bouma and H.Vethuijsen, IOS Press, 1994

[FIW 95] Third International Workshop on Feature Interactions in Telecommunication Software Systems, (Editions) K.E. Cheng and T. Ohta, IOS Press, 1995

[FIW 97] Fourth International Workshop on Feature Interactions in Telecommunication Software Systems, (Editions) Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, 1995

[FaciL 91] M.Faci, L.Logrippo and B.Stepien, "Formal Specification of Telephone Systems in LOTOS", Computer Networks and ISDN Systems, North Holland, 1991, pp.52-67.

[FaciL 94] M.Faci and L.Logrippo, "Specifying Features and Analyzing their Interactions in a LOTOS Environment", Second International Workshop on Feature Interactions in Telecommunication Software Systems, IOS Press, 1994, pp. 136 - 151.

[Ferenc 89] Ferenc Belina and Dieter Hogrefe, "The CCITT Specification and Description Language SDL ", Computer Networks and ISDN Systems, vol. 16, (1988/89) pp. 311 - 341, Elsevier Science Publishers B.V. (North Holland)

[Gamme 94] A.Gammelgard and J.E.Kristensen, "Interaction Detection: A logical Approach", Feature Interactions in Telecommunication Systems, 1994, IOS Press.

[Geode] Introduction to ObjectGeode, Verilog Inc, France (please refer to www.verilog.fr or www.verilogusa.com)

[Gerard 91] Gerard J. Holzmann, "Design and Validation of Computer Protocols", Prentice Hall, New Jersey, 1991, ISBN 0-13-539925-4 (also available online at: cm.bell-labs.com/cm/cs/who/gerard/popd.html)

[Gupta 96] Naresh Gupta, "Message Sequence Chart description of some telephony features", "MSC description of Feature Interactions in Telecommunication Systems", Technical Project Report, ECE, Concordia University, 1996

[Hussein 93] M.Haj-Hussein, L.Logrippo, and J.Sincennes, "Goal Oriented Execution of LOTOS Specifications", Formal Description Techniques, by M.Diaz and R. Groz (Eds.), V. North Holland, 1993, pp. 311 - 327.

[ISO8807 89] ISO, IS8807, "LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behavior", May 1989.

[ISO9074 89] ISO/TC 97/SC 16, ISO 9074. Information Processing System – Open System Interconnection – Estelle: A Formal Description Technique Based on an Extended State Transition Model, 1989.

[ISO-L 89] ISO, Information Processing Systems – Open System Interconnection, LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behavior, 1989.

[Jalel 96] Jalel M., "Formal Specification and Feature Interaction Detection in

Intelligent Network", master thesis, University of Ottawa, 1996.

[Johan 97] Johan Blom, "Normalization of Requirements with emphasis on Feature Interaction Detection", Feature Interactions in Telecommunications and Distributed Systems IV, Petri Dini et al. (editions) IOS Press, 1997

[KhenB 92] M. Erradi, F. Khendek, R. Dssouli, and G. Bochmann, "Dynamic Extension of Object-Oriented Distributed System Specifications", First International Workshop on Feature Interactions in Telecommunications Software Systems, Florida USA, 1992

[Kimbler 95] K.Kimbler & Hugo Velthuijsen, "Feature Interaction Benchmark", Discussion paper for FIW'95 Panel on Benchmarking. FIW95, 1995

[Kimbler 97] K.Kimbler, "Addressing the Interaction Problem at the Enterprise Level", Feature Interactions in Telecommunication Networks IV, edited by Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 13-22, 1997 (FIW97, Montreal, Canada)

[Lin 94] F.J Lin & Y.J Lin, "A Building Block Approach to Detecting and Resolving Feature Interactions", Feature Interactions in Telecommunication Systems, pp. 86-119, 1994

[Lucidi 96] F.Lucidi et al., "Object-Oriented Modeling of AIN services with SDL-92", Technical paper, Sett. Elaborazione dell'Informazoine, Fondazione Ugo Bordoni, Rome, Italy, 1996 (nando@fub.it)

[MNakam 97] M. Nakamura et al., "Petri-Net based Detection Method for Non-Deterministic Feature Interactions and its Experimental Evaluation", Feature Interactions in Telecommunication Networks IV, edited by Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 138-152, 1997 (FIW97, Montreal, Canada)

[MSC 96] Recommendation Z.120, "Message Sequence Chart" , Tele-communication Standardization Group, International Tele-communication Union (ITU), 1996

[Nakam 97] Nakamur T. et al., "Analyzing Non-Determinism in Tele-communication Services using P-invariant of Petri-Net Model", Proceedings of IEEE INFOCOM'97, April 1997.

[Ohta 94] T. Ohta and Y.Harada, "Classification, Detection and Resolution of Service Interactions in Telecommunication Services", Feature Interactions in Telecommunication Systems, IOS Press, 1994

[Olsen 94] Anders Olsen et al., "Systems Engineering Using SDL-92", Elsevier Science B.V., 1994, ISBN 0-444-898727

[Q.12xx 93] ITU-T, New Recommendations Q1200--Q series: Intelligent Network Recommendation, COM XI-R 210-E, 1993

[Q.1200] ITU-T, Recommendation Q.1200, "Q-series Intelligent Network Recommendation Structure", 1993

[Q.1201] ITU-T, Recommendation Q.1201, "Principles of Intelligent Network Architecture", 1993

[Q.1204] ITU-T, Recommendation Q.1204, "Intelligent Network Distributed Functional Plane Architecture", 1993

[Q.1205] ITU-T, Recommendation Q.1205, "Intelligent Network Physical Plane Architecture", 1993

[Q.1211] ITU-T, Recommendation Q.1211, "Introduction to Intelligent Network Capability Set 1", 1993

[Q.1214] ITU-T, Recommendation Q.1214, "Distributed Functional Plane for Intelligent Network CS-1", 1993

[Ralph 97] Ralph B.B. et al, "Addressing Feature Interaction During Service Creation", Feature Interactions in Telecommunication Networks IV, edited by Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 364-370, 1997 (FIW97, Montreal, Canada)

[Roche 97] S.M. Rochefort et al., "An Exercise in Using Constructive Proof Systems to Address Feature Interactions in Telephony", Feature Interactions in Telecommunication Networks IV, edited by Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 364-370, 1997 (FIW97, Montreal, Canada)

[Rolv 96] Rolv Braek, "SDL Basics", Computer Networks and ISDN Systems, Elsevier Science B.V., 28 (1996), pp 1585 - 1602

[Simon 97] Simon T., Evan H.M, "Behavior Based Run-Time FI Detection and Resolution Approaches for IN", Feature Interactions in

Telecommunication Networks IV, edited by Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 254-270, 1997 (FIW97, Montreal, Canada)

[StepL 93] B.Stepien and L.Logrippo, "Status-Oriented Telephone Service Specification", Theory and Experiences for Real-Time System Development. AMAST Series in computing, Vol.2, World Scientific, 1994, pp.265-286.

[StepL 95] B.Stepien and L.Logrippo, "Feature Interaction Detection using Backward-Reasoning with LOTOS", Protocol Specification, Testing and Verification, XIV Proceedings of the 14th International Symposium, Vancouver, 1995, pp. 71 - 86.

[Stephen 97] Stephen M.Rochfort and H.J. Hoover, "An Exercise in Using Constructive Proof System to Address Feature Interactions in Telephony", Feature Interactions in Telecommunication Networks IV, edited by Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 329-341, 1997 (FIW97, Montreal, Canada)

[TadaY 94] Tadashi Ohta and Yoshio Harada, "Classification, Detection and Resolution of Service Interaction in Telecommunication Systems", Feature Interactions in Telecommunication Systems, (Eds.) W.Bouma and H.Vethuijsen, IOS Press, 1994.

[Turner 97] K.J. Turner, "An architectural foundation for relating features", Feature Interactions in Telecommunication Networks IV, edited by Petri Deni, Raouf Boutaba and Luigi Logrippo, IOS Press, pp. 226 - 241, 1997 (FIW97, Montreal, Canada)

[Visser 95] John Visser, "Tutorial on Intelligent Network Basics", IEEE IN'95 Workshop, Ottawa, May 1995.

[Z.100 93] ITU-T, Recommendation Z.100, "CCITT Specification and Description Language (SDL)", March 1993.

[Zave 93] P. Zave, "Feature Interactions & Formal Specifications in Telecommunications", IEEE Computer, August 1993, pp. 20 - 30.

[1998]