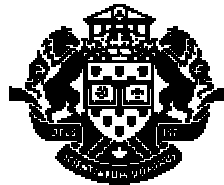

CNAP Specification and Validation: A Design Methodology Using LOTOS and UCM

Zhimei Yi

Thesis submitted to
the School of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Master of Computer Science

under the auspices of the Ottawa-Carleton
Institute for Computer Science



Université d'Ottawa · University of Ottawa
Ottawa, Ontario, Canada
January 2000

Copyright Zhimei Yi, Ottawa, Canada, 2000

Acknowledgments

I would like to thank all the people who assisted me in completing this work.

First, I would like to express my deep appreciation to my supervisor, Prof. Luigi Logrippo and his LOTOS group for their support and fruitful discussions throughout my graduate studies. Those discussions pointed me to the right direction, and greatly influenced on my research work. Also, Prof. Logrippos's accuracy in reviewing my drafts of the thesis has greatly improved its contents and presentation.

In particular, I am much thankful to Daniel Amyot and Rossana Andrade for providing many research ideas of this thesis. I also give special thanks to Jacques Sincennes for his technical support and his patient help with the use of the LOTOS language and toolkit. Other friends, Masahide Nakamura and Ralph Boland helps me a lot on my proof reading and I would like to express my sincere thankfulness to them.

I should thank our industrial collaborators, John Visser and Jim Hodges, for initiating the idea of this thesis and for providing technical information and assistance. Nortel-Networks, Communication and Information Technology Ontario and the Natural Science and Engineering Research Council must be thanked for their financial support.

At last, I have deep gratitude towards my family, especially my father and aunt Ethel Yik. They give me love and support as much as they can, and I believe that this thesis is the best gift that I can give to them.

Abstract

Over the past few years, the subject of Wireless Intelligent Network (WIN) has captured the interest of the North American telecommunications community. The objective of WIN is to integrate Intelligent Network (IN) concepts into the IS-41 architecture. The introduction of a Service Control Point (SCP) in the Interim Standard IS-41 architecture enables independent specification of services. By using the IN architecture, call processing intelligence and feature functionality is separated from network switches. Subscribers can take advantage of such IN services as call name presentation (CNAP) in a wireless environment.

In the first part of the thesis, we apply a scenario analysis technique to capture the requirements of the CNAP system. The analysis includes building a design model composed of User Case Maps (UCMs) and Message Sequence Charts (MSCs). This technique helps in specifying the designed system at a high level, brings us one step closer towards the specification of CNAP in LOTOS, and contributes a method for generating test cases for further validation of the specification.

In the second part of the thesis, we validate the design of the CNAP feature by constructing a LOTOS specification for it and validating the specification. The formal specification of the CNAP service is based on the structure of Network Entities (NEs) that are defined in the Network Reference Model (NRM). The specification emphasizes the establishment of call connection and processing of services in a wireless environment.

It is concluded that UCMs, being an intuitive notation that can be used loosely and at several levels of detail, is appropriate for the requirement and service description in the development of standard. LOTOS, being fairly precise but being able to be used abstractly, is appropriate for prototyping and validating the call procedures and services of WIN.

ABBREVIATIONS

A:

AC—Authentication Center

ADT—Abstract Data Type

B:

BS—Base Station

C:

CNAP—Calling Name Presentation

CCF—Call Control Function

CNA—Calling Name Information

CCS—Calculus of Communicating System

CSP—Communicating Sequential Processes

D:

DFP—Distributed Functional Plane

F:

FDT—Formal Description Technique

G:

GSM—Global System for Mobile Communication

GPRS—General Packet Radio Services

H:

HLR—Home Location Register

I:

ICS—Incoming Call Screening

IN—Intelligent Network

IS—Interim Standard

ISDN—Integrated Services Digital Network

IP—Intelligent Peripheral

L:

LOTOS-- Language Of Temporal Logic Specification

LRF—Location Registration Function

M:

MS—Mobile Station
MSC-- Message Sequence Chart
MSC—Mobile Switching Center
MIN—Mobile Station Identification Number
MDN—Mobile Station Directory Number
MACF—Mobile Station Access Control Function

N:

NRM—Network Reference Model

P:

PCS—Personal Communications Services
POTS—Plain Old Telephone System
PSTN—Public Switch Telephone Network

R:

RND—Redirecting Name Display

S:

SMSC—Short Message Service Center
SN—Service Node
SS7—Signaling System 7
SCP—Service Control Point
SCF—Service Control Function
SIM—Service Infrastructure Model
SSF—Service Switch Function

T:

TIA—Telecommunications Industry Association
TLDN—Temporary Local Directory Number

U:

UCM---Use Case Map

V:

VLR—Visitor Location Register
VCS—Voice Call Screening

W:

WIN—Wireless Intelligent Network

INDEX

Chapter 1 Introduction	3
1.1 Background and Motivation	3
1.2 Objective and Approach	4
1.2.1 Objective	4
1.2.2 Approach.....	5
1.3 Organization of the thesis	6
1.4 Related Work	8
1.4.1 Formal Specification of Telephony Features in a distributed environment... 8	
1.4.2 Scenario Analysis.....	9
Chapter 2 Wireless Intelligent Network	11
2.1 Background	11
2.2 WIN Architecture	12
2.2.1 Network Reference Model (NRM) of WIN	13
2.2.2 Distributed Function Plane (DFP) of WIN	15
2.3 WIN Services	16
2.3.1 CNAP Service Subscription.....	17
2.3.2 CNAP Architecture	20
2.3.3 Strategy for Deploying the CNAP service	22
2.4 Conclusion:	24
Chapter 3 Selected Techniques	25
3.1 Introduction	25
3.2 The stages in the standardization Process	25
3.3 Techniques applicable to these stages	26
3.3.1 An Overview of Use Case Map (UCM).....	26
3.3.2 An Overview Of Message Sequence Charts (MSC)	30
3.3.3 Overview of the LOTOS Language	32
3.3.3.1 LOTOS Data Part	33
3.3.3.2 LOTOS Control Part	35
3.3.3.3 Specification Styles of LOTOS.....	38
3.4 Evaluation	40
3.5 Recommendations	42
Chapter 4 Scenario Oriented Analysis for CNAP service in WIN	43
4.1 Introduction	43

4.2	Service Description Stage of CNAP	45
4.3	Message Sequence Information Stage of CNAP	51
4.3.1	Component Types	52
4.3.2	Bound UCMs.....	54
4.3.3	MSCs of CNAP service.....	57
4.4	Conclusion	61
Chapter 5 LOTOS Specification of CNAP		62
5.1	Introduction	62
5.2	Data Type of the specification	63
5.3	Control Part of the specification	66
5.3.1	Process MS_Subscribers	68
5.3.1.1	Process MS.....	70
5.3.1.2	Process Originator	71
5.3.1.3	Process Terminator.....	72
5.3.2	Process WIN_CNAP.....	74
5.3.2.1	Process MSCstack.....	74
5.3.2.2	Process HLRstack	79
5.3.2.3	Process VLRstack	80
5.3.2.4	Process SCPstack	80
5.3.3	Process WIN_Database.....	82
5.4	Conclusion	84
Chapter 6 Validation of CNAP		85
6.1	Overview of the validation method	85
6.2	Tool support	85
6.3	Simulation, Testing and New Scenario Generation.....	87
6.3.1	Simulation	87
6.3.2	Specification Testing.....	89
6.3.2.1	Derivation of Test Cases from Unbound UCM	91
6.3.2.2	Structure Coverage.....	97
6.3.3	Automatic Graphic Scenario Generation	104
6.4	Discovery of a number of ambiguities and inconsistencies in the Draft Standard .	106
6.4.1	Scenario 8.1.3.....	106
6.4.2	scenario 8.1.5.....	108
6.5	Conclusion	108
Chapter 7 Contributions and Future Work		110
7.1	Contribution of the thesis.....	110
7.2	Future Work	111

Chapter 1 Introduction

1.1 Background and Motivation

The telecommunication industry needs high quality standards in many areas. The specific area targeted in this thesis is the description of mobile telephony features. Currently, features are usually documented using informal descriptions, tables and Message Sequence Charts (MSC). These descriptions evolve dynamically, and their readability and validation become difficult to manage. The following problems are detected in current standard documentation:

- Abstraction level is too low with respect to the available knowledge. Irrelevant details obscure the main idea behind a feature. A focus on tables and MSCs is necessary for detailed design, but cannot give us an abstract scenario view for defining functionalities at the initial design stages.
- Information that is relevant to a feature is often distributed over several parts of the documentation. Several levels of detail (abstraction) may be mixed in a single description.
- There are possible ambiguities, inconsistencies, or interactions inside or between feature descriptions, or between levels of abstraction. It is difficult to detect them with conventional inspection methods.

The TR45.2 WIN task group is the Telecommunications Industry Association (TIA) Standards Committee for standardizing Wireless Intelligent Network (WIN) documents.

Within the TR45.2 WIN task group (and elsewhere), these difficulties are encountered in the development of readable, usable, maintainable, and evolvable standards. Few participants in the TR45.2 WIN group are willing to use techniques other than ad hoc ones, and significant effort is required to perform validation and verification of specifications, that can lead to the discovery of inconsistencies and omission.

Formal Description Techniques (FDTs) have been well known in the standards world. They are increasingly recognized as tools that provide validation and verification methodologies for improving standard quality as size and complexity of software systems grow. LOTOS, for example, shows notable flexibility for prototyping system behavior and for providing precise descriptions of telecom standards. A survey of this and related work will be given in the section 1.4.1. A detailed discussion of LOTOS syntax and validation methods will be introduced in Chapter 3.

Use Case Maps (UCMs), on the contrary, are quite informal. They suggest a description of behavior with architectural concepts, outlining the system's structural properties. They define component entities and imply relationships among components. A lot research has been done towards formalizing UCMs. We will discuss this in section 1.4.2. An overview of the UCM technique will be given in Chapter 3.

Above all, an effort will be required to try a new modern approach to the specification and validation of the WIN standards. Our thesis intends to be a contribution in this direction.

1.2 Objective and Approach

1.2.1 Objective

We aim to contribute to the development of a new standard in the domain of wireless technologies and provide a suitable approach for analyzing, formalizing, and validating a WIN service. Our goal is to understand the strengths and weaknesses of the selected

techniques, to demonstrate relevant practices for describing the service requirements, functions, and protocols, to improve the human understanding and the quality of standard. In contrast to earlier work on approaches to specification and validation of telecommunication systems in LOTOS, the present study attempts to combine the intuitive descriptive power of UCMs with the analytic power of LOTOS and its tools. As an example, we chose to apply this approach to the CNAP service in the WIN standard.

1.2.2 Approach

This thesis defines and applies a design approach for structuring a WIN development cycle according to the techniques that have been proposed by our research group. This approach allows us to further refine and adapt our techniques to industrial realities, while providing a coherent and validated view of the feature we have selected. It consists of two parts.

- In the first part, we build a scenario model through analysis of informal requirements. The scenario model consists of a set of unbound UCMs, bound UCMs, and MSCs. Unbound UCM scenarios provide a high level CNAP service description without commitment to the underlying architecture. This description can be used to capture the requirements from the perspective of a service subscriber. Bound UCMs, instead, are dependent on the underlying architecture. They include the definition of component types and the interfaces between different components. More detailed information is complemented by the use of MSCs. MSCs provide rigid message exchange information, as appended details, for interactions between components along the paths of UCMs.
- In the second part, based on the set of collected scenarios, we generate an executable LOTOS specification. Using this specification prototype, we can apply validation through three steps. Through these three steps, some symptoms of possible design errors are found, and are fed back to the standardization committee for their consideration.

- 1) In the first step, we make use of the LOLA toolkit to simulate the system step by step.
- 2) In the second step, we synchronize our validation suite generated from requirements with the LOTOS specification. This leads to test results, such as Must pass, May pass, and Reject. Through this process, many system traces with internal steps are derived from the specification. We can also measure the structure coverage for the generated test cases by using probes.
- 3) These generated system traces are made readable in a graphical format by using the third step. A perl script is used to translate the formal symbolic trace format into a MSC PR format which can be accepted by the SDL MSC editor.

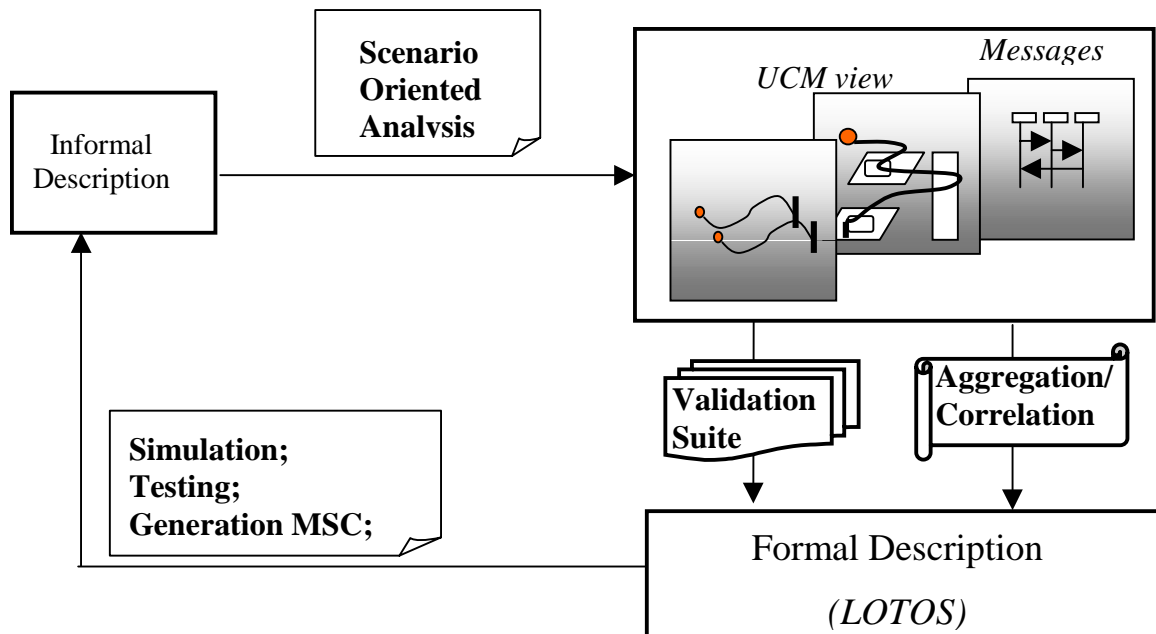


Figure 1-1 Scenario-Oriented Approach

1.3 Organization of the thesis

This thesis is organized as follows:

Chapter 2: Wireless Intelligent Network

General concepts of the Wireless Intelligent Network are reviewed by briefly describing the architecture of the network model. The CNAP service, one of the WIN services, will be used as an example in this thesis.

Chapter 3: Chosen Techniques

This chapter contains an overview of three candidate techniques – UCM, LOTOS and MSC. By the end, we will have an evaluation and assessment of each of them with the consideration of application stages.

Chapter 4: Scenario Oriented Analysis for CNAP service in WIN

A scenario-oriented analysis process is described for specifying the evolution from the informal requirement towards a set of system scenarios at different stages. We present the approach by analyzing the CNAP service in WIN.

Chapter 5: LOTOS Specification Of CNAP

Based on the scenarios given in chapter 4, we aggregate and correlate the architecture and behavior information to derive a formal LOTOS specification. In this chapter, we present a formal LOTOS specification of the CNAP service. The purpose of generating a LOTOS specification is to make use of various validation methods provided by this language and to demonstrate a method of obtaining precisely defined protocols.

Chapter 6: Validation Of CNAP

Our validation focuses on the simulation, testing, and graphical MSC Generation. As a result, some inconsistencies and ambiguities existing in the CNAP requirement are found. We have developed a perl script to facilitate the readability of our scenarios for validation purposes.

Chapter 7: Conclusion and Future Work

Conclusions and possible areas for future research are present in this Chapter.

1.4 Related Work

Specifying and validating mobile telephony features is a very new area of research and very little, if any, literature exists. Below, we review some related literature. Other related references will be included later in the appropriate sections.

1.4.1 Formal Specification of Telephony Features in a distributed environment

The early work of specifying a simple telephone system was based on the concept of Plain Old Telephone System (POTS). [Bo91] [BoLo93] specified telephony systems in the wire-line world that provided users with a number of telephony features, which included Call Forward, Ring Again, Call Transfer/Three-Way Calling, etc.. This research mainly concentrated on showing the suitability of LOTOS for specifying such systems and on showing that the resulting telephone specifications can be validated by the use of an interpreter (ISLA). The Feature Interaction problem was also addressed in [BoLo93]

With the infrastructure provided by POTS, the task of introducing new services is very costly. intelligent network (IN) concept somehow solves this problem. Beginning in the early 1980s, the IN was applied to the development of new services in wire-line telephone networks. The IN concepts allow rapid introduction and independent implementation of new features. Various methodologies for presenting IN system were published since then. [BoZu92] is a paper that describes the IN Global Functional Plane in LOTOS. [DaNa93][Cheng94] discussed and designed methodologies for the specification and analysis of IN services in LOTOS. Recent work related to IN features is [JaLo98]. This research concentrated on showing the LOTOS specification of the IN infrastructure, where a formal specification using the Service Independent building Blocks was described, and three telephony features were included within the defined IN model.

IN concepts can also apply to the wireless telephony world. However, it should be noted that wireless standardization has a different focus from the conventional IN. Whereas the focus of IN standardization work was on value-added services in the fixed networks, the work on wireless networks was to assist service providers and end users with the ability to obtain telecommunications services regardless of their location and to be in motion while continually accessing telecommunications services.

The Telecommunications Industry Association (TIA) Standards Committee TR45.2 is developing Wireless Intelligent Network (WIN) [WIN98]. The phase I of this standard draft has been standardized in Dec 1998. The charter of this committee is to drive IN capabilities into wireless networks which are based on interim standard (IS)-41. [LaRa95] illustrates the idea of integration of IN Services into future wireless network - Global System for Mobile Communications (GSM). The LOTOS specification of GSM was described in [TuLo98], but there is no IN feature included in the GSM specification.

1.4.2 Scenario Analysis

LOTOS is a mathematically based specification language. It provides a precise specification of system behavior at the desired level of abstraction. The gap between the early informal requirements and the formal LOTOS-based processes can be filled by scenario descriptions. Buhr's Use Case Maps (UCMs) methodology [Buhr96] starts system development with the gathering and analysis of high level scenarios. Using UCMs as a bridge between requirements and LOTOS specification is a research area where a number of papers were published in recent years.

In [ABBL95], a design methodology which allies the graphical expressiveness of the time-thread notation (Use Case Maps) with the analytical power of the LOTOS language and its associated tools is presented. It demonstrates in detail how to manually generate LOTOS specifications from UCM. A simple telephone system is used as an example. [ALF97] presents an approach where informal requirements are described with UCMs,

and formal specifications and test cases are written in LOTOS. The approach is presented by using an example: a GPRS group call service of the mobile data system. [AmAn99] uses UCMs as part of a new technique for designing WIN scenarios. This paper shows the suitability of applying UCMs before MSCs and LOTOS in order to achieve better and more complete descriptions of telecommunication standards. This paper focuses on illustrating the Incoming Call Screening (ICS) service as one example chosen from WIN. [ALBG99] applies the idea of integrating UCMs and LOTOS to an interesting telephony field: “Feature Interaction”. It presents this integrated approach to capture and validate several telephony features defined from the First Feature Interaction Contest [GBGO98]. It also discussed the result of this experiment, as well as strengths and weaknesses of this approach.

Chapter 2 Wireless Intelligent Network

2.1 Background

In this chapter, we give an overview of the Wireless Intelligent Network (WIN) in terms of the architecture and enhanced services. The content of this chapter is based on the standardizing document IS-41 and the WIN draft of Telecommunications Industry Association (TIA) Standards Committee TR45.2.

Before 1980s, telephone service was switch-based, which means that all the data and logic processing required by the service were located within the local node. Beginning in the early 1980s, the concept of Intelligent Network (IN) capabilities introduced a new method for developing new services in the wire-line industry. This means that the service control is centralized in some specific non-switching nodes. These nodes are known as Service Control Points (SCPs). IN service functional capabilities support creation and execution of service logic programs. Those functional capabilities reside outside of the switching equipment but are centralized in SCPs. SCPs work collaboratively with the switching equipment based on a common definition of call model and protocols. By introducing IN capabilities in the wire-line world, we facilitate and accelerate the implementation and provision of telephony services in a cost-effective manner.

WIN supports the usage of intelligent network concepts and focuses on the integration of IN capabilities with mobility. In wireless networks, many of the call activities are not just the actual phone call. They are associated with movement. Customers can roam out of their local calling area or out of their service provider's area. They may want to be able to use the same services as they use in their home area. They also may want these services to work in the same way. Mobility dictates a need for technologies or standards that make it possible for different networks to talk to each other.

IS-41 is a well-established standard for mobile telephony in North America. The Network Reference Model (NRM) is provided by IS-41 as a wireless network architecture to define network entities and their associated interface reference points. The WIN standard is based on IS-41. Basing WIN standards on this protocol enables a graceful evolution from IS-41 to an IN without making the current network infrastructure obsolete. On the basis of NRM, the WIN standard adds three new network entities to enhance IN feature. The three network entities are the Service Control Point (SCP), the Service Node (SN) and the Intelligent Peripheral (IP). The SCP is very important since the service control is moved away from the MSC and up to a higher element in the network, usually the SCP.

A new distributed functional plane (DFP) and three advanced network services are described in the current WIN standard. The DFP provides a different view of the network in terms of functional entities (FEs) than in terms of network entities (NEs). Each FE performs distinct actions in the network. A grouping of actions across one or more FE provides the required WIN service execution. In the following sections, we have a more detailed discussion on WIN architecture and WIN services.

2.2 WIN Architecture

The following discussion is based on [WINT]. We introduce first the Network Reference Model (NRM) and then the Distributed Functional Plane (DFP).

2.2.1 Network Reference Model (NRM) of WIN

The network entities of the NRM are shown in Figure 2-1. They include the authentication center (AC), the base station (BS), the equipment identity register (EIR), the home location register (HLR), the Intelligent peripheral (IP), the message center (MC), the mobile station (MS), the mobile switching center (MSC), the short message entity (SME), the service node (SN), the visitor location register (VLR) and the service control point (SCP). In Figure 2-1, the network entities that are mostly involved with our specification of CNAP service are marked in gray. These entities will be processes in our resource-oriented specification.

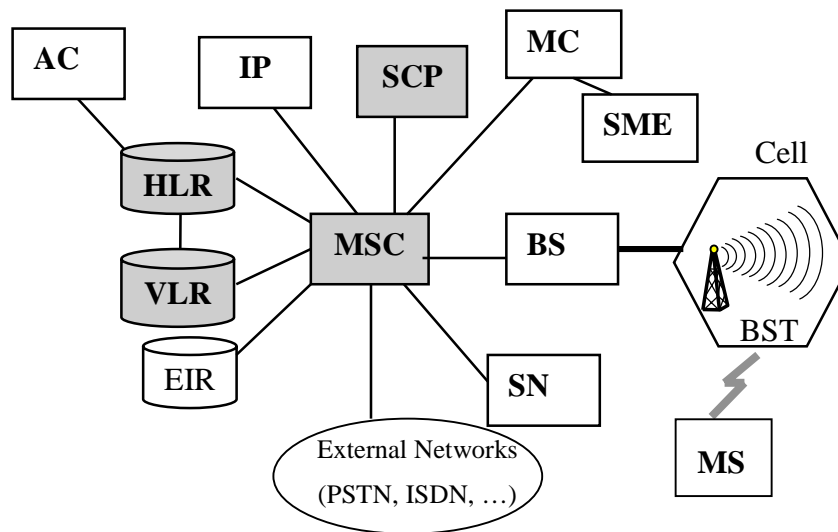


Figure 2-1 WIN Network Reference Model

Mobile Station (MS)

An MS is a terminal equipment, which the subscriber uses to access and obtain services from the network. Different types of numbers are associated with an MS.

Mobile Station Directory Numbers (MDNs) are assigned to mobile stations as dialable, public numbers conforming to the international numbering plan. MDNs are used to support functions such as:

- Dialing/addressing for call origination,

- Providing location and/or service information to the users.

Mobile Station Identification Numbers (MINs) are assigned to mobile stations internally by the mobile network. MINs are used in the mobile network as 'roaming numbers' provided by the visiting network to the home network for call setup to the roaming subscriber. These roaming numbers are not dialed by the user and are not assigned to the mobile stations as directory numbers. They are used to support such mobility management functions as:

- Identification of the MS on the radio control path for location update/registration
- Identification of the MS for all signaling on the radio control path (e.g. , for paging)
- Identification of the MS for updating or retrieving subscriber profiles, which may contain such items as subscribed features and triggers to query service logic for enhanced services.

Home Location Register (HLR)

An HLR contains a database that stores information related to a registered set of subscribers. The current subscriber's service subscription status is maintained in the HLR. Some data are permanent, while others are temporary and thus change from call to call. The HLR also manages the location information in the network: for instance, it has to tell the old MSC/VLR to erase a subscriber record when this subscriber is registered under a new MSC/VLR.

Visitor Location Register (VLR)

A VLR may be in charge of one or several MSC location areas. The VLR constitutes the database that supports the MSC in the storage and retrieval of subscribers present in its area. This is why we often talk of MSC/VLR as one joint entity. When a MS enters the MSC area borders, its signals its arrival to the MSC for storing its identity in the VLR. Meanwhile, the information necessary to manage the MS is transferred from HLR to VLR so that they can be easily retrieved if required. The subscriber's current VLR address, stored at the HLR, is accordingly updated. The data stored in the VLR are more or less the same as in HLR. This provides the information necessary to complete

calls to roaming mobiles. Nevertheless the data are present in the VLR only as long as the MS is registered in the area related to that VLR.

Service Control Point (SCP)

The SCP acts as a centralized element in the network that provides service control functionality to WIN subscribers. SCP provides the mechanism for new services independent of their switching system. High-level services can be moved away from the MSC and be controlled at SCP. This mechanism simplifies new service development and is cost effective since the MSC becomes more efficient and does not waste cycles processing new services.

2.2.2 Distributed Function Plane (DFP) of WIN

The Distributed Functional Plane (DFP), as described for IN-structured network [Q1204], includes a set of functional entities (FEs). Each FE can perform a variety of functional actions and can cooperate with others. FEs defined in this plane are generally classified into three types: call control related functions; service control related functions; management related functions. The scope of the DFP architecture for WIN is based on the IN DFP but is driven by the requirements of desired wireless services. Therefore, some FEs related to mobility control and radio communication control are included. The SCF, CCF/SSF, MACF, LRF_V, and LRF_H are the functions mentioned in our thesis for describing CNAP services.

- Call Control Function (CCF) provides call and service processing and control. It is a function that handles all normal calls by providing the process and the control of call/connection between network subscribers. It also provides the IN service access.
- Location Registration Functions (LRF_V LRF_H) provides the service logic and service data function to manage the mobility aspects for wireless users;
- Mobile Station Access Control Function (MACF) stores subscriber data and dynamically associates system resources with a particular call.

- Service Control Function (SCF) executes service logic. It provides capabilities to influence call processing by requesting the SSF/CCF to perform specified actions.
- Service Switching Function (SSF) is associated with CCF and provides the set of functions required for interaction between the CCF and a service control function (SCF) by managing signaling between them;

2.3 WIN Services

The WIN provides seamless terminal services, personal mobility services and advanced network services in the mobile environment by using intelligent network capabilities [WIN98].

Terminal mobility refers to the ability of a mobile terminal to access telecommunication services from any location, and the capability of the network to locate and identify the mobile terminal as it moves. A set of these services will be associated with each mobile terminal based on the capabilities of the terminal and terminal subscription selections, irrespective of the terminal user. The prerequisite for the user to achieve terminal mobility is to carry a terminal and to be within the radio coverage of the wireless network.

Personal mobility is the ability of end users to originate and receive calls, and access subscribed telecommunication services on any terminal, in any location, and the ability of the network to identify end users as they move. A set of these services will be based on personal needs or business entity needs irrespective of terminals or networks. Personal mobility does not require customer to carry a terminal. The customer may utilize a variety of mobile and fixed terminals at different locations. But he/she needs to have a personal number.

Advanced Network Service mobility refers to the network capability to provide subscribed services at the terminal or location designed by the user. Service mobility means that the services that users can invoke at a designed terminal depend on the capability of the terminal and the serving network. If a user has registered to an advanced network service, this service supposes to work when the user roams to another network.

This however is based on network capabilities. For example if the user roams to a network which supports ANSI-41 but is not WIN compliant, the WIN services are not available.

- **Terminal mobility services:** services based on the terminal capability irrespective of the terminal user. Terminal mobility is provided between wireless networks.
- **Personal mobility service:** services based on personal needs or business needs irrespective of terminals or networks. Personal mobility is provided between wireless and wire-line network.
- **Advanced Network services:** services based on the network and terminal capability. Seamless service mobility is provided between wireless and wire-line networks. The initial WIN standard describes three advanced network services such as Calling Name Presentation Service (CNAP), Voice Controlled Service (VCS) and Incoming Call Screening Service (ICS). Future WIN standards will add additional WIN services. In this thesis, the calling-name presentation (CNAP) service will be an example for applying a scenario oriented-analysis specification and validation approach. This feature's architecture and the call process will be described in the following sections.

2.3.1 CNAP Service Subscription

Before we start the informal description of CNAP service subscription, we need to define some terminology used in the requirements.

Registration

The procedure by which an MS becomes listed as being present in the service area of an MSC.

Originating MSC¹

Originating MSC represents the MSC which initiates the call delivery procedures of the calling party.

¹ Note that the IS-41 standard uses the term "Originating MSC" to denote the MSC where the call starts on the call terminator side. This should perhaps be called as "Terminating Anchor" MSC.

Serving MSC

Serving MSC represents the MSC that serves the called party at one of its cell sites within its coverage area.

Available

The MS can accept a call delivery (i.e., the MS is in a known location and it is in a state that is able to accept call deliveries). The availability of an MS to accept a call delivery is maintained only by the MSC.

Unavailable

The MS cannot accept a call delivery (i.e., the MS is in an unknown location or is in a state unable to accept call deliveries). The availability of an MS to accept a call delivery is maintained only by the MSC.

Active

The MS is available for call delivery. The MSC, the VLR and the HLR maintain this state. For example, if an MS registers, the serving MSC will apply an operation to the serving VLR to indicate that an MS is active, and then the serving VLR will notify the HLR of the registration of the MS.

Inactive

The MS is unavailable for call delivery. The MS may not be registered. Or the MS may be registered, but it is out of radio contact or is intentionally inaccessible for periods of time. The MSC, the VLR and the HLR maintain this state.

Restricted

The name information of the originating MS is not allowed to display on the terminating party due, for example, to reasons of protection.

The requirement specification of the CNAP service is based on [WIN98]. Calling Name Presentation (CNAP) provides the name identification of the calling party (e.g. personal

name, company name, “restricted”, “not available”) to the called subscriber. Typically, this will be shown in the display window of the MS. The Calling Name Information (CNA) may be provided to the terminating network from the originating network or it may be derived from the Calling Number Information (CNI) which is generally provided to the terminating network from the originating network. CNAP does not impact a subscriber’s ability to originate calls or to receive calls. Redirecting Name Delivery (RND) is a CNAP subscription option. When CNAP is active and a call has been redirected, RND provides the name of the last redirecting party as well as the name identification of the original calling party to the called subscriber. Table 2-1 shows the subscription activation options for the CNAP and RND service. Clients who make the options are those who have already subscribed to the CNAP and RND service.

Subscription Options	Activation	Activation Status
CNAP Activation		Permanent. CNAP is active while authorized
		Demand. The subscriber is authorized to control the activation and de-activation of CNAP.
RND Activation		Permanent. RND is active while authorized.
		Demand. The subscriber is authorized to control the activation and de-activation of RND

Table 2-1 CNAP Service Subscription

Normal Operation with Successful Outcome

When the CNAP service is invoked, the terminating network shall send the calling name information to the terminal during alerting on incoming calls. If the calling name is not restricted, the terminal shall display the calling name information. The stored calling name shall preserve the presentation restriction indications (e.g. the user may restrict

delivery of her name to some users). In the event of redirection when CNAP and RND are both active, two names will be presented to the subscriber:

- The name associated with the original calling party
- The name associated with the last redirecting number

Exception Procedure or Unsuccessful Outcome

If the calling name is not available, the terminating network shall provide an indication of this fact to the mobile station. If the calling name is available and the presentation of calling name is restricted, the serving system shall provide an indication to the mobile station that the calling name is restricted.

If the redirecting name is not available, the terminating network shall provide an indication to the mobile station that the redirecting name is not available. If the redirection name is available and the presentation of the redirecting name is restricted, the serving system shall provide an indication to the mobile station that the redirecting name is restricted.

2.3.2 CNAP Architecture

In this section, we introduce a subset of WIN architecture, which can be modeled as a system environment for CNAP. The model, shown in

Figure 2-2, is composed by multiple network entities and their associated function entities. The model intends to provide a level of abstraction that can facilitate the specification of intersystem processing of CNAP service. WIN standard uses the terminology “reference point” for interface between network entities.

- Reference Point AUm is the MS to MSC interface
- Reference Point B is the MSC to VLR interface
- Reference Point C is the MSC to HLR interface
- Reference Point D is the VLR to HLR interface
- Reference Point E is the MSC to MSC interface
- Reference Point T1 is the MSC to SCP interface
- Reference Point T2 is the HLR to SCP interface

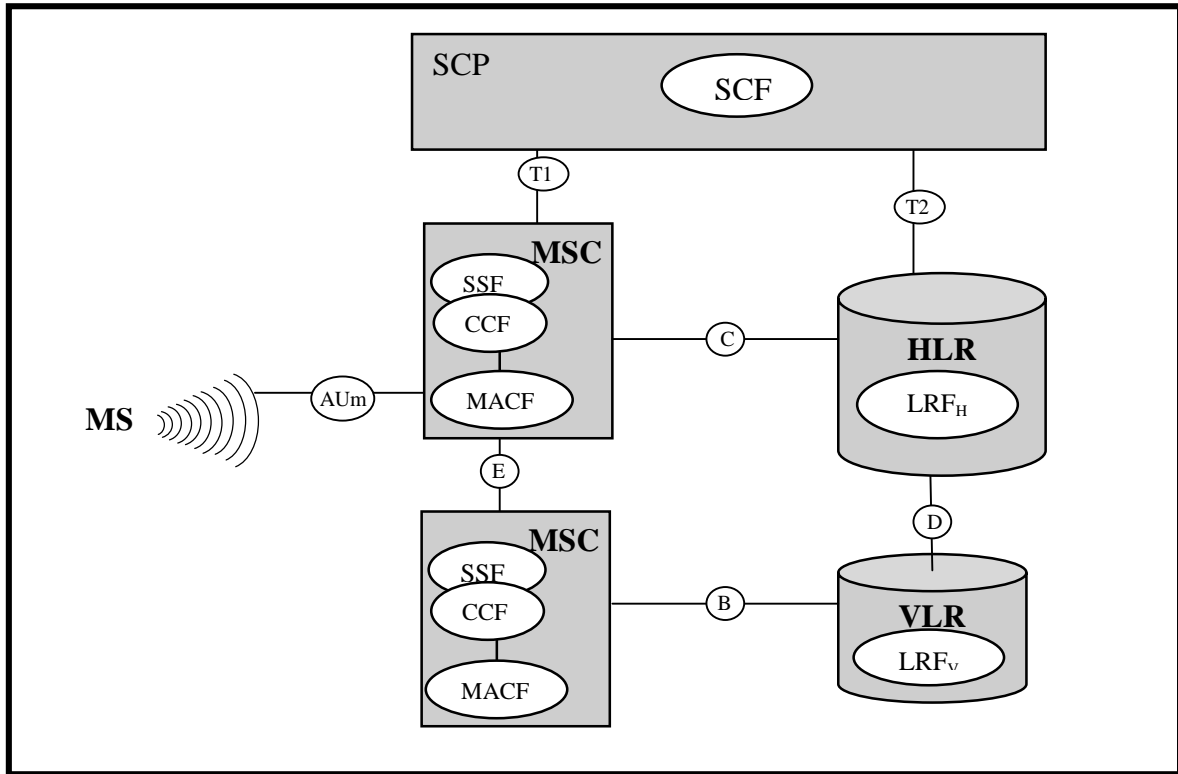


Figure 2-2 WIN CNAP Architecture Model

- Within the SCP, the SCF contains the logic and processing capability required for handling the WIN CNAP service.
- Within the HLR, LRF_H maintains the subscriber profile and interacts with LRF_v
- Within the MSC, there is a close coupling of the SSF and the CCF functional entities. The SSF/CCF provides the call setup and service invocation. The MACF provides the paging and interacts with LRF_v.

- Within the VLR, LRFv stores the subscriber profile, interacts with LRFH and the MACF functional entities and provides a routing address.

2.3.3 Strategy for Deploying the CNAP service

The WIN standards committee defined two methods for delivering the Calling Name Presentation service. Either method provides the correct environment for IN service deployment. Figure 2-3 demonstrates a SCP-based service deployment strategy by showing how an enhanced service can be delivered by using a service query from the MSC. Figure 2-4 depicts the other optional HLR-based service deployment by showing how to trigger a service query from the HLR. Similar alternative methods were proposed for other WIN services.

SCP-based Service Deployment Strategy:

The SCP-based service deployment strategy is the preferred and recommend method for offering new WIN services. The basic idea is that the mobile switch queries a number-to-name database service on an SCP using the calling party's number, which was delivered using a signaling interface such as Public Switch Telephone Network (PSTN). In Figure 2-3, an originating MS1 sends an INCOMING CALL indication that it wishes to originate a call. The indication is delivered to its Originating MSC. Dialed digits are included in the call origination indication. On determining that the call is to a mobile subscriber, the Originating MSC assigns a radio channel to MS1, Meanwhile, the Originating MSC sends a LOCATION REQ to the HLR registered by the called subscriber. The HLR determines the current serving MSC for the called MS and, if necessary, it sends a ROUTE REQ to the serving system for obtaining a Temporary Local Directory Number (TLDN). TLDN is a network address temporarily assigned for call setup. The MACF function entity inside the Serving MSC receives the ROUTE REQ and starts handling the request by assigning a TLDN to the called MS and returns it to the HLR. Then, the HLR returns the routing information to the Originating MSC. The Originating MSC can use the TLDN to route the call over the PSTN to the Serving MSC. After the call link is setup between the Serving MSC and the Originating MSC, the

serving MSC pages the called MS2. If MS2 responds, a voice channel is assigned to MS2. When the call is delivered to the serving MSC, the caller's number is also delivered. Noting that the called customer has subscribed to calling name presentation service, the serving MSC queries a number-to-name service from the SCP. SCP will return proper display text. As a result, MS2 is alerted with the calling name information and an alerting indication is provided to the Originating MSC. If MS2 answers the call, the peer communication starts.

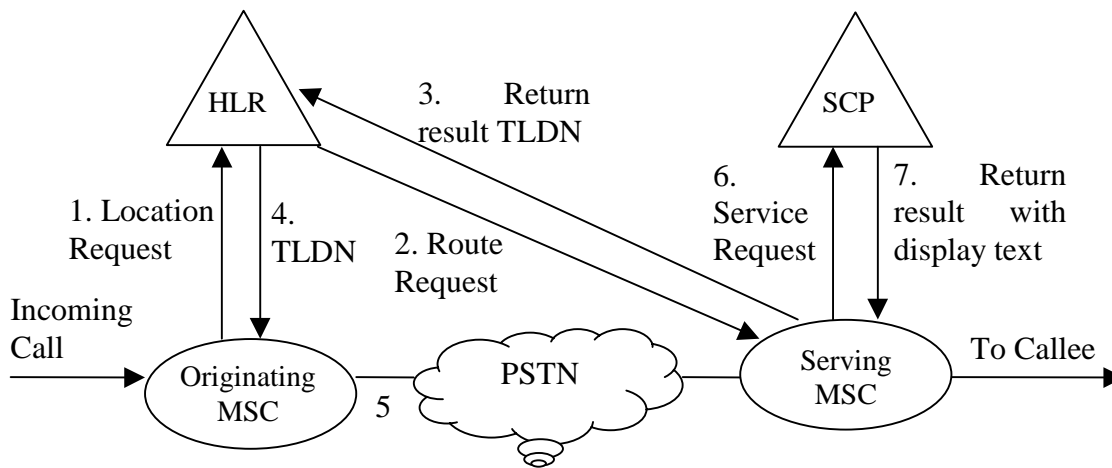


Figure 2-3 SCP-based SIM CNAP

HLR-based Service Deployment Strategy:

We can use the HLR-based service deployment strategy as an alternative way to provide the CNAP service. This method is also defined by the WIN standard committee. The reason to introduce this strategy is that, currently, MSCs from multiple vendors are often implemented differently, causing situations where services do not perform in the same way. Considering this fact, the HLR-based SIM method was introduced by the standards group as a short-term solution to allow multi-MSC networks to provide ubiquitous services. In the future, the SCP based SIM strategy will be more readily available when all MSCs in a network can provide the standardized triggers and capabilities. Figure 2-4 shows the service deployment process. The originating MS1 sends an INCOMING CALL indication that it wishes to originate a call. Dialed digits are included in the call origination indication. The call is delivered to the Originating MSC.

On determining that the call is to a mobile subscriber, the Originating MSC assigns a radio channel to MS1. Meanwhile, the Originating MSC sends a LOCATION REQ to the HLR. Before the HLR sends a ROUTE REQ to the serving system for obtaining a TLDN, the HLR queries the number-to-name database on SCP. Upon receipt of the display text, the HLR may send a ROUTE REQ with this text to the Serving MSC to obtain a TLDN. The serving MSC stores the display text until the call comes in from the PSTN. When the serving MSC delivers the call to the mobile station, the text is displayed during alerting.

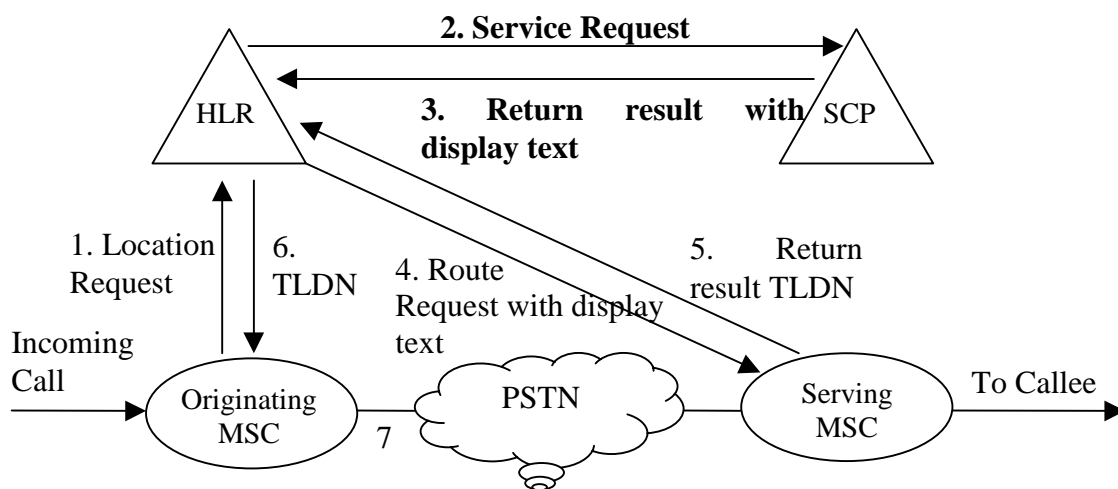


Figure 2-4 HLR-based SIM CNAP

2.4 Conclusion:

The WIN standard describes two methods for offering WIN services, as described above. According to the standards groups, the preferred and recommended method for offering new WIN services is that the service should be launched from the MSC to the SCP. In this case, the transaction does not have to take place if the mobile is busy or cannot receive the call. Therefore, the network can operate more efficiently. Our thesis will choose the recommended method for deploying the WIN CNAP service.

Chapter 3 Selected Techniques

3.1 Introduction

This chapter introduces the readers to the description techniques that are used in this thesis. In particular, we are going to see that the development of a telecommunication standard goes through at least three stages and that different techniques can be used to help the designer through these stages. The bulk of this chapter will consist of discussion of the techniques and their application.

3.2 The stages in the standardization Process

Following the practice in several standard groups, ITU-T divides the development of telecommunication standard into three stages: service description (stage 1), message sequence information (stage 2), and protocol and procedure specification (stage 3). The three-stage methodology was initially developed to describe Integrated Services Digital

Networks (ISDN) services and is carried out for each service in telecommunication standard.

- In stage one, a description of a service should be given from the perspective of users. This stage does not try to describe how the system works, but instead, what functionality it provides.
- In stage two, the capabilities and processes within the network that are required to provide the service are designed. The output of this stage is the functional decomposition of the network components as well as the Information Flows (IFs) to support the service.
- The final stage produces the protocol specification.

3.3 Techniques applicable to these stages

Based on the ITU-T three stage methodology, we recommend different techniques for each stage. These techniques aid human understanding and help to produce better-quality standards. The techniques discussed here are:

- Use Case Maps (UCMs)
- Message Sequence Charts (MSCs)
- Language Of Temporal Ordering Specification (LOTOS)

3.3.1 An Overview of Use Case Map (UCM)

UCMs [BoBu97] are a visual notation developed at Carleton University. UCMs used to be called time threads since executing responsibilities and traversing path segments correspond to event sequence in time. More detail about time threads can be found in [ABBL95]. The essential idea of UCMs is to describe scenario paths in terms of causal relationships between responsibilities. “Causal relationship” between components can be interpreted in many ways, depending on the component structure. More than one MSC may be derived for a single UCM. In Figure 3-1, the paths in the UCM show the causal sequence *ab* in an abstract manner. Two possible implementations of this UCM are shown in the form of two MSCs.

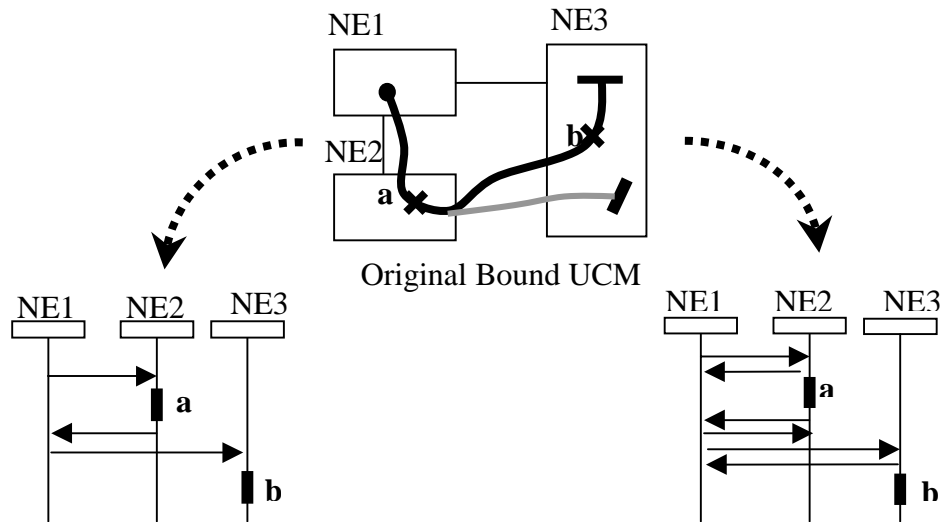


Figure 3-1 Causal Sequence of a UCM

Depending on whether they bind components along the path or not, UCMs can be called bound UCMs or unbound UCMs. Unbounded UCMs are defined in terms of paths without showing the components in which the responsibilities are performed. They are used at early design stages, when it has not yet been decided what components may exist in the system. Figure 3-2 shows a simple unbound UCM.

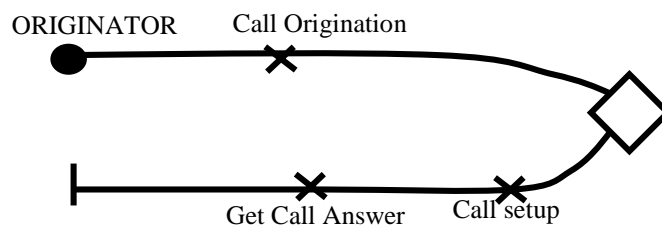


Figure 3-2 Basic Notation of UCM



Path. A path may have any shape. It represents a causally related sequence of responsibilities



Waiting place. A filled circle represents a place to wait for events from other paths. A starting point is a waiting place for a stimulus to start the path.



Stub. When maps become too complex to be represented as one single UCM, a mechanism for defining and structuring sub-maps becomes necessary. A top-level UCM, referred to as a *root map*, can include containers (called stubs) for sub-maps (called plug-ins).



Bar. A bar ends a path or marks a place where concurrent path segments begin or end.



Basic Path. The most basic path consists of a beginning marked by a waiting place and an end marked by a bar.



Responsibility. A responsibility is a named, short prose description of some localized action a system must perform.

Some other useful UCM notation is listed in Table 3-1:

Unbounded UCM	Notation	Explanation
	OR join OR fork	The effect of this notation is to join multiple scenarios without synchronization. The path to follow is determined randomly or conditions (guards) could be added to forks.
	OR fork OR join	The effect of this notation is to take one random path or conditions (guards) could be added to forks.
	AND fork AND join	The effect of an AND fork is to split an entering scenario into multiple concurrent paths. AND join is used to end concurrency
	Pure Synchronization	One entering scenario per incoming path is synchronized with the others. After the synchronization, it follows multiple concurrent paths.


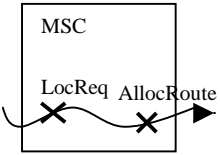
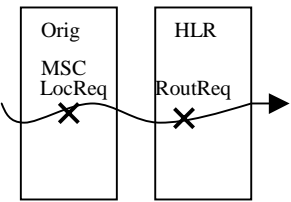
	Rendezvous	One entering scenario per incoming path is synchronized. All the synchronization scenarios must follow the shared path as one.
---	------------	--

Table 3-1 Notation of unbounded UCM

At later design stages, the UCMs become bound. This means that responsibilities are assigned to components. Therefore, bound UCMs represent causal paths that cross components. The binding is made with labeled responsibilities that are both positioned along the paths and imposed on the components. The binding component illustrated in Table 3-2 is called *Team*. It may be replaced with other suitable component types if necessary. In total, there are six types of operational components in UCM notation. *Teams, processes, objects, interrupt service routines, slots, and pools*. Three of them are used in this thesis: *teams, processes* and *objects*. More details of these three component types will be discussed in Chapter 4.

Notation	Explanation
	Responsibilities bound to the same component “ <i>MSC</i> ”
	Responsibilities bound to different components, such as “ <i>OrigMSC</i> ” and “ <i>HLR</i> ”

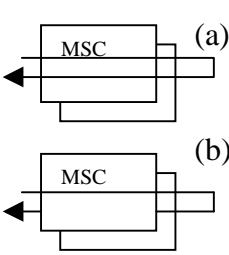
 <p>(a)</p> <p>(b)</p>	<p>A component stack means that many identical but distinct components are bound to the same path in the same way.</p> <p>In diagram (a), any particular scenario applies to one selected component <i>MSC</i>.</p> <p>In diagram (b), any particular scenario might apply to different selected component <i>MSC</i>.</p>
---	--

Table 3-2 Notation of Bound UCM

3.3.2 An Overview Of Message Sequence Charts (MSC)

Message Sequence Charts (MSCs) [ITU-T96] [RGF96], are a graphical and textual language that describes the interactions between system components. It is a well-known and mature notation. The main application for MSCs is to specify the communication behavior of real-time systems. MSCs focus on the communication behavior of system components and their environment by means of message exchanges. A set of MSCs usually cover only a part of a system's behaviors since each MSC represents only one scenario. Thus, the main focus of MSCs is not on complete system descriptions but rather on the specification of special system properties or functions. MSCs are frequently used in requirement specification, simulation, validation, and test-case specification.

To define the message exchanges, we need to follow some rules. Such rules can be provided in the notation. Each scenario has two kinds of notations: One is the graphic format *MSC/GR*. The other is the textual format *MSC/PR*.

MSC/PR represents an MSC in a pure textual format. This is particularly important where execution sequences have to be recorded. *MSC/PR* is usually used internally by tools such as SDT. *MSC/GR* is a graphic scenario diagram convention. As shown in Figure 3-4, each event identifier (step) in a scenario diagram is accompanied by a textual

description of the information flow involved. A tabular listing of the parameters follows the scenario steps that involve information flows. In the WIN documents, such diagrammatic convention is used to illustrate the information exchanges between network entities.

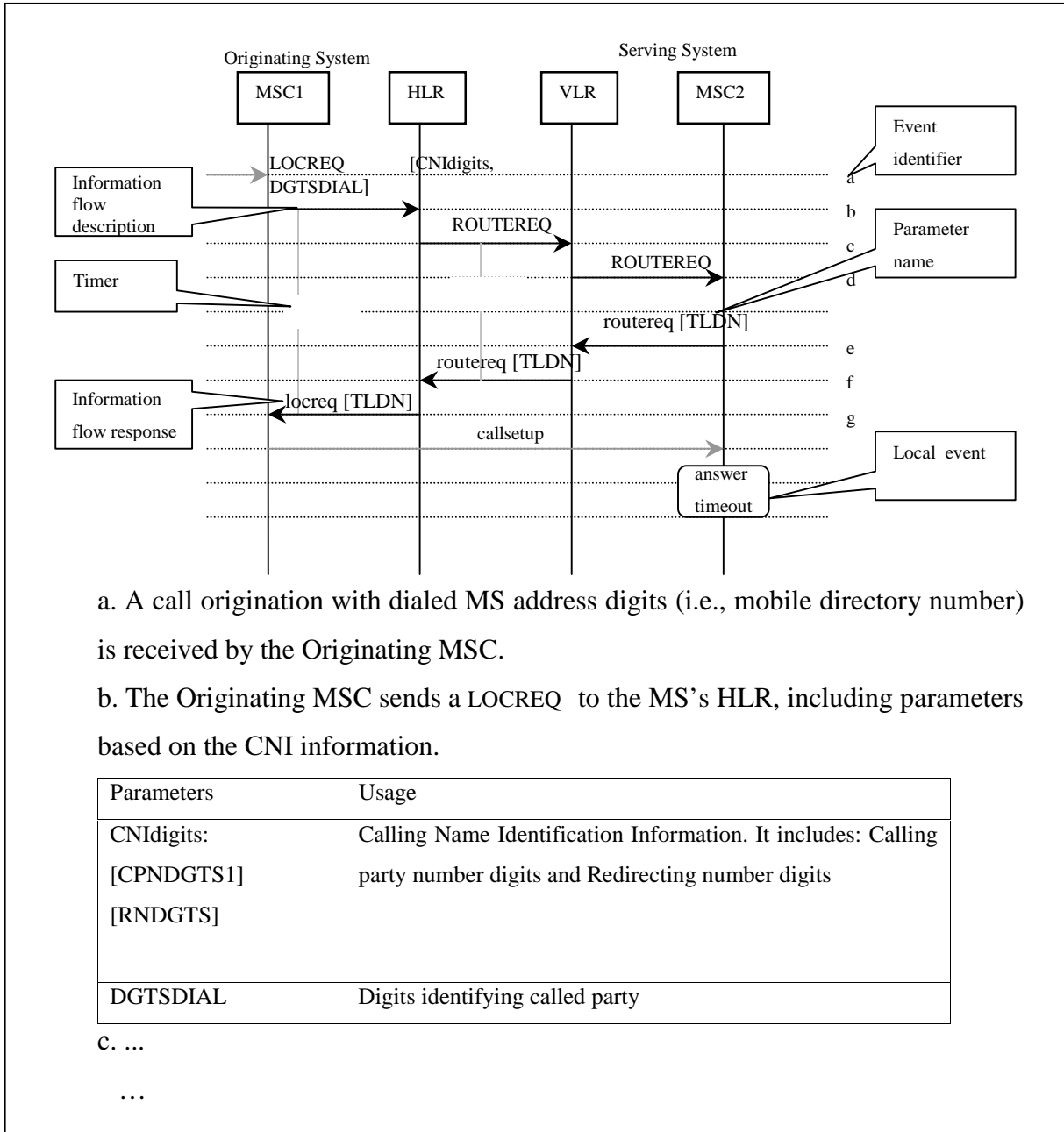


Figure 3-4 MSC GR representation

In order to combine the advantage of both MSC syntax formats, a translation of syntax forms is allowed. MSC/PR representation can be transformed automatically into a corresponding MSC/GR representation and vice versa. An example of the MSC /GR and of the corresponding MSC / PR representation is shown in Figure 3-5.

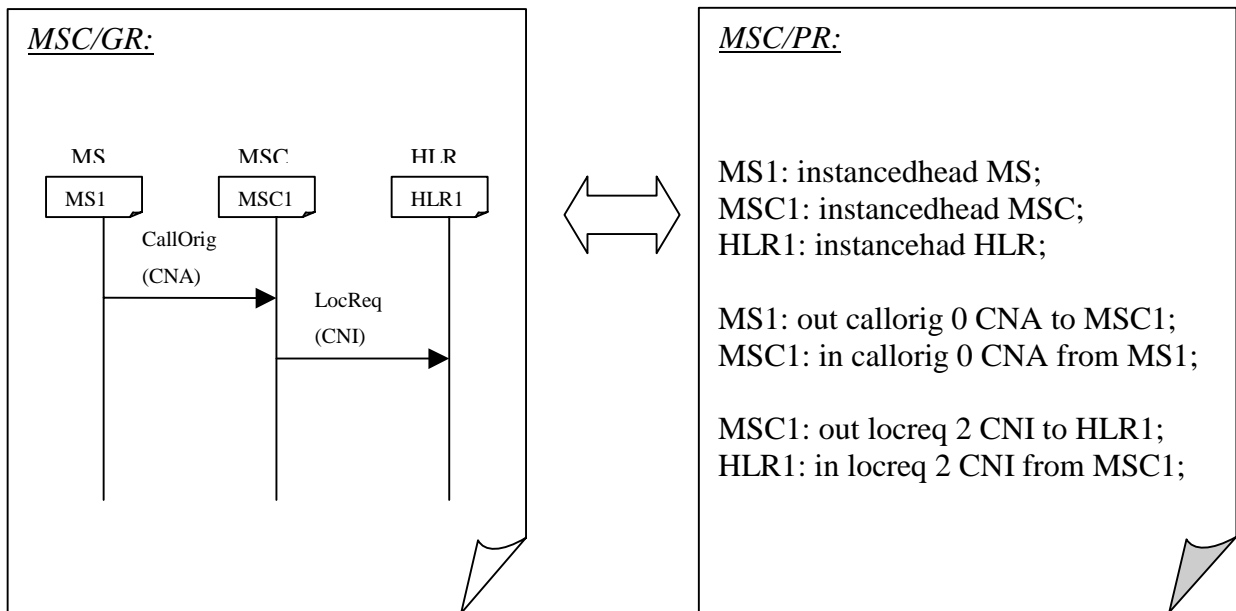


Figure 3-5 MSC in MSC/GR and in the corresponding MSC/PR

3.3.3 Overview of the LOTOS Language

LOTOS (Language Of Temporal Ordering Specification) is a Formal Description Technique (FDT), standardized by ISO for describing the formal specification of open distributed systems. It has been an ISO standard since 1989. LOTOS specifications can be used for very complex systems. Nowadays, LOTOS applications have been extended to cover many domains including telephony system. Furthermore, under certain condition, LOTOS is executable. LOTOS models allow the use of a number of validation and verification techniques such as step-by-step execution (simulation), random walks,

testing, expansion, model checking, and goal-oriented execution. A number of good tools were developed for validation purposes. LOLA, as an example, is the LOTOS tool used in Chapter 6 to validate our LOTOS specification.

LOTOS has an abstract data part and a control part. The first part defines all the data types and value expressions based on the formal theory of algebraic abstract data types, ACT-ONE [EM85]. The second part describes the behaviors of the system based on Milner's Calculus of Communicating Systems (CCS) [Mil80] and Hoare's Communicating Sequential Processes (CSP) [Hoar85].

3.3.3.1 LOTOS Data Part

Abstract types describe the possible data values and the operations on them without saying how they are actually represented and manipulated in memory. The summarization of the LOTOS data type is listed as following:

Basic Types

Commonly required data types can be included from the standard library to save time and space. For example:

```
Library
  Boolean, NaturalNumber
Endlib
```

The type *Boolean* has been specified with terms of sort *Bool*. There are the usual constants *true* and *false*, and the *not* operation that complements a Boolean value. The *NaturalNumber* type specifies whole numbers(positive or zero), which should again be familiar. There are also some extended capabilities for specifying abstract data types, such as the combination and extension, combination, conditional equations and the renaming. They will be introduced as follows.

Extension

Types may be extended with new sorts, operations and equations. In this example, the new type *NaturalNumber* enriches the Boolean type character with an *Is_0* operation for comparison with 0.

```

TYPE NaturalNumber IS Boolean
  SORTS
    Nat
  OPNS
    0 (*| constructor |*),
    ...
  EQNS
    FORALL x, y, z, w: Nat
      OFSORT BOOL
        Is_0 (0) = true ;
    ...
ENDTYPE

```

Combination

Types may be combined to build more complex types. In this example, the sorts, operations and equations of *TypeMDNID*, *TypeMINID*, *etc*, are all included to produce the richer type *ADDRESS*.

```

type TypeADDRESS
is TypeMDNID, TypeMINID, TypeMSCID, TypeVLRID, TypeHLRID
sorts
  ADDRESS
opns
  Undefined :->ADDRESS (*constructor *)
  ...
eqns forall d:MDNID, x:MINID, ..
ofsort MDNID
  GetMDN(ADDRESS(d,x,y,w,z))=d;
ofsort MINID
  GetMIN(ADDRESS(d,x,y,w,z))=x;
...
endtype(* TypeADDRESS *)

```

Conditional Equations

The applicability of an equation *eleof(a, insert(b,l))* may be made to depend on a Boolean *condition* $h(a) \text{ eq } h(b)$. If the condition holds for a given set of values, the equation applies for these values.

```

Type TypeACTION_List is TypeACTION
sorts ACTION_List
opns
  {} :-> ACTION_List (* constructor *)
  insert: ACTION, ACTION_List -> ACTION_List (* constructor *)
  eleof: ACTION, ACTION_List -> Bool

```

```

eqns forall
  a, b:ACTION, l: ACTION_List
  ofsort Bool

  eleof(a,{}) = false;

  h(a) eq h(b) =>
  eleof(a, insert(b,l)) = true;
  ...
endtype

```

Renaming

VLRID indicates the natural identification of VLR. Since *VLRID* and Natural Number are similar, the rename facility of LOTOS is used, which allows a new type *VLRID* to be specified by changing the names of Natural Number

```

type TypeVLRID is NaturalNumber renamedby
sortnames
  VLRID for Nat
Endtype

```

3.3.3.2 LOTOS Control Part

The control part of LOTOS is typically written as follows:

specification *spec_name*[*g1,g2,..gn*] (*v1,v2,..vm*)

behavior

<behavior expression>

where

<process definition>

endspec

The syntax of a process definition is of the form:

process *proc_name* [*g1, g2, ..gn*] (*v1,v2,..vm*)

<behavior expression>

where

<process definition>

endproc

LOTOS behavior expressions consist of basic behavior expression, such as *stop* and *exit* and *process instantiation*, as well as actions and behavior expression combined by means of operators, such as prefix, choice, enable, disable, parallel composition, etc. A LOTOS action consists of a gate name followed by 0 or more experiments. Experiments can be queries or value offers. For example, *AUm ? a:event !MINID !id* is an action on gate *AUm*. This action queries the environment for a value for variable *a*, and offers to the environment values contained in variables *MINID* and *id*. The environment can be either a process that is composed in parallel with the process containing the action, or the external environment to both processes. Action '*i*' denotes an internal event to a system. It can be written explicitly in the specification, or can be obtained as a result of *Hide*. Table 3-3 shows a brief summarization. More detail is available in [LFH92]

Behavior Expression	Name	Explanation
Stop	Deadlock	Can not engage in any interaction (deadlock)
Exit	Successful Termination	Indicates that a process has successfully performed its actions.
a; B	Action Prefix	<p>Defines the order of actions. The action prefix operator is written as a semi-colon.</p> <p>For example, when a user's phone ring and then the user picks up phone to answer this call. This can be expressed by a behavior expression composed of two actions synchronizing at the gate <i>Aum</i> and offering two values which are <i>Alert</i> and <i>CallAnswer</i></p> <pre>Aum !Alert...; AUm !CallAnswer...; Exit</pre>
B1 [] B2	Choice	<p>Allows the user to define different alternatives for a given process.</p> <p>For example, when a user gets the phone ring, he/she can either answer the phone or not. This could be expressed by the behavior expression:</p>

		<pre>AUm !CallAnswer ...; [] AUm !CallNoAnswer ...;</pre>
B1 >> B2	Enabling	Used to sequence two behavior expressions. B1 has to exit before B2 executes.
B1 [> B2	Disabling	Used to express situations where B1 can be interrupted by B2 during normal functioning, even before B1 starts
B1 [g1,...,gn] B2	Parallel Composition	<p>Composition in which B1 and B2 synchronize at gate g1,...,gn</p> <p>For example, consider two processes <i>Subscribers</i> and <i>WIN_CNAP</i> synchronizing on a gate <i>AUm</i>, and this case is described as:</p> <pre>Subscribers[AUm...](...) [AUm] WIN_CNAP[AUm...](...)</pre>
B1 B2	Interleaving	<p>Composition in which B1 and B2 behave independently. It expresses the concept of parallelism between behaviors where no synchronization is required.</p> <p>For example, Mobile Stations in the network behave independently of each other. Each Mobile Station is represented by a process <i>MS</i> having its network identification as a parameter. The behavior expression specifying three <i>MSs</i> in the network is :</p> <pre>MS[AUm](id1) MS[AUm](id2) MS[AUm](id3)</pre>
B1 B2	Full Synchronization	Composition in which B1 and B2 synchronize on all their gates
Hide g1,...,gn in B	Hiding	Used to hide actions (g1,..., gn) which are internal to a system. These actions cannot synchronize with the

		environment.
[P] -> B	Guarded Behavior	<p>B can be executed if P is true</p> <p>For example, a CNAP service can be triggered if the user subscribes to the feature. This could be expressed by the behavior expression.</p> <pre>[eleof(CNAP, GetSubscribedFeatures(MDNStatus] -> CNAP[AUm, C1](servMSC_id, GetMIN(DGTSDIAL), Get MDN(DGTSDIAL), CNIdigitsBCD, NAME)</pre>
Let x : s = E in B	Local Definition	<p>Substitute a value expression (E) by a variable identifier (x) of sort s in B.</p> <p>For example, a user has an initial status which includes his/her name, registration location, and some subscribed features. We describe this by the definition expression.</p> <pre>let Status_A:MDNStatus = MDNStatus(A, Name_of_A, ADDRESS(A, 1, 1, 1, 1), false, insert(CNAP, insert(RND, { })))</pre>

Table 3-3 LOTOS behavior expression

3.3.3.3 Specification Styles of LOTOS

LOTOS can be used at many different abstraction levels. A major concern when producing LOTOS specification is the selection of appropriate specification structure or styles. LOTOS supports four well-defined specification styles [VSSB89], called a *monolithic*, a *constraint-oriented*, a *state-oriented*, and a *resource-oriented* style. Following is a brief discussion of the different styles and their application in LOTOS specification.

In the monolithic style, all possible sequences of actions are presented explicitly. It allows only sequential compositions, choices, guards and recursions, and thus the

specification appears as a tree of alternatives. According to Milner's expansion theorem [Mil80], every LOTOS specification can be transformed step-by-step into a monolithic one. This style is very useful for the design of simple specification, but the exclusion of more complex operators, such as parallel operators, causes difficulty to produce large-size specification.

In the constraint-oriented style, specification is a parallel composition of processes. Each process represents a constraint. Those process do not denote implementation modules, they are pure constraints. All processes must be simultaneously satisfied. Such style makes the design of the specification easy to extend and modify.

The state-oriented style identifies system states by using state variable. States change while parameters change [VSSB89]. This style is good for specifications mainly when explicit states exist in problem definition. Also, states can be described more convenient as processes. However, for the lacking of structure, it is not suitable for large and complex system.

In the resource-oriented style, components become visible. Processes that represent the actual physical resources describe the system. Resources are implementation modules and defined by a temporal ordering of both internal and external interactions. Interactions among internal module are hidden. External interactions are through clean interfaces. This style allows modularity and parallel operators. It has flexible structure.

Each one of these specification styles has its own use. The choice of which style to use depends on many factors such as the requirement definition, the size and complexity of the system, etc. In this thesis, we are going to adopt the resource-oriented style.

3.4 Evaluation

In this section, we will evaluate the three selected techniques according to a wide range of criteria on the basis of our own experiences with specification techniques and on existing surveys [AALSY99] [WPJ98] [DW96] [CGR94]. The criteria is summarized as followings:

- *Readability*: descriptions need to be readable by domain experts (and not only by experts in the description technique). There is a strong emphasis here in human understanding, and in common understanding among different participants, including the client.
- *Abstraction*: this criterion is concerned with the level of detail that needs to be addressed, and with separation of concerns. An abstraction mechanism that supports two-way tractability allows to go from complex and high-level viewpoints to simple and low-level viewpoints.
- *Looseness*: in the early stages of the standardization process, few details are available, and a loose description technique should permit some level of incompleteness and non-determinism in a description.
- *Simulation and validation*: V&V is greatly improved when descriptions can be executed, simulated, and tested. Also one should be able to obtain test cases from formal description.
- *Tool Support*: with a good tool support, we can easily capture, edit, simulate, and test the description. We are especially interested in multi-platform, qualified tools, where support and training is available.

Technique	Readability	Abstraction	Looseness	Completeness & Consistency	Simulation & Validation	Tool Support
UCM	+	+	+	-	NA	-
MSC	+	-	0	-	NA	+

LOTOS	0	+	-	+	+	0
-------	---	---	---	---	---	---

Table 3-4: + = Strength; - = Weakness; 0 = Adequate; NA = Not applicable

Table 3-4 summarized the evaluation result. UCMs are based on a graphical, intuitive notation which is highly readable. In UCMs, there is no commitment to interfaces, protocols, methods, messages, state machines, or anything else about how the components are implemented, or how other components interact with them. Because of this character, it is relatively easy to combine system behavior patterns and accomplish component decompositions for an initial high-level system design. Therefore, it is a very powerful approach to give the designer looseness to specify system at a level of whatever detail is available. And also such characteristic of UCM allows us to specify system at different levels of abstraction. Because of the notation's informality and looseness, completeness and consistency checking, as well as verifiability, become difficult issues and are hard to support. Only one prototype editing tool is available for UCMs. It is called UCM Navigator.

Being simple and intuitive in nature, MSC is the other visual notation which provides strong readability. But each MSC only focuses on disjoint scenario, therefore abstraction is fairly weak. Because of the fact that MSCs are not executable, they are also poor for completeness and consistency checking, simulation and testing,. However, MSCs are widely used to represent the result of V&V activities. Some industrially supported tools can be used for MSC editing, such as SDT MSC editor.

LOTOS can be used at many different abstraction levels. The specification style in LOTOS can adapt itself to different expressive needs. LOTOS requires precision. Hence it requires that action sequences be specified exactly, although several non-deterministic alternatives can be specified, Looseness is low. LOTOS is an executable language, many inconsistency and incompleteness problems have to be solved at the time the LOTOS specification is written. Hence, simulation and validation are well supported within

LOTOS methodology. A number of tools have been developed for supporting the V&V of LOTOS specification. Among them, LOLA is a step-by-step executor, a tool for testing. It is fairly robust, although it is not industrially supported. Table 3-4 is explained further in [AALSY99].

3.5 Recommendations

Based on the evaluation result, Figure 3-6 shows the recommended techniques applied to the three stages of the standard drafting process/lifecycle that we will follow in this thesis. UCMs being an intuitive notation that can be used loosely and at several levels of detail, is appropriate for the first stage of the standardization process. LOTOS, being fairly precise but being able to be used abstractly, can come into play at Stage 2 and carry through Stage 3. MSC are good for intermediate description of protocols and procedures, but neither for the early stage, not for the final one. They can be generated from UCMs, and can guide the prototyping of LOTOS description in stage2 and 3.

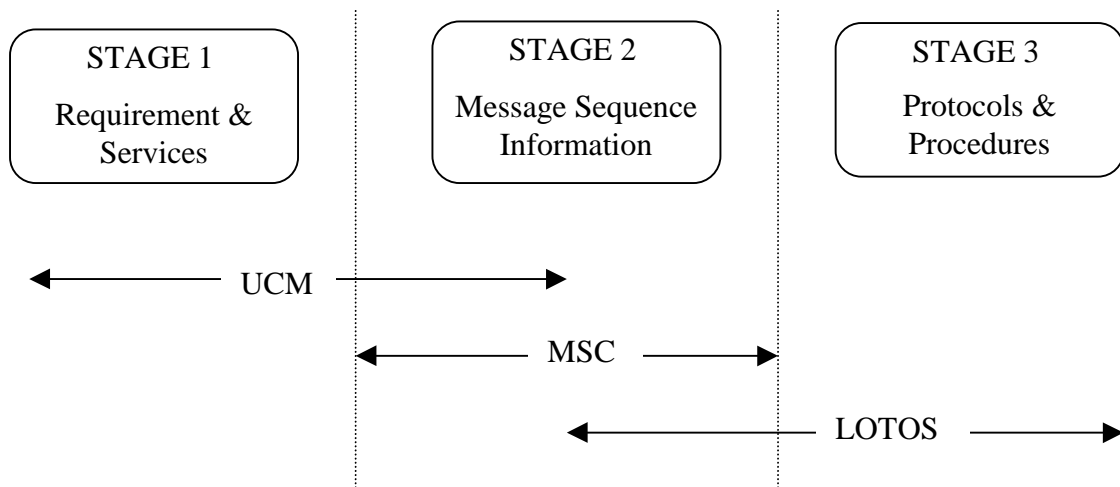


Figure 3-6 Relevant Methods for the Three Drafting Stages.

Chapter 4 Scenario Oriented Analysis for CNAP service in WIN

4.1 Introduction

In this chapter, we will present a scenario-oriented analysis of Wireless Intelligent Network Service Design. The complexity of real-time and distributed system architecture makes its design a true challenge. The current network design information of CNAP service [WIN98] is represented in the format of MSCs, tables, and procedures. It specifies the message interactions between components, it defines parameters included in each message, and it specifies the functional behavior of the system in terms of pseudo code. Although the current design information gives us substantial detail, it lacks a complete high level view of the system. We need to know, in a general way, what network entities and responsibilities should be included in the system, and where those responsibilities should be performed. The purpose of this analysis process is to produce a high-level scenario description of the system. Our technique is based on Use Case Maps (UCMs). This technique helps in specifying the designed system at a high level, and contributes a method for generating test cases for the further validation.

The chapter is organized as follows. Section 4.2 shows use case maps from the service description stage of the current WIN standard. This section presents unbound UCMs of the Call Name Presentation (CNAP) service description from both a user-centered and a system-centered viewpoint. Section 4.3 concerns the message sequence information stage. At this stage, beside expressing UCMs in terms of responsibility sequences, we define component types where responsibilities are applied, and then we can add architecture details such as component types, the communication framework, and interaction sequences. Bound UCMs and Message Sequence Charts (MSCs) are applicable at this stage.

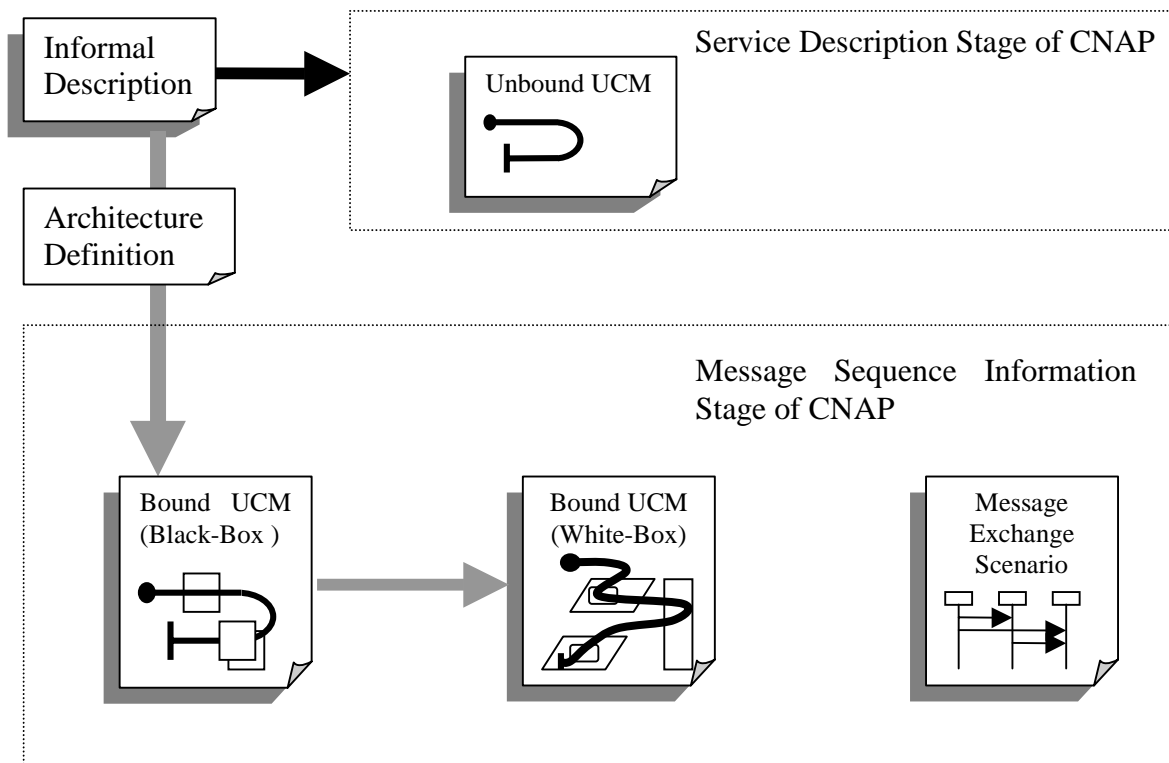


Figure 4-1 Scenario Oriented Analysis of CNAP

As a result of the analysis, we will provide a high-level perspective of system behavior in terms of UCM and MSC scenarios. This scenario-oriented technique, which is used here for a service in the Wireless Intelligent Network, is a general one and can be applied to other types of distributed systems.

4.2 Service Description Stage of CNAP

Here, we present the steps involved in the production of the first use case map. At this stage, we apply only the unbound use case map technique to CNAP service description. There is no commitment to components for responsibilities at this stage. The following scenarios are grouped into two categories: a high-level service description from a user's perspective, and a underlying network communication from a system designer's perspective.

In our following UCM pictures, we call a subscriber who starts a call *originator*, and a subscriber who receives a call *terminator*.

CNAP/RND Service Subscription UCM : Both originator and terminator can subscribe to a WIN feature at any given time, but they only need to subscribe once. Redirecting Name Delivery (RND), which is a CNAP subscription option, is represented as an alternative path of service subscription by an OR-Fork. Users who subscribe to CNAP can also subscribe to RND. After the user subscribes to either feature, s/he becomes an authorized subscriber. S/he can activate the feature by setting it *permanent active*, or *on demand active*. If the value is set as *on demand*, then the subscriber is authorized to control the activation and de-activation of the feature. The control of feature activation is not specified in the CNAP/RND service subscription UCM, because it is not directly related to service subscription. We will describe such behavior in the *originator* UCM and *terminator* UCM. The plug-in WIN database management stub, which is shown with a diamond-shape notation, will also be explained when we discuss the WIN Database management Stub UCM. By applying the stub notation, we are able to abstract the network level design, and let it be transparent to the client at the early service description stage. The stub notation also gives us the advantage to construct a complete scenario by recursively selecting appropriate plugins for the stubs.

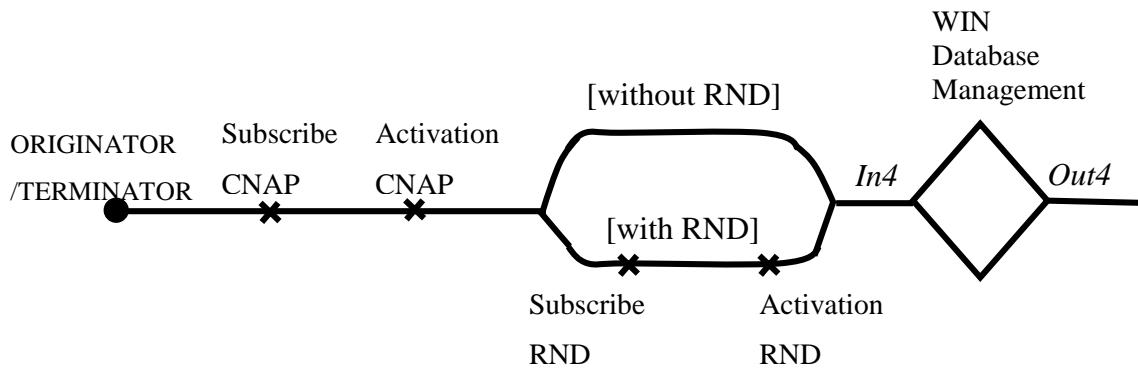


Figure 4-2 CNAP/RND service subscription UCM

Originator UCM: when a user starts a call, s/he sends a *Call Origination* request to the system, which in Figure 4-3 is represented as a *WIN CNAP/RND stub*. By using such stub at this level of design, details of the WIN network, such as the intercommunication of network entities or network data used to set up the call, are invisible to the originator. After the call link is established within the *WIN CNAP/RND stub*, there is a waiting place along the originator's path to wait for an event occurring from a terminator's path. Such event can be either that the terminator answers the call or that s/he does not answer the call. After this event is performed by the terminator, the originator will either *Get A Call Answer/Call No Answer*, or *Get a Busy Tone*. Meanwhile, the originator is able to do other things concurrently, i.e. s/he can roam to another area at any time. Therefore, the *Location Update* action can happen before the start of the call, or in the middle of the call.

An originator that has activated the feature by setting it *on demand*, can also start by performing a "*feature active update*". However, this can't be done in the middle of the call. Therefore, OR_Fork is used to specify the two exclusive paths between the call origination path and the Feature Active Update path. The updated data is recorded in *the WIN database management stub*.

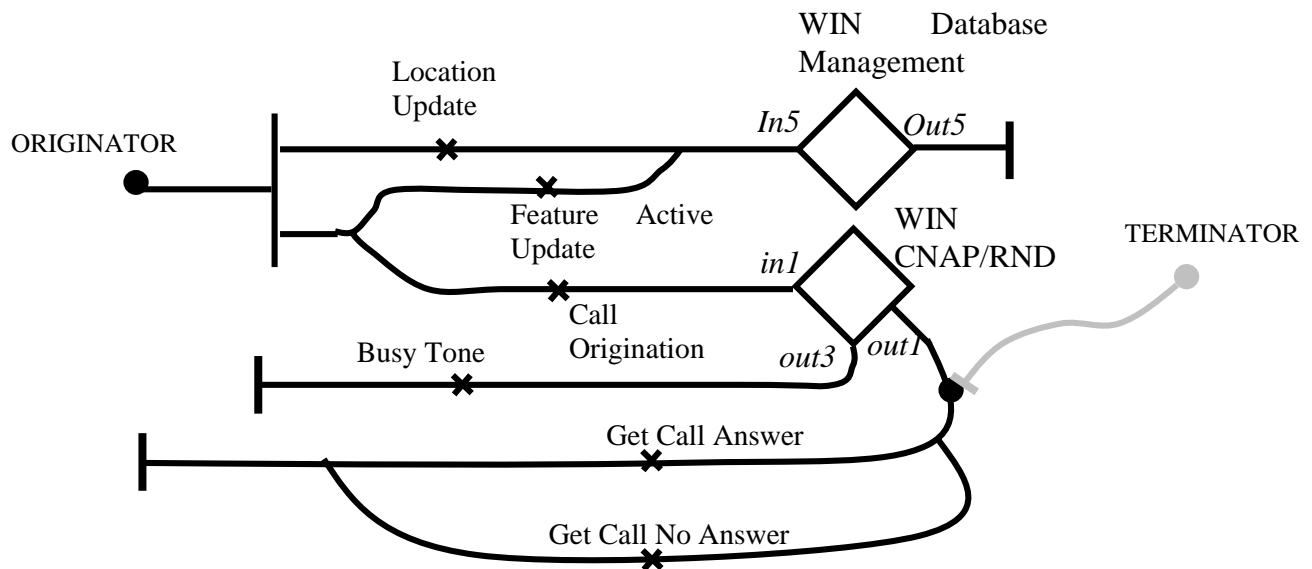


Figure 4-3 Originator UCM

Terminator UCM: Similar to Originator, a terminator may roam (*Location Update*) and may be able to open or close the CNAP and/or RND feature authorization anytime to allow or deny access to the CNAP feature (*Feature Active Update*). As a result of feature authorization, CNAP can be invoked for any terminating calls to a terminator who has CNAP active, and RND can be invoked for any terminating calls to a terminator who has CNAP and RND active when the call has been redirected. In **Figure 4-4**, there is a waiting place along the terminator's path to wait for a call origination from another user. In other words, an originator may try to contact a terminator at a given time and the event *call origination* in originator's path is considered as a stimulus for this waiting place to continue the terminator's path in the WIN CNAP/RND stub. If CNAP/RND service is invoked in the stub, normally the terminator is given a *Successful Alert* with appropriate calling name presentation information. But in some special case, s/he may be given an *Exceptional Alert* without the calling name information due to system failure. After getting either alert, the terminator can choose to answer the call (*Call Answer*) or not answer the call (*Call No Answer*). Normally, the path will end and the resulting event for the terminator's path causes *Get Call Answer* or *Get Call No Answer* event to happen in

the originator's path. In some particular cases, the terminator needs to redirect his/her call. Such cases are represented as a backward path to the WIN CNAP/RND stub.

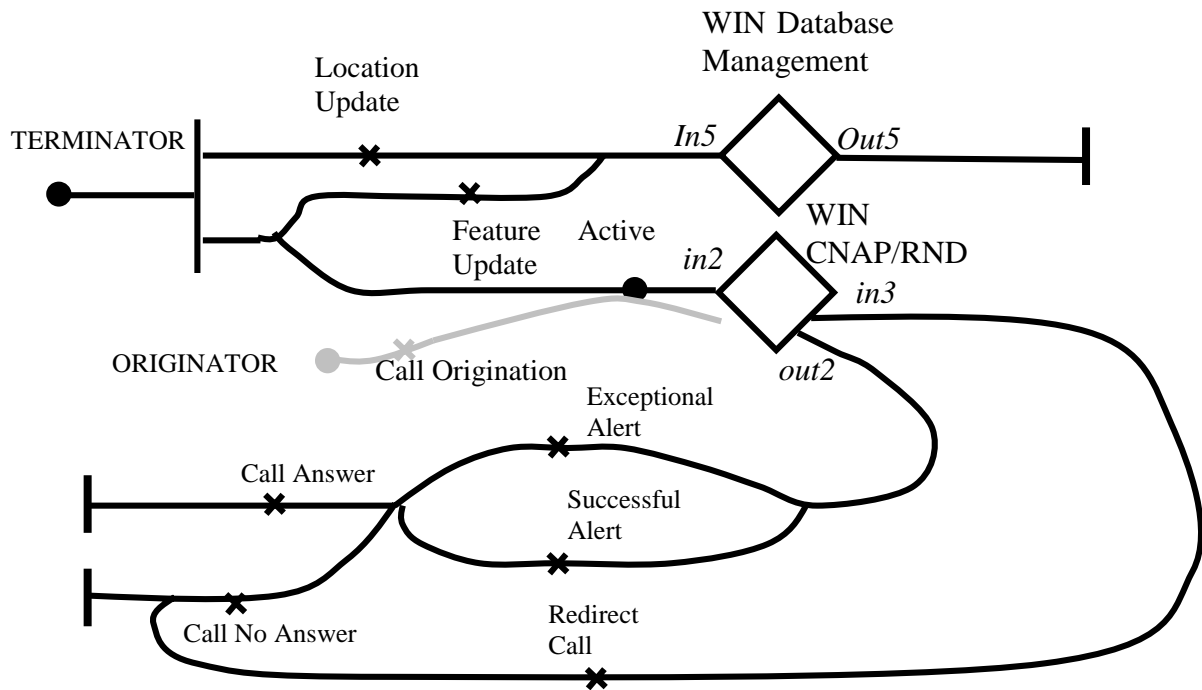


Figure 4-4 Terminator UCM

WIN CNAP/RND Plug-In UCM: The following plug-in UCM is bound to the WIN CNAP/RND stub of the originator and terminator UCMs. In the originator and terminator's UCM, the entry and exit points are marked as lowercase: *in1, in2, in3, out1, out2, out3* while the start and end points in the stub UCM are marked by the same name, but in uppercase. The binding is made by associating the entry and exit points to the start and end points of the plugin map { (in1, IN1), (in2, IN2), (in3, IN3), (out1, OUT1), (out2, OUT2), (out3, OUT3) }.

The network design of CNAP can be very complex. The main responsibilities in CNAP service are *Location Request*, *Route Request*, *Page*, *Allocate Route*, *Call Connection*, *CNAP service request*, *CNAP/RND service provision*, *Call release*, *Redirection Request*. These responsibilities are large ones since it is useful to simplify use case maps by making some responsibilities very large. This is particularly useful in cases where interaction is required back and forth between components and we want to defer the details. There are three basic paths in the WIN CNAP/RND stub UCM. The precondition

of basic path IN1 is a call origination request event from an originator. The original calling name information and the destination calling number should be included in the request. Along this path, *location Request* is invoked from the originator's side to obtain call instructions. Then, *Route Request* is used to inquire the appropriate terminator's route information for the pending call. After the *Route Request*, path IN1 either exits (OUT3) or needs to be synchronized with another basic path IN2 in order to establish a voice link between the originator and terminator. Basic path IN2 represents a terminator's network side in the stub UCM. After the terminator is paged and the terminator's network assigns routing information, this routing information is returned to the originator's network side, based on which, the physical call connection can be established. After that, the path IN1 exits the stub as OUT1. Meanwhile, path IN2 continues *CNAP Service Request* and *CNAP Service Provision* for the terminator. Our WIN design is restricted to the CNAP service, and so we suppose that terminators subscribe the CNAP feature. After the service is provided, the path IN2 exits the stub as OUT2. There is a third basic path IN3 which indicates a *Redirect Call* event from a terminator. If that event happens, the previous call connection needs to be released and the new network connection needs to be started from path IN1 all over again.

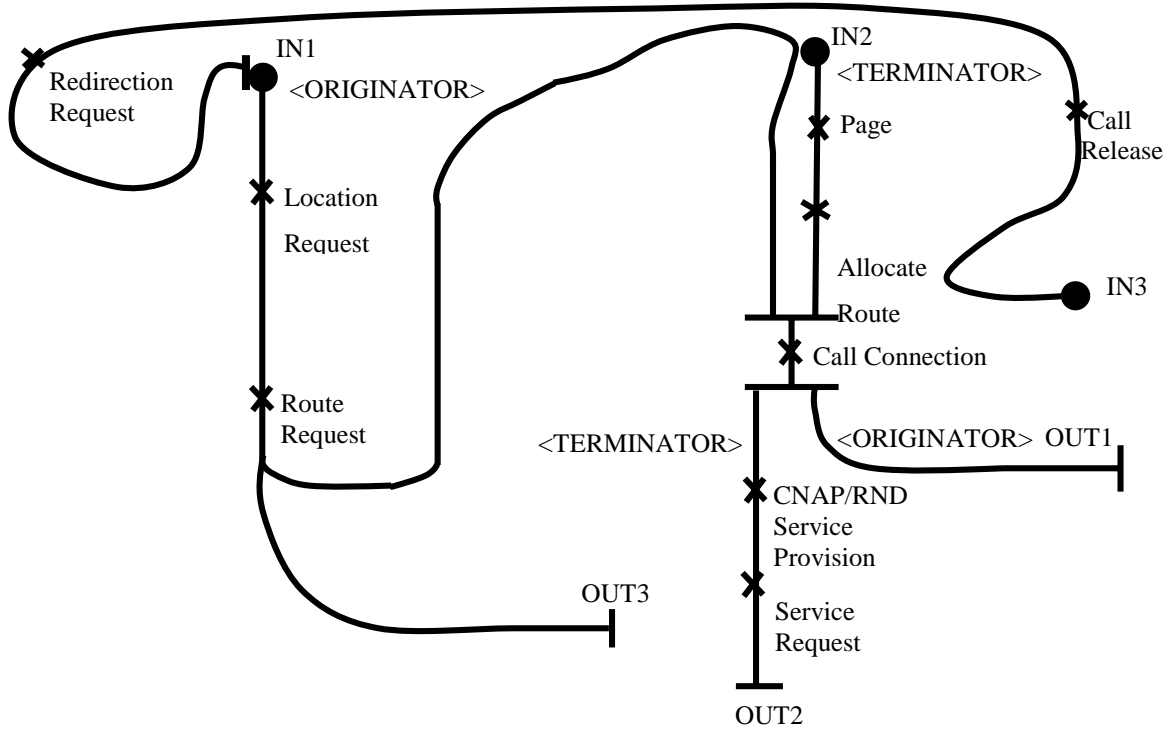


Figure 4-5 WIN CNAP/RND Stub UCM

WIN Database Management Stub UCM: Both Originator and Terminator UCM are bound to the WIN Database Management stub by associating the entry and exit points to the start and end points of the plugin stub map. In the originator and terminator's UCM, the entry and exit points are marked as lowercase: *in4*, *in5*, *out4*, *out5*. While the start and end points in the stub UCM is represented in uppercase. The binding is { (*in4*, IN4), (*in5*, IN5), (*out4*, OUT4) (*out5*, OUT5)}. There are two databases in WIN: Home Location Register Database and Visitor Location Register Database. The Home Location Register database provides the service data which contains permanent and temporary data fields. This data provides necessary support for HLR's responsibility to manage the mobility aspects for wireless users. The permanent data associated with the mobile station does not change as they move from one area to another, while the temporary data change from call to call. In our HLR database of a simplified WIN model, the data includes: Directory number of Mobile Station, Unique identification of Mobile Station, Name Information of Mobile Station Users, Subscribed features, Subscribed features status, and Supplementary services related parameters, such as Forwarded-to number, registration status, activation status. The VLR database only tracks the state of MSs in its area. The information necessary to process the call is contained in the HLR. It is transferred to the VLR once the mobile station enters into the VLR's area and is deleted after the mobile station leaves. By using the VLR database, the roaming subscriber's data is easily retrieved so that the call can be speeded up. In practice, the data contained in these two databases is more or less the same. Figure 4-6 presents two basic paths in the WIN database management UCM. Each path has a constraint start. This means that the database allows only one instance of the path to be initialized at a time. The old instance has to terminate for a new instance to start. For example, once a feature status is updated, the relative information stored in the Home Location Register database should be changed by *HLR Data Update*. Meanwhile, if there is a *HLR Data Query* invoked by some other components, the Database management has to finish the previous *HLR Data Update* in order to process the coming *HLR Data Query*. In other words, data querying and data updating can be requested concurrently, but they have to be sequentially processed in our database management design model.

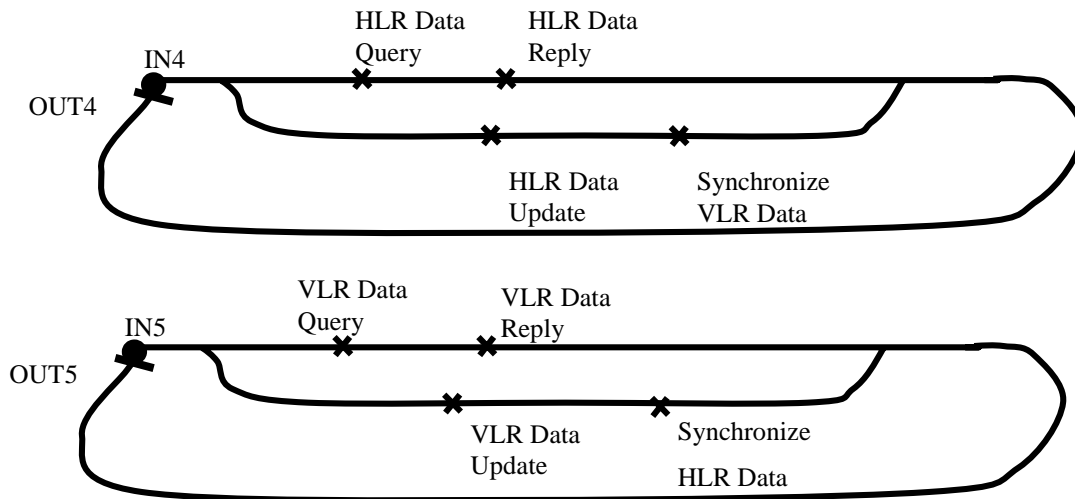


Figure 4-6 WIN Database management Stub UCM

On the basis of the five UCMs shown above, we can get a general picture of the CNAP service. A subscriber can register to the CNAP/RND service. At any time, s/he can start a call as an originator or s/he may receive a call as a terminator. When the CNAP service is invoked and the calling name is available, the terminator's terminal shall display the calling name information. In the event of call redirection when CNAP and RND are active, the name associated with the original party and the name associated with the last redirecting number will be presented to the terminator. The detailed CNAP/RND service in the network layer design is represented as a WIN CNAP/RND stub and as a WIN database management stub.

In our next stage, we will discuss the component types, the communication framework, and the interaction sequences based on these two network stubs.

4.3 Message Sequence Information Stage of CNAP

In Chapter 2, we discussed the network entities, function entities, communication interface and service deployment framework where the CNAP service should be invoked.

These give us the basis to define the component types of the UCMs and understand the message sequence between those components.

4.3.1 Component Types

The six types of operational components in the use case map notation were mentioned in Chapter 3. We could apply many of them to different level system design. In this thesis, three of them are used. They are teams, processes, and objects. The general mappings between UCM component and WIN architecture entities are: teams = network entities, processes = function entities, object = database. Each mapping is shown below:

- **Teams** 

A team is a default component to use when we do not know the exact nature of the component. It may include *objects*, *processes*, or other *teams*. A team can be treated as a black box suitable for the initial phase. Showing teams in use case maps implies the existence of components but defers the decision of the precise component.

In [WIN98], a network architecture is modeled by a set of network entities. These network entities communicate with each other through predefined interfaces. The related Network Entities in the CNAP service are: Mobile Station (MS), Home Location Register (HLR), Visitor Location Register (VLR), Mobile Switching Center (MSC) and Service Control Point (SCP). In our initial step, we declare each of these as a large grained network component and simply bind UCM paths through them. There is a many-to-many relation among components. For example, a VLR can control many MS in its geographical domain, and one MS can roam around and be served by many VLRs. To represent this in the bound map, we need to use a UCM stack². Inside the stack, each element is a distinct component, but is operationally identical. One element of the stack is selected at a given time, and more than one may be selected along different paths at the same time. The left half of Figure 4-7 is our simplified WIN network model of the CNAP

² Note that in UCM literature, the word “stack” is used to refer to a collection of components, not necessary first in last out.

service. This network is transformed into a set of UCM component stacks, which include MS teams, VLR teams, MSC teams, HLR teams, and an SCP team. Except for SCP, the other network entities are represented as stacks. The mobile station and Mobile Station Centre can be further split into two sets of component teams, based on the user's role in the call (i.e., Originating or Terminating).

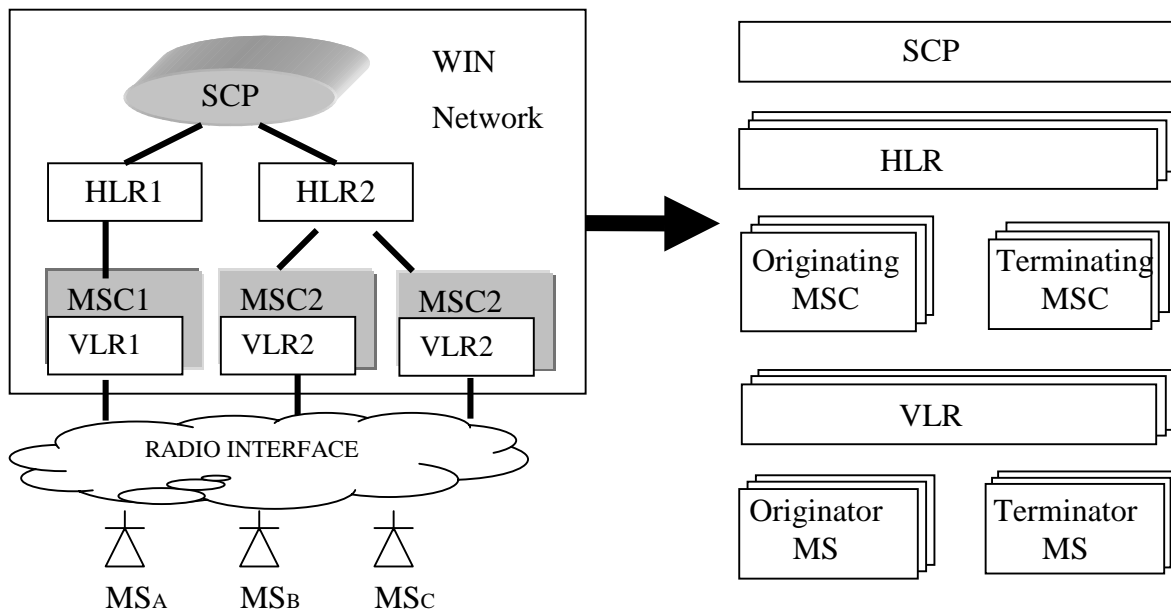


Figure 4-7 The Simplified WIN Network Model and UCM structure

- **Processes** 

A process is a component that may operate concurrently with other processes. Its internal logic is sequential; in other words, there are no concurrent elements inside a process. Essentially, multiple paths can be bound on a process. But the process will have to deal sequentially with the composition of the responsibilities along the paths that overlap it. Two concurrent UCMs might have to be treated sequentially if they both progress to a point where their responsibilities need to be performed by the same component. In our design, teams can be further developed into functional entity processes. Multiple paths are bound into some functional entity process and they are performed sequentially. For example, the Terminator MSC team contains SSF_CCF function process to handle call connection responsibility, and later on, the same process may need to perform *Call*

Release responsibility caused from another path. Our functional entity processes are based on Distributed Function Plane, for which we refer to Fig 2.2 of chapter 2.

Beside the functional entity process from the DFP, we also define a database management process to manage the HLR database and VLR database. The main purpose for this is to maintain the data retrieval and storage function and to keep data synchronization between VLR database and HLR database. With the help of this process, we are able to simplify our design model by abstracting from real implementation details such as how data are transferred from the HLR database to the VLR database, or how data are updated in the HLR database when a user roams, etc.

- **Objects** 

From the behavioral perspective of UCMs [Buhr96], an object is a fixed component that supports a data abstraction through an interface. The interface lets other components ask the object to perform its responsibilities. Objects are not further decomposable into other types of components. The main constraints of component decomposition are:

- processes and objects cannot include teams
- objects cannot include processes.

The other possibilities are allowed. In our design, we have a database management process. It contains the VLR database and HLR database. We define each database as an object component and put both of them into the database management process component. Some other components such as HLR team, VLR team or MSC team can send data query/update requests to the database management process, which can be considered as an interface of the objects.

4.3.2 Bound UCMs

In this section, we add components to the WIN CNAP/RND stub UCM and WIN Database management Stub UCM. In a bound UCM, responsibilities are bound to the components where they execute. Before we analyze the component type for each responsibility, we can choose a team as a default component for the first step. The team component is like a black-box, it is not necessary to know its internal structure, such as

the network function entity process or network database object. For example, a *Location Request* is performed by an originating MSC. We therefore simply bind this responsibility inside the MSC team.

In the second step, we can further decompose the component by replacing the team with other components. By either replacing the rectangle by another shape, or expanding it into more components, the teams may evolve into sets of objects, processes, or other teams. Table 4-2 shows the mapping between responsibilities and their bound component-type for the two plug-in UCMs.

Responsibility	Team	Process, Object
Location Request	Originating MSC team	SSF_CCF process
Route Request	HLR team, VLR team,	LRFH process, LRFv process
Allocate Route	Terminating MSC team	MACF process
Call Connection	Terminating MSC team	SSF_CCF process
Service Request	Terminating MSC team	SSF_CCF process
CNAP Service Provision	SCP team	SCF process
Call Release	Terminating MSC team	SSF_CCF process
Call Redirect Request	Originating MSC team	SSF_CCF process
HLR Data Query; HLR Data Reply; HLR Data Update;		Database management process, HLR database object
Synchronize VLR database		Database management process
VLR Data Query; VLR Data Reply; VLR Data Query;		Database Management process, VLR database object
Synchronize HLR database		Database Management process

Table 4-1 Black-Box Binding For Component-Based UCM

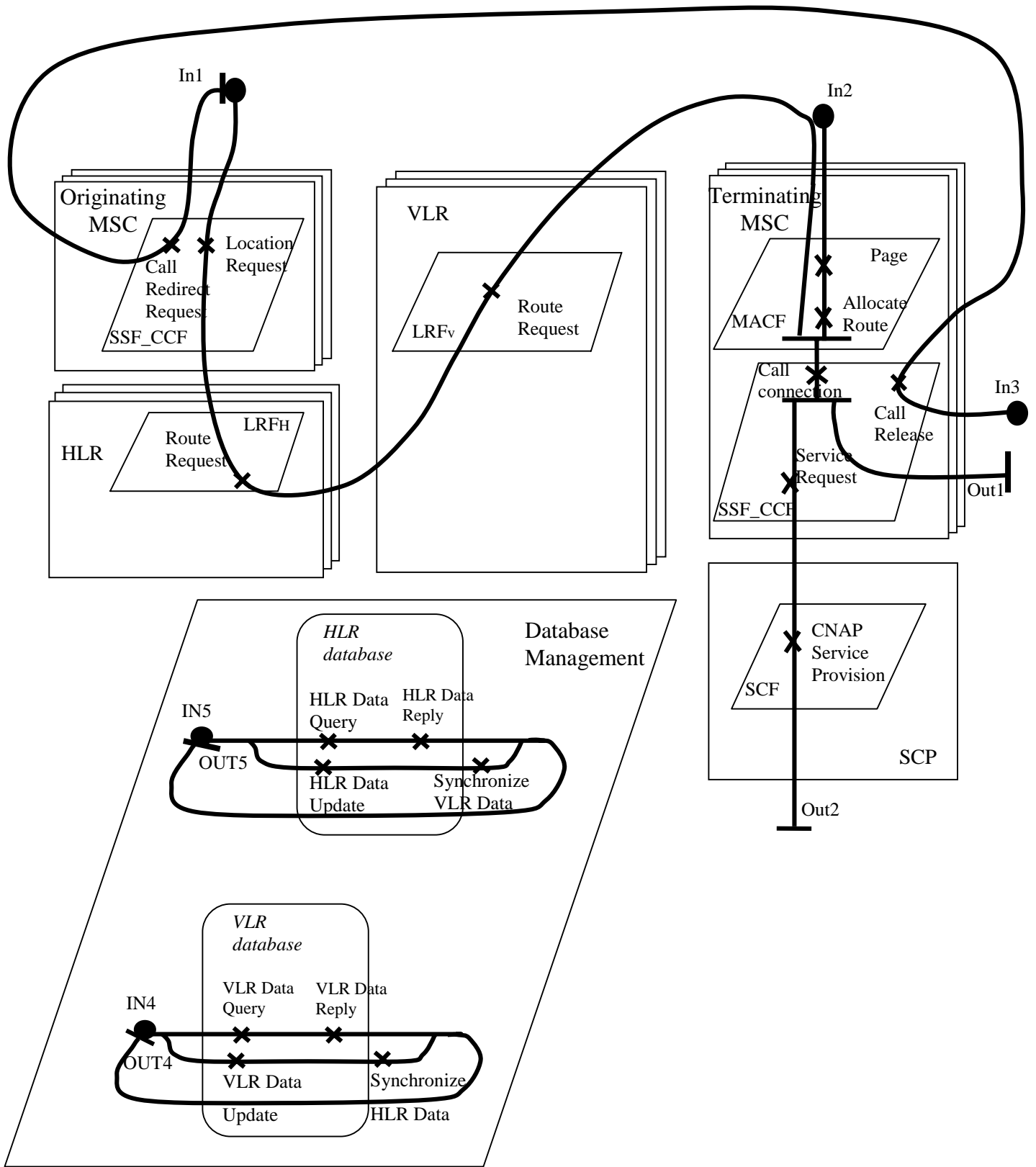


Figure 4-8 Bound Plug-In For WIN Stubs UCM

In Figure 4-8, to coordinate the call connection, at *IN1*, the originating MSC/SSF_CCF invokes the *Location Request* operation to obtain call instructions from a serving HLR, which serves for the destination party. Following the *Location Request* operation, the HLR/LRFH initiates the *Route Request* and sends it to VLR/LRFv. When the VLR/LRFv receives the *Route Request*, it forwards the request to the terminating MSC/MACF and waits for the route information to be allocated by the terminating MSC/MACF. At *IN2*, in response to the *Route Request*, the terminating MSC/MACF *pages* the called MS and determines whether the MS is currently idle. If the MS is idle, the terminating MSC/MACF allocates the routing information and sends such information back. After the originating MSC/SSF_CCF receives the routing information, the voice link is then established between the originating MSC/SSF_CCF and the terminating MSC/SSF_CCF. If the user registered the CNAP feature, the *Service Request* is invoked by the terminating MSC/SSF_CCF and the *CNAP Service Provision* is processed by SCP/SCF. Under certain circumstance, a call may need to be redirected. This case is described by path *IN3*. At *IN3*, the terminating MSC/SSF_CCF first releases the previous call (*Call Release*), and then the originating MSC/SSF_CCF sends a *Call Redirect Request* to the serving HLR in order to get the forwarded destination number. Once the originating MSC/SSF_CCF gets the forwarded destination number, it invokes the *Location Request* operation to obtain call instructions from a serving HLR. The whole procedure as described above will be started all over again. A Database Management process is also included in Figure 4-8. It should be clear, and so we do not explain it.

4.3.3 MSCs of CNAP service

Section 4.3.2 presents a bound use case map for the CNAP service. It gives us a more detailed design of system behavior. However, it does not include information on how components communicate and what quantities flow along paths. A causal relation from a UCM can be implemented in terms of many different message exchanges. Therefore, we need MSC, as a complementary design technique of UCM, for the message sequence

stage. In the current WIN standard, a set of MSCs related to CNAP has been documented. We can take inspiration from the standards MSCs.

During the time when the thesis was written, there were five CNAP MSCs in the WIN document. These MSCs illustrate some interactions among network entities in various situations related to the Calling Name Presentation (CNAP). They are summarized in Table 4-2. Here, we only show scenario 1 in Figure 4-9 as an example, others can be referred in [WIN ANSI-41.3-C Additions].

Scenario 1 (See Figure 4-9)	This scenario describes CNAP Invocation to an Idle MS with successful name display
Scenario 2 referred in [WIN ANSI-41.3-C Additions]	This scenario describes CNAP Invocation to an idle MS with unsuccessful name display due to a SCP Response Timeout
Scenario 3 referred in [WIN ANSI-41.3-C Additions].	This scenario describes CNAP Invocation with successful name display after call redirection
Scenario 4 referred in [WIN ANSI-41.3-C Additions].	This scenario describes CNAP Invocation by the HLR. The choice of the service is invoked by MSC or by HLR is predetermined by the system designer. As discussed in Chapter 2, we focus on the recommended service model where CNAP is invoked by MSC. Therefore, this scenario is not very useful for our purpose.
Scenario 5 referred in [WIN ANSI-41.3-C Additions].	This scenario describes both CNAP and RND Invocation with successful name display after call redirection.

Table 4-2 Five CNAP MSCs

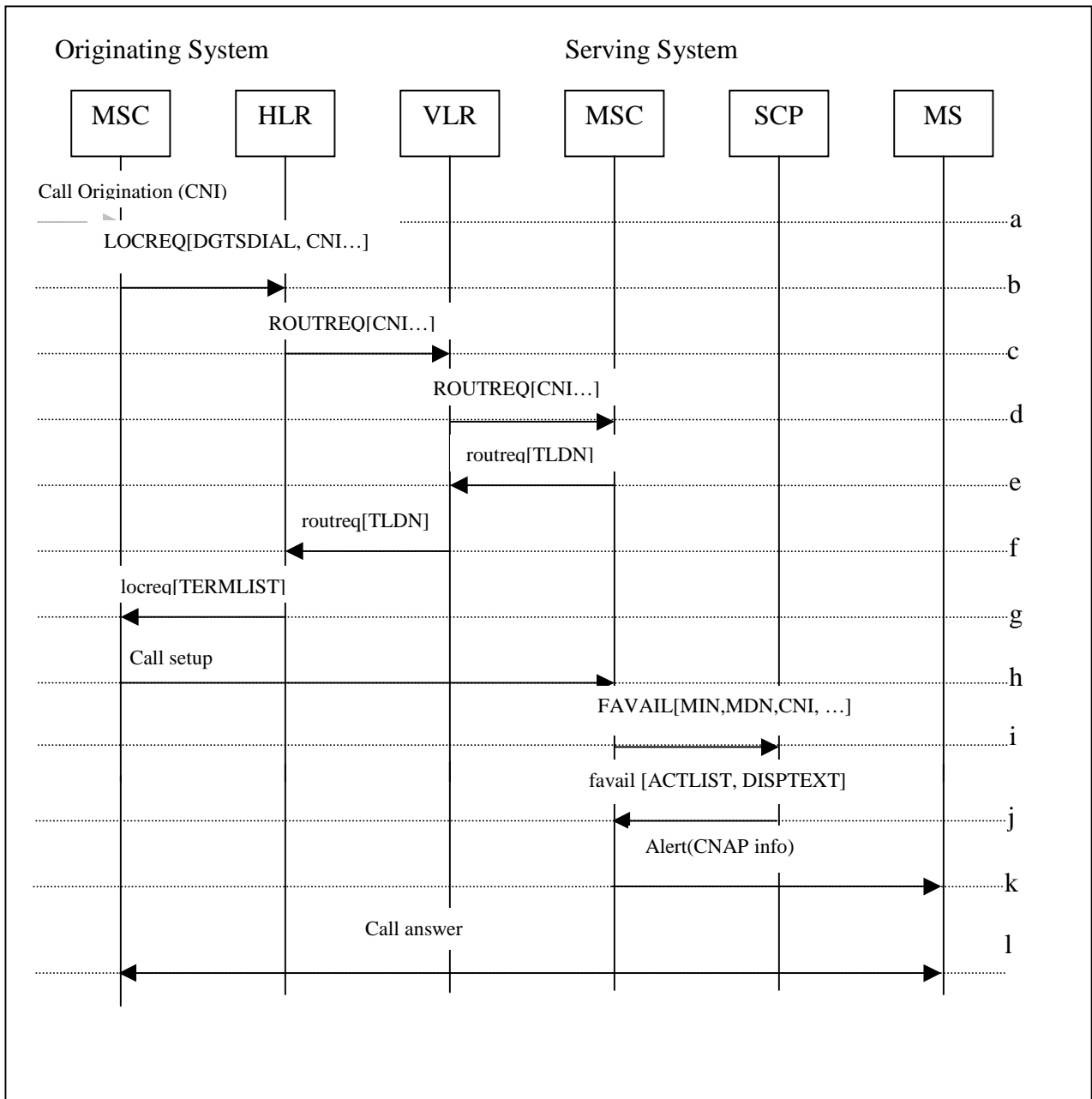


Figure 4-9 Scenario #1 Message Sequence Chart

This MSC represents:

- a.) A call invocation to an idle, authorized MSC with dialed MS address digits. (This is implied by *Call Origination* Responsibility in *Originator/Terminator* UCMs)
- b.) The Originating MSC sends a LOCREQ to the MS's HLR, including CNI parameters. (This is implied by *Location Request* Responsibility in Figure 4-8 UCM)
- c.) The HLR sends a ROUTREQ, including the CNI parameter, to the VLR where the MS is registered (This is implied by *Route Request* Responsibility in Figure 4-8 UCM)
- d.) The VLR forwards the ROUTREQ to the current Serving MSC. (This is implied by *Route Request* Responsibility in Figure 4-8 UCM)
- e.) In response to the ROUTREQ, the Serving MSC checks its internal data structures and determines that the MS is currently idle. Therefore the Serving MSC allocates a TLDN and returns this information to the VLR in the routreq. The Serving MSC stores the received CNI parameter. (This is implied by *Page* and *Allocate Route* Responsibilities in Figure 4-8 UCM)
- f.) The VLR sends the routreq to the HLR. (This is not shown in Figure 4-8 UCM, because it is considered to be an implementation detail)
- g.) When the routreq is received by the HLR, it returns a locreq to the Originating MSC. The locreq includes routing information. (This is also not shown in Figure 4-8 UCM, because it is considered to be an implementation detail)
- h.) A voice path is established between the Originating MSC and Serving MSC. (This is implied by *Call Connection* Responsibility in Figure 4-8 UCM)
- i.) The Serving sends a FAVAIL to the called party's SCP including the called subscriber identity. (This is implied by *Service Request* Responsibility in Figure 4-8 UCM)
- j.) The SCP sends the favail to the Serving MSC including the calling name information. (This is implied by *CNAP Service Provision* Responsibility in Figure 4-8 UCM)
- k.) When the inter-MSC call is received at the Serving MSC, the MS is alerted. (This is implied by *Successful Alert* Responsibility in *Terminator* UCM)

- 1.) When the served MS answers, the originating MS gets the answer. (This is implied by *Call Answer / Get Call Answer* Responsibilities in *Terminator/Originator* UCM)

4.4 Conclusion

In this chapter, we applied our scenario analysis technique to the requirements of the CNAP system. The analysis includes building a design model composed by UCM and MSC. Such analysis brings us one step closer towards the specification and validation of CNAP in LOTOS, which is discussed in the next chapters.

Chapter 5 LOTOS Specification of CNAP

5.1 Introduction

In this chapter, we describe the LOTOS specification of the WIN model and of the CNAP service based on the UCMs and message exchange scenario information analyzed in Chapter 4. Our main objective for the specification is to capture the descriptions of this IN feature in a distributed and concurrent communicating wireless environment.

By using LOTOS, we define data by using Abstract Data Types (ADTs) and we describe the system's externally observable behaviors as well as internal ones. The specification structure we choose to use is resource oriented [VSSB89]. This style allows us to show the architectural components of the system. As described in Chapter 2, the most important network components that are relevant to build our WIN model are MS, MSC, VLR, HLR, SCP. These network components cooperate with each other and provide the basic architecture for the deployment of IN features in a wireless environment. Some other function related network entities are not included since they are not essential for the service processing and deployment. For example, the Authentication Center (AC) is an entity that manages the authentication information. The Short Message Entity (SME) composes and decomposes short messages, and Equipment Identity Register (EIR) is an area for further study, etc.

5.2 Data Type of the specification

In this section, we will introduce the ADT part of the specification. We use ACT ONE [GHM 78] as a notation for describing structured information. The information contains some data values intended to be conveyed across the communication medium. It also defines a collection of abstract equations that describe the meaning of the operations at a high level abstraction. Our design on abstract data types is based on the CNAP requirements and on the existing data parameters given in the standard's MSCs. The main data types in our design are listed as follows:

Component ID

This type defines the identification of network components. When many connections are concurrently established, component IDs are useful for recognizing different components so that they can be synchronized properly. An MS has two identifiers *MIN* and *MDN*, which have been explained in Chapter 2. All other components, such as MSC, VLR, HLR have their own distinguished IDs.

Address

This type contains the subscriber's identifications as well as static location information. The static location information includes the user registered HLR ID, and user registered MSC ID.

Name_Info

This type is used to define the set of displayed names of the calling users, i.e. the name information of those users who are involved in CNAP service. The Name_Info can also be "*restrict*" or "*not available*"

Subscribed Features

A subscriber can register for many WIN features. Subscribed Features can be *CNAP*, *RND*, and *CNAF*. They are grouped as a set, one for each subscriber. We can check if a user subscribe to certain feature by using a defined set membership operation.

Feature Status Set

Feature Status Set is a set of variable indicating feature activation status of *CNAP* and *RND*. There are six elements contained in the Feature Status set : *CNAP_Permanent*, *CNAP_On_Demand_Active*, *CNAP_On_Demand_InActive*, *RND_Permanent*, *RND_On_Demand_Active* and *RND_On_Demand_InActive*. We have one subset for each subscriber. An operation *IsActive()* is defined to check the activation of a feature.

MDN Status

MDNStatus is a major part of HLR and VLR profile item. It contains all fields mentioned above, such as Subscriber's Address, Subscriber's Name Information, Subscribed Features, Feature Status Set and some optional fields related to the Subscribed Features as shown in Figure 5-1. The optional fields, for example, can be the forwarded party's *ADDRESS* if the subscriber registers a call forward feature. Basically, except for *Feature Status Set*, data stored in *MDN Status* are permanent, and so will not be changed dynamically during the call.

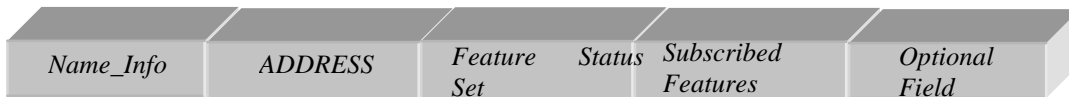


Figure 5-1 Data Structure of MDN Status

HLR ProfileItem

HLR ProfileItem shown in **Figure 5-2** is a data entry of the HLR Profile. In the HLR, there is one such record per user. It contains *MDNStatus* and the identifiers of the current serving VLR (*VLRID*) and serving MSC (*MSCID*) for this subscriber. The last two fields need to be changed depending on where the subscriber roams.



Figure 5-2 Data Structure of ProfileItem

The operations of *HLRProfileItem* type are defined in LOTOS as follows:

```
type TypeHLRProfileItem is TypeVLRID, TypeMDNStatus
  sorts HLRProfileItem
  opns
  NULL :-> HLRProfileItem (* constructor *)
  HLRProfileItem: MDNStatus, VLRID, MSCID -> HLRProfileItem (* constructor *)
  GetServVLR: HLRProfileItem -> VLRID
  GetServMSC: HLRProfileItem -> MSCID
  GetMDNStatus: HLRProfileItem -> MDNStatus
  LocationUpdate: VLRID, MSCID, HLRProfileItem -> HLRProfileItem
  AuthorizationUpdate: FeatureStatus, HLRProfileItem -> HLRProfileItem
...
endtype
```

Five operations are defined for the *HLR ProfileItem* record. The first three operations *GetServVLR* , *GetServVLR*, *GetServVLR* are used to extract the first, second and third component of the record. The fourth operation *LocationUpdate* is an operation that takes the new *VLRID* and *MSCID*, as well as the old *HLRProfileItem*, and yields the updated *HLRProfileItem*. In practice, the location update could be very complex and be implemented by the registration and handoff procedures within the mobile radio interface. However, we abstract from these details and only update the dynamic value in the *HLRProfileItem* record. The fifth operation *AuthorizationUpdate* is similar. It takes the new *FeatureStatus* and old *HLRProfileItem*, and yields the updated *HLRProfileItem*.

HLR Profile

The HLR Profile shown in Figure 5-3 is composed of a set of *HLRProfileItem*. In our specification there are two HLR profiles. A header *HLRID* is attached to each *HLRProfileItem*, indicating to which HLR profile the *HLRProfileItem* belongs. Although such information can be found in the *ADDRESS* field of *MDN Status*, using a header enables quicker searches of *HLRProfileItems*.



Figure 5-3 HLR profile data structure

The operations of *HLRProfile* type are defined in LOTOS as follows:

```
type TypeHLRProfile is TypeHLRProfileItem
  sorts HLRProfile
  opns
  {}:-> HLRProfile (* constructor *)
  InsertHPItem:HLRID,HLRProfileItem,HLRProfile -> HLRProfile(*constructor *)
  GetHPItem: MDNID, HLRID, HLRProfile -> HLRProfileItem
  UpdateHPItem: MDNID, HLRID, HLRProfileItem HLRProfile -> HLRProfile
  eqns
  ...
endtype
```

Three operations are defined in this record. The first operation *InsertHPItem* is to insert a new *HLRProfileItem* associated with a *HLRID* header into a *HLRprofile*. The second operation *GetHPItem* is an extraction function which extracts from the database *HLRProfile* an *HLRProfileItem* corresponding to a given pair of *MDNID* and *HLRID*. The third operation *UpdateHPItem* takes a new *HLRProfileItem*, and updates from the database *HLRProfile* the old *HLRProfileItem* identified by a given pair of *MDNID* and *HLRID*.

VLR Profile

VLR Profile is a database which maintains the visiting mobile subscriber's information. The information of an MS is transferred from the HLR to VLR. Therefore, the data of an MS stored in two databases are almost identical. Once the information is stored in the VLR database, the serving MSC can easily manage the roaming MS by getting relevant information from its local VLR database. Since the data component *VLRProfileItem* of *VLR Profile* is almost the same as the *HLRProfileItem* of *HLR Profile*, we omit the detailed explanation here.

5.3 Control Part of the specification

In this section, we describe the control structure of the CNAP service and the processes of which the specification is composed. In order to achieve a clear and readable specification, we adopt a top-down approach. We firstly describe the highest level

processes that compose the system, and then we decompose the process into lower detail. Figure 5-4 shows the top level of the LOTOS specification:

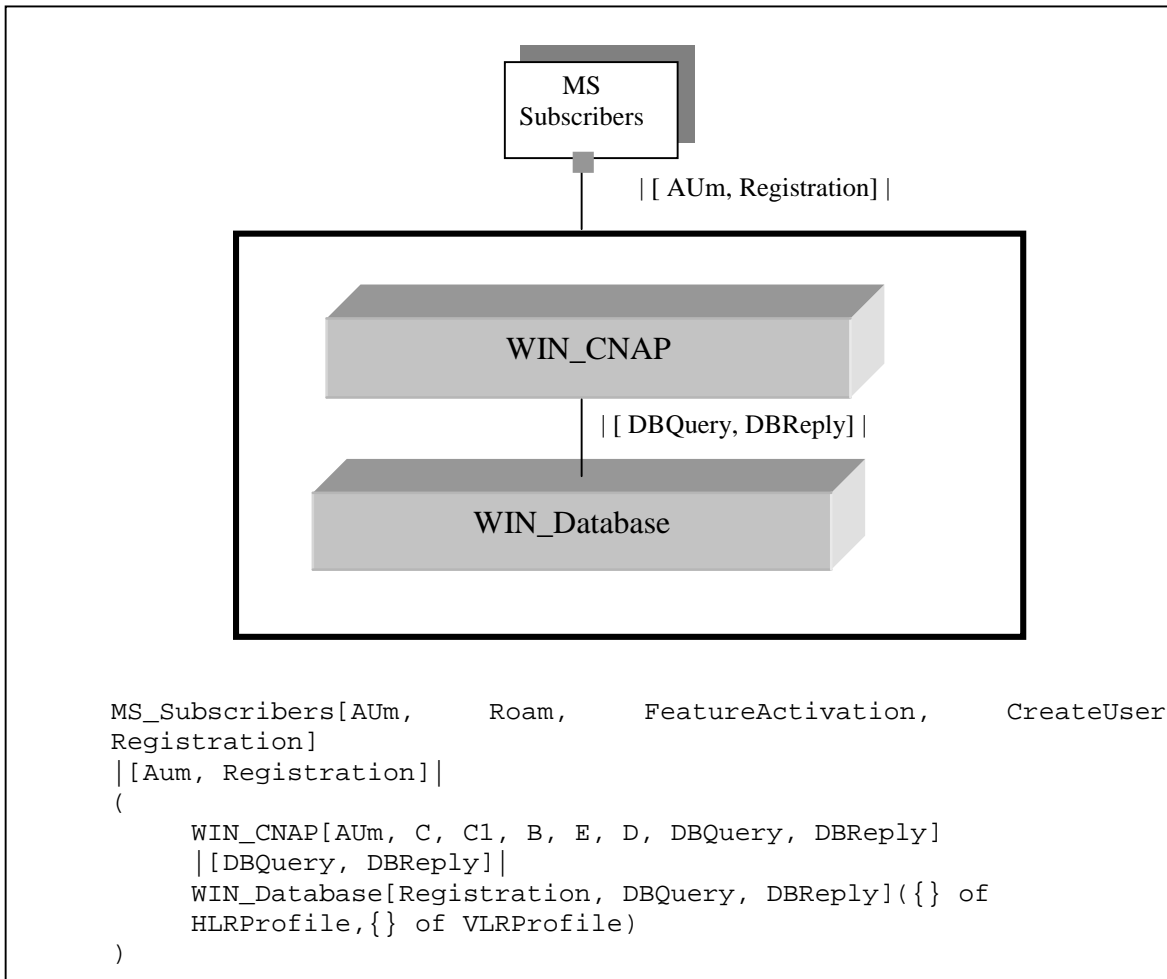


Figure 5-4 Top Level of the Specification

At the highest level, the core system is composed of two processes: process *WIN_CNAP* and process *WIN_Database*. The process *WIN_CNAP* specifies CNAP service provision for the network subscribers. All network activities required for service provision are included in this process. The process *WIN_CNAP* only exposes its gate *AUm* to the subscribers. Other internal gates, such as *C*, *C1*, *B*, *E*, *D*, *DBquery*, *DBreply*, are hidden from the subscribers. The other process *WIN_Database* manages the data functions such as storage, retrieval and synchronization for HLR and VLR databases. The gate *Registration* is visible to the subscribers, while gates *DBquery* and *DBreply* are invisible

to the subscribers. They are used for internal communication between the process *WIN_CNAP* and the process *WIN_Database*.

The process *MS_Subscribers* represents subscribers of the WIN network. The Subscribers can perform their actions through the system's external gates: Aum, Roam, FeatureActivation, CreateUser, and Registration.

5.3.1 Process *MS_Subscribers*

In order to analyze the CNAP feature, it is convenient for us to be able to create as many subscribers as we want. By using recursive instantiation, it is possible to create an unlimited number of subscribers. However, when we use recursion in a LOTOS specification, we need to be aware of the state explosion problem. We have two ways to create infinite number of users. The first way causes immediate state explosion problem, but the second one does not. In the first way, *MS_Subscribers* process is specified as follows:

$$P(\text{MS_Subscribers}) := \text{CreateUser}; \text{stop} \parallel P(\text{MS_Subscribers})$$

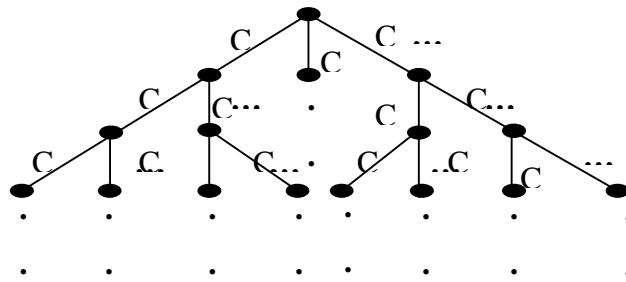
$$= P(\text{MS_Subscribers}) := \text{CreateUser}; \text{stop} \parallel \text{CreateUser}; \text{stop} \parallel \dots$$


Figure 5-5. LTS of Process *MS_Subscriber* (Infinite number of branch)

Figure 5-5 shows the resulting behavior tree where C is used as an abbreviation for action *CreateUser*. As shown in Figure 5-5, this use of recursion unfortunately leads to infinite branches once the process starts and thus results in illegal termination due to memory overflow.

However, this specific pitfall can be avoided by using “guard actions” on the recursion. The guard actions can reduce the infinite branches and make the recursion executable by the existing tools. In the second way of specifying *MS_Subscriber* process, at least one action (*CreateUser*) has to be executed before the recursion. The improved specification is shown as follows.

$P(\text{MS_Subscribers}) := \text{CreateUser}; (\text{stop} \parallel P(\text{MS_Subscribers}))$

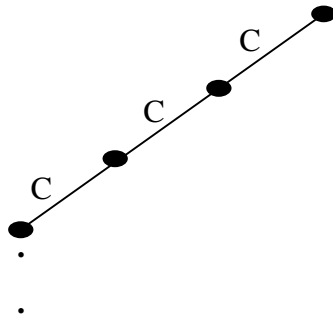


Figure 5-6. LTS of Process MS_Subscriber (one branch)

Figure 5-6 shows the improved resulting behavior tree, where C is used as an abbreviation for the action *CreateUser*. In our specification, we adopt the second way of specifying process *MS_Subscriber* in order to avoid this type of state explosion. The LOTOS specification of this process is described as follows:

```

process MS_Subscribers [AUm, Roam, Feature_Activation, CreateUser,
Registration]:exit :=
CreateUser ? h_id: HLRID ? HPItem:HLRProfileItem;
(
    Registration !Adduser !h_id !HPItem
    MS[...](...);
    |||
    MS_Subscribers[AUm, CreateUser]
)
endproc (* MS_Subscribers *)

```

In the *MS_Subscribers* process, the action *CreateUser ? h_id : HLRID ? HPItem : HLRProfileItem* is used to create a new subscriber for the system. *h_id* is a variable for HLRID header, whereas, *HPItem* is a variable for HLRProfileItem. The symbol ‘?’

means that the value of the variable needs to be received from the environment. The value of the input item must follow the ADT type definition described in section 5.2. Once the values of h_id and $HPItem$ are entered, the action `Registration` synchronizes with `WIN_Database` process in order to add the user's information into the user's home HLR database. After the `Registration` is successful, a process MS is created for representing the new subscriber. To limit the scope of the WIN system, we only generate three MS processes for our validation purpose.

5.3.1.1 Process MS

The LOTOS specification of process MS is shown below:

```

process MS [AUm, Registration] (id:CNI): exit:=
(
  (
    ( originator [AUm](id) [] terminator [Aum](id) )
    |||
    Roam ? servMSC : MSCID ? servVLR : VLRID;
    ...; exit
  )
  []
  Activation ? fs : FeatureStatus !id;
  ...; exit
)
...
endproc (* MS *)

```

An MS process can either initiate a call or terminate a call. Therefore, we firstly specify the process MS as a choice of two sub processes *Originator* and *Terminator*. Process *Originator* deals with the call processing on behalf of a calling party, while process *Terminator* handles the call processing on behalf of a called party. The details of each process will be explained in the next sections.

When we recall the *Terminator/Originator* UCMs in Chapter 4, we know that either the originator or the terminator may roam to another area. The roaming action is irrelevant to the call processing action and so can happen at any time. By using the LOTOS parallel operator “|||”, we can express such independent relation between call processing and

roaming. If the action *Roam* is performed, *servMSC* id and *servVLR* id in the new area are required to be provided by the environment.

When the user is not in the call processing, s/he may activate or de-activate the feature status in order to allow or deny the CNAP/RND service. The feature activation or deactivations cannot be done during the call, thus the LOTOS choice operator “[]” is used to describe the exclusive relation between the call processing and feature activation. The type of feature status (*fs*) can be entered through action *Activation*. There are six types of feature status, as mentioned: *CNAP_Permanent*, *CNAP_On_Demand_Active*, *CNAP_On_Demand_InActive*, *RND_Permanent*, *RND_On_Demand_Active*, *RND_On_Demand_InActive*. The input data updates the corresponding user’s record in the WIN_database.

5.3.1.2 Process Originator

The LOTOS specification of process *Originator* is shown as follows:

```

process Originator [AUm](id:CNI):exit:=

AUm ! CallOrigination ? dest : ADDRESS ! id ! GetMIN(GetADDRESS(id)) !
GetMSC(GetADDRESS(id));
(
  AUm ? a: ACTION ! GetMSC(GetADDRESS(id)) ! GetMIN(GetADDRESS(id));
  exit
  []
  AUm ! BusyTone; exit
)
endproc (* originator *)

```

In the *Originator* process, a user can originate a call. This is described by “*AUm ! CallOrigination ? dest : ADDRESS ! id ! GetMIN(GetADDRESS(id)) ! GetMSC(GetADDRESS(id))*”. This *CallOrigination* action is executed at gate *AUm*. The symbol of “? dest” represents that the destination number needs to be entered. Once the user enters the destination number, the number as well as other information (*GetMIN(GetADDRESS (id))*) are sent to the Originating MSC whose id is extracted from *GetMSC(GetADDRESS(id))*. In the distributed environment, many *CallOrigination* actions may be performed simultaneously by different users and so each action needs to be distinguished by a user’s identifier (! *id*).

After the *CallOrigination* action is successfully performed, the user waits for the response from the called party. *AUm ? a: ACTION* is a place to wait for the response. The response from the called phone might be either an answer or no answer. Another possibility is that the called phone is busy, and for this, there is an event “*BusyTone*”.

5.3.1.3 Process Terminator

The LOTOS specification of process Terminator is briefly shown as follows:

```
process Terminator [AUm] (id:CNI):exit:=  
AUm ?a: ACTION_List ...;  
(  
  AUm !CallAnswer ...;  
  exit  
  []  
  AUm !CallNoAnswer...;  
  exit  
)  
endproc (* Responder *)
```

Process *Terminator* specifies the behavior of the WIN subscriber who responds to a call. In the *Terminator* process, the subscriber gets event (*?a:ACTION_List*) on gate *AUm* from the WIN network. Such event can be *alert with call name display* or *alert without call name display*. Once the event has happened, the terminator can choose to answer the phone or not.

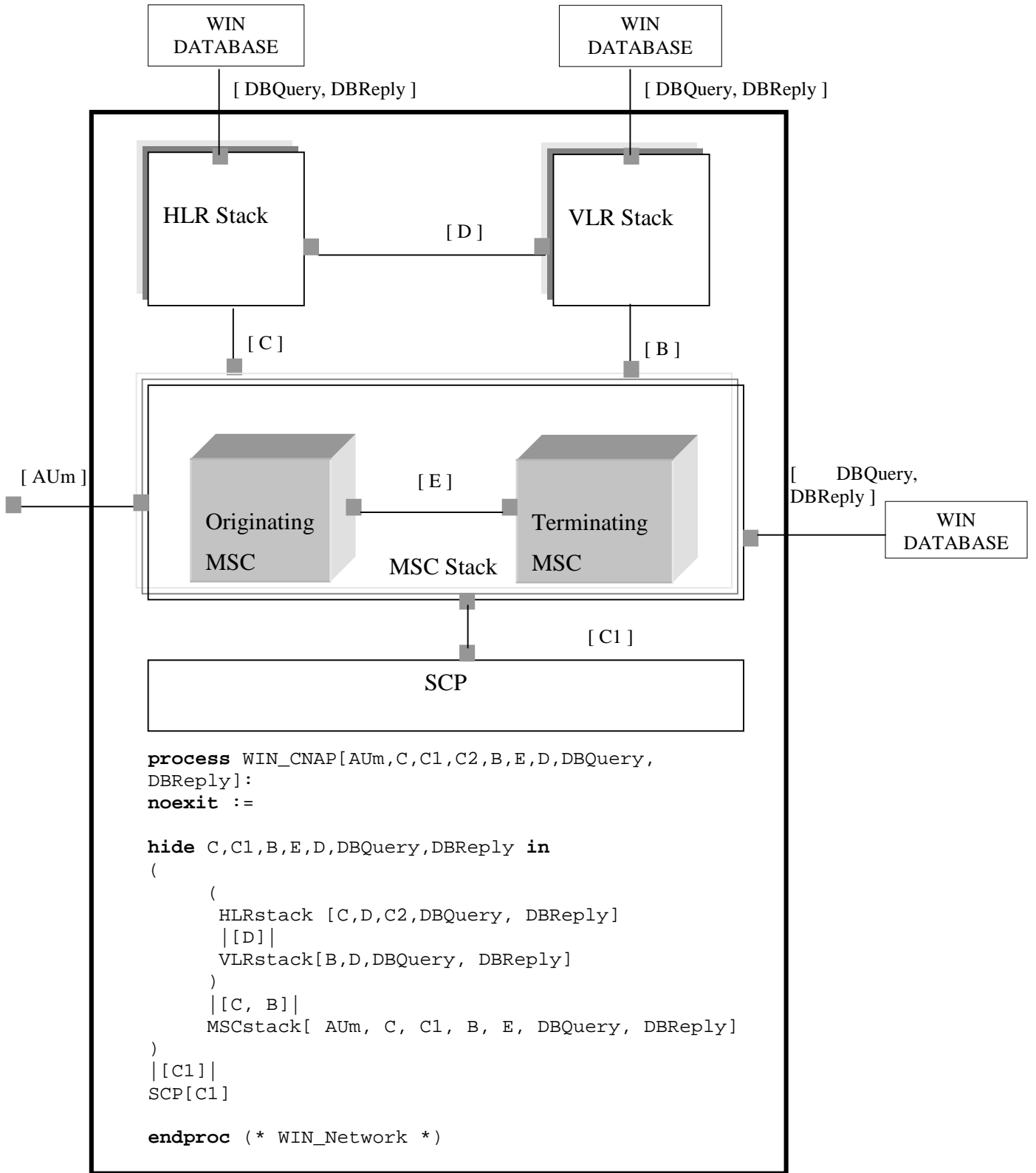


Figure 5-7 WIN_CNAP Specification

5.3.2 Process WIN_CNAP

WIN_CNAP is a large and important process in our specification. **Figure 5-7** shows the top-level LOTOS specification of the *WIN_CNAP* process schematically. The *WIN_CNAP* process is composed of four sub-processes. Each of the sub-processes is listed as follows:

- Process *HLRstack* includes two HLRs composed by a parallel operator “|||”. It communicates with other processes through gate *C*, *D*, *DBQuery*, and *DBReply*.
- Similarly, Process *VLRstack* contains two VLR processes which communicate with other processes through gate *B*, *D*, *DBQuery*, and *DBReply*.
- Process *MSCstack* specifies the behavior of two MSCs. Each can act either as an originating MSC or a terminating MSC for a certain user. MSCs can communicate with each other through gate *E*. Each MSC provides an interface (gate *AUm*) to the mobile user. The MSC can also communicate to other processes through gates *C*, *B*, *C1*, *DBQuery*, *DBReply*.
- Process *SCP* gets the service request from an MSC and responds through gate *C1*.

The specification details related to each sub-process will be discussed in the next sections.

5.3.2.1 Process MSCstack

Process *MSCstack* is a composition of two MSC processes synchronizing through gate *E*. According to the function role, an MSC process can be either an originating MSC or a terminating MSC.

Originating MSC

In the bound UCM of *WIN_CNAP* stub described in Chapter 4, Originating MSC contains one function entity *SSF_CCF*. However, in the Originating MSC process, we specify only a part of the behavior of *SSF_CCF*, the one that relates to *CNAP*. Therefore, to make it clear that the *SSF_CCF* is not the whole *SSF_CCF* function, we use

a different name *SSF_CCF_1*. The LOTOS specification of Originating MSC process is shown below:

```

process OrigMSC[AUm,C,C1,B,E,DBQuery,DBReply](id:MSCID) : exit :=
AUm ! CallOrigination ? DGTSDIAL:ADDRESS ? CNIdigitsBCD:CNI ! id;
(
  MSC[AUm,C,C1,B,E,DBQuery,DBReply,Probe](id)
  |||
  (
    SSF_CCF_1[AUm,C,C1,B,E,DBQuery,DBReply]
    (id, DGTSDIAL, CNIdigitsBCD, insert(GetNAME(CNIdigitsBCD),{}))
    []
    i;
    SSF_CCF_1[AUm,C,C1,B,E,DBQuery,DBReply]
    (id, DGTSDIAL, CNIdigitsBCD, insert(Not_Avail,{})) )
  )
)
endproc (* OrigMSC *)

```

In the *OrigMSC* process, we instantiate a new MSC process each time the *CallOrigination* request is received. The new instantiated MSC process is used to accept other call requests while the current call is processing. This strategy is not only applied to the *OrigMSC* process, but also to many other processes which we will discuss later.

To process the current call, *SSF_CCF_1* is initiated. Note that there are two possible ways of calling *SSF_CCF_1*, which are placed in alternative operator ([]) in the specification. Their parameter list in the first alternative corresponds to the case where the name of the user can be displayed, and the appropriate name is inserted into the name list. The second alternative corresponds to the case where the name is not available, and so *Not_Avail* is generated and inserted into the name list. The *SSF_CCF_1* process is relatively complex, we only show its skeleton structure.

```

process SSF_CCF_1[AUm,C,C1,B,E,DBQuery, DBReply]
(id:MSCID, DGTSDIAL:ADDRESS, CNIdigitsBCD:CNI, NAME:NAME_List) :eixt :=
C ! LocReq ! DGTSDIAL ! CNIdigitsBCD ! NAME ! CalleeHLR_id !id;
C ! ReLocReq ? TERMLIST : TERMLIST_INFO ! CalleeHLR_id ! id;
...
[servMSC_id eq id] ->
( C1 !Favail ! ...

```



```

        C1 !ReFavail ? ACTLIST:ACTION_List ? DISPLEXT:NAME_List ...
        ...
    )
[servMSC_id ne id] ->
( ... )
endproc (* SSF_CCF_1 *)

```

In the *SSF_CCF_1* process, a *LocReq* is sent to the HLR process. Some parameters are included in the request, such as called phone number (*DGTSDIAL*), calling name information (*CNIdigitsBCD*), etc. As a response, the HLR returns the routing information in the format of *TERMLIST*, which implies the location of the terminating MSC.

It is also possible that the originating MSC needs to act as a terminating MSC since the terminator could be in the same area as the initiator's. This case is indicated by the fact that the premise “[servMSC_id eq id]” is true, and in this case the service request (!Favail ...) is invoked on gate C1, and the action list (!ReFavail ? ACTLIST:ACTION_List ...) is returned on the same gate.

Terminating MSC

The terminating MSC process contains two function entities: MACF and SSF_CCF. For reason similar to those we explained in the previous section, we adopt a new name *SSF_CCF_2* to distinguish this process from the whole *SSF_CCF* function. The LOTOS specification of the terminating MSC process is shown below:

```

process TermiMSC[AUm,C,C1,B,E, DBQuery, DBReply](id:MSCID) :exit:=
B ! RouteReq ? x:MINID ? y:MDNID ? z:CNI ? u:NAME_List ? VLR_id:VLRID ! id;
(
    MSC[AUm, C, C1, B, E, DBQuery, DBReply](id)
    |||
    (
        MACF[B,DBQuery, DBReply](x, y, VLR_id, id)
        >>
        accept busy: Bool in
        [Not(busy)] ->
            SSF_CCF_2[AUm, C, C1, B, E, DBQuery, DBReply]
    )
)

```

```

                (id,y,u,VLR_id)
            [busy] -> exit
        )
    )
    exit
endproc (* TermiMSC *)

```

TermiMSC process firstly synchronizes on the *RouteReq* action with the VLR process at gate B, where the received value includes the terminator’s MS identifier (x), the terminator’s identifier (y), the originator’s calling name information (z), and the contacting VLR identifier *VLR_id*. Once the *TermiMSC* process receives the *RouteReq*, the *MACF* function determines whether the terminator’s MS is busy or not. This is done by querying of the VLR database. If it is busy, the busy status is returned in the *ReRouteReq*. Otherwise, the *MACF* function assigns the routing information and returns it in the *ReRouteReq*. According to the standard, the route information is represented in the format of TLDN. *Channel* (*id*, *x*) is a resource allocation function. It assigns a radio channel for the terminator’s MS. The *MACF* function is described by the following process:

```

process MACF[B, DBQuery, DBReply]( x:MINID, y:MDNID, VLR_id:VLRID,
id:MSCID) :exit :=

DBQuery ! Query_VLR_Profile ! VLR_id ! y;
(* query the VLR database for user’s busy status *)

DBReply ? value:VLRProfileItem ... ;
(
    let busy:Bool = IsBusy (value) in
    [busy] -> B ! ReRouteReq ! TLDN(NULL, busy); exit (busy)
    []
    [Not(busy)] -> B ! ReRouteReq ! TLDN(channel(id, x), idle) ! VLR_id !id;
        exit (busy)
)
endproc (* MACF *)

```

The *exit* action of the *MACF* process passes the Boolean value “busy” to the successor behaviors. This Boolean value indicates whether the terminator’s MS is busy or not. The process *SSF_CCF_2* is instantiated only if the terminator’s MS is currently idle. In the

SSF_CCF_2 process, action *Callsetup* synchronizes with the originating MSC process on gate E. After the call is setup, a request of updating the terminator's current state as "busy" and a query of the terminator's registered WIN feature is sent to the Database Management by action *DBQuery*. The terminator's registered WIN feature is received as an answer by action *DBReply*. The subscribed WIN features will be invoked by action *Favail* on gate *C1* and the returned action list (*ACTLIST*) is provided by action *ReFavail* on the same gate.

The *SSF_CCF_2* process provides the necessary alert actions to the terminator MS, e.g. alert the with the calling name information *DISPTEXT* (*AUm ! ACTLIST !DISPTEXT...*). From the terminator, there are two possible responses: "call answer" or "call no answer". In the case of *Call Answer*, the originating MSC will synchronize the terminator's *call answer* action through gate E and then exists. In the other case of *Call No Answer*, the originating MSC may synchronize on *call no answer* action or may redirect the call if the terminator has the CNAF feature. To redirect a call, *SSF_CCF_2* needs to send redirect request and to release the current call link. The skeleton LOTOS structure of process *SSF_CCF_2* is specified as follows:

```

process SSF_CCF_2[AUm,C1,E,DBQuery, DBReply]
  (id:MSCID, y:MDNID, u:NAME_List, VLR_id:VLRID) : exit:=

E ! CallSetup ...;

DBQuery ! Update_VLR_Profile_Busy ! VLR_id !y;
(* a request of updating the terminator's current state as "busy" *)

DBQuery ! Query_VLR_Profile ...;
DBReply ? value:VLRProfileItem ...;
(* a query of the terminator's registered WIN feature *)

C1 ! Favail ...
C1 ! ReFavail ...

AUm ! ACTLIST !DISPTEXT ... ;
AUm ? a: ACTION ...;
(
  [a eq CallAnswer] -> E ! CallAnswer ...; exit
  []
  [a eq CallNoAnswer] ->
  (
    [eleof(CNAF,GetSubscribedFeatures(GetMDNStatus(value)))eq false]->
    E ! CallNoAnswer ...; exit
    []
  )
)

```

```

        [eleof(CNAF,GetSubscribedFeatures(GetMDNStatus(value))) ]->
        E ! Redirection ...;
        E ! Release ...; exit
    )
)
endproc (* SSF_CCF_2 *)

```

5.3.2.2 Process HLRstack

The process *HLRstack* contains two HLR processes composed by an interleave operator “|||”. In the HLR process, the HLR receives the location request (*LocReq*) sent by the originating MSC on gate *C*, and sends the route request (*RouteReq*) to the VLR process on gate *D*.

Another alternative behavior of the HLR is to reply to the query from the originating MSC about the redirected calling number if the call needs to be redirected. In order to retrieve such information from HLR database, *DBQuery* action is sent to the Database Management process after *TrigNoAnswer* action occurs on gate *C*. And the queried information is returned as a *value* parameter of action *DBReply*. The skeleton LOTOS specification of HLR process is shown as follows:

```

process HLR [C,D,DBQuery,DBReply] (id:HLRID) :noexit :=
C !LocReq ...; (* receives the location request *)
(
    HLR[C,D,DBQuery,DBReply,Probe](id)
    |||
    ...
    D !RouteReq ...;
    ...
) (* C! Loc *)
[]
C !TrigNoAnswer ...; (* receives the query of forwarded call number *)
(
    HLR[C,D,DBQuery,DBReply] (id)
    |||
    (
        DBQuery ! Query_HLR_Profile ! MDN;
        DBQuery ? value :HLRProfileItem;
        ...
        C !ReNoAnswer ...; (* replies to the query of forwarded call number *)
        ...
    ) (* ||| *)
) (* C !Trig *)

```

```
endproc (* HLR *)
```

5.3.2.3 Process VLRstack

Similarly, the process *VLRstack* has two sub process composed by an interleave operator “|||”. According to the scenario described in Chapter 4, *VLR* tracks the states of all MSs in its area. In our specification, *VLR* receives the route request (*RouteReq*) from the HLR through gate *D*. But before it forwards the request to the Terminating MSC, it needs to find out the *terminating MSC id* by sending the *DBQuery* to the *VLR* database. The queried information is returned as a *value* parameter of action *DBReply* and the *terminating MSC id* can be retrieved by an ADT operation *GetMSC(value)*. After the Terminating MSC receives the forwarded route request on gate B, it will eventually return the route information *TLDN* to the *VLR* process by synchronizing action *ReRouteReq* on gate B. Furthermore, this information is returned to the HLR process by synchronizing action *ReRouteReq* on gate D. The skeleton LOTOS specification of process *VLR* is shown as follows:

```
process VLR [B,D,DBQuery,DBReply] (id:VLRID) :noexit :=
D !RouteReq ...;
(
  VLR [B,D,DBQuery,DBReply](id)
  |||
  DBQuery ! Query_VLR_Profile !id ! MDN;
  DBReply ? value :VLRProfileItem ! MDN;

  B !RouteReq ... ! GetMSC(value);
  B !ReRouteReq ? TLDN:TLDN_INFO !id !GetMSC(value);

  D ! ReRouteReq ! TLDN ! HLR_id ! id;
)
endproc (* VLR *)
```

5.3.2.4 Process SCPstack

Process *SCP* contains SCF function. The SCF process is shown below:

```
process SCF[C1,Probe]:exit :=
```

```

hide Timeout, Success in
C1! Favail ?fs :SubscribedFeatures ? FStatus : FeatureStatusSet...;
(
  SCP[C1,Probe]
  |||
  [eleof(CNAP,fs) and ActiveCNAP(FStatus)] ->
  (
    Timeout;
    ...
    []
    Success;
    (
      [eleof(RND,fs) and ActiveRND(FStatus)] ->
        (* the terminator has RND service active)
        C1 !ReFavail !insert( Alert_with_CNI, {})! nl ! ...;
      ...
      []
      [eleof(RND,fs) eq false]->
        (* the terminator does not have RND service *)
        C1 !ReFavail !insert( Alert_with_CNI, {})! GetOrig(nl) ! ...;
      ...
    ) (* Succ *)
  ) (* [ele *)
) (* C1 *)

endproc (* SCF *)

```

When a service request (*Favail*) is received on gate C1, SCF firstly checks the premise *[eleof (<CNAP> <feature set>)]*, which indicates whether the terminator has a *CNAP* service in his/her subscribed *feature set*. If the terminator has a *CNAP* feature, there are two possible service provision results given by the SCF. The action *Timeout* indicates an unsuccessful service provision result due to some unpredictable system errors. The other action *Success* represents the normal service provision. In this case, the SCF checks if the terminator's *CNAP* feature contains *RND* option and if the *RND* is active. Based on the result, the response (*ReFavail*) is replied at gate *CI* with an appropriate action list and a displayed name list. The action list will be received by process *terminating MSC*. It is represented in our specification as an *insert (...)* operation. The displayed name list can be either (a) the originator and the last forwarded party's calling name (*nl*), or (b) the originator's calling name (*GetOrig(nl)*).

5.3.3 Process WIN_Database

As shown in Figure 5-8, the process *Database_Management* manages HLR and VLR data profile. This process communicates with the *WIN_CNAP* process through gate *DBQuery*, *DBReply*, and with the *MS Subscribers* process through gate *Registration*.

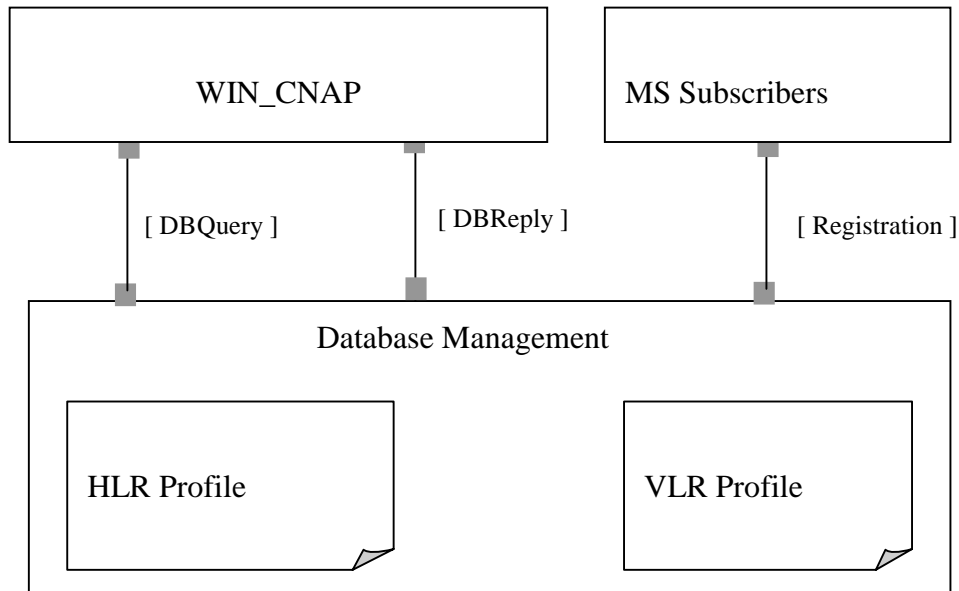


Figure 5-8 WIN Database Management Process

The brief LOTOS specification of this process is shown as follows:

```

process Database_Management[Registration, DBQuery, DBReply](hs:
HLRProfile, vs:VLRProfile):noexit :=

DBQuery ! Query_HLR_Profile ? h_id:HLRID ? ms_id:MDNID;
DBReply ! GetHPItem ( ms_id, h_id, hs) ! ms_id;
Database_Management [Registration,DBQuery, DBReply](hs, vs)
[]
DBQuery ! Query_VLR_Profile ? v_id:VLRID ? ms_id:MDNID;
DBReply ! GetVPItem ( ms_id, v_id, vs) !ms_id;
Database_Management [Registration,DBQuery, DBReply](hs, vs)
[]
DBQuery ! Update_VLR_Profile_Busy ? v_id:VLRID ?ms_id:MDNID;
...
Database_Management [Registration,DBQuery, DBReply](hs, vs)
[]
Registration ! AddUser ? h_id :HLRID ? value : HLRProfileItem;
...
Database_Management[Registration, DBQuery, DBReply](hs,vs)
[]
  
```

```

Registration ! Handover ? servMSC:MSCID ? servVLR:VLRID ?...;
...
Database_Management[Registration, DBQuery, DBReply](hs, vs)
[]
Registration ! Activation_Update ? fs :FeatureStatus ? id:CNI;
...
Database_Management[Registration, DBQuery, DBReply](hs, vs)

endproc (* Database_Manager *)

```

We allow concurrent access to the database through gate *DBQuery* and *DBReply* at any time. However, the responses for these requests are sequentially processed by using the LOTOS choice operator '[]', and so the consistency of data can be protected. There are three actions on gate *DBQuery*. Action *Query_HLR_Profile* requests a user's HLR record corresponding to a given pair of *h_id* and *ms_id*. The user's HLR record can be retrieved by an extraction function *GetHPIItem* (*ms_id*, *h_id*, *hs*) and the extracted value is returned on gate *DBReply*. Similarly, action *Query_VLR_Profile* is to query user's VLR record and the extracted value is returned on gate *DBReply*. The third action *Update_VLR_Profile_Busy* marks a busy status for a given user (identified by *ms_id*) in a given VLR database (identified by *v_id*).

On gate *Registration*, there are three actions. The first action *addUser* is used for creating a new subscriber's record in the HLR database. At the beginning, there is no subscriber in the system, therefore the parameters (*hs*, *vs*) of this process are initialized as empty. Later on, when the process *MS_Subscribers* generates a new subscriber, it sends *addUser* action request to the *database_Management* process on gate *registration*. As a result, this subscriber's HLR profileitem record (*h_id*, *value*) is generated and inserted into the HLR profile (*hs*). The content of this record is copied into the subscriber's serving VLR profile (*vs*). The data stored in the VLR database is only meaningful as the MS is registered in this VLR area. If a subscriber's MS roamed to another area, the user's record in the old VLR database needs to be deleted and the new entry needs to be created in the new VLR database. The second action *Handover* represents such case. When a subscriber roams to a new area, the *Handover* request sends to the *Database_Management* process, and so the subscriber's old VLR profileitem record is deleted and a new record is created in the

current serving VLR profile (*vs*). Furthermore, the subscriber's current serving VLR identifier is recorded in the user's HLR database. The third action *Activation_Update* is to handle the feature activation update request from a subscriber. Given by the subscriber's identifier (*id*) and a new feature activation status (*fs*), the *Database_Management* process updates the subscriber's corresponding feature activation field in his/her HLR and VLR profileitem record.

5.4 Conclusion

In this chapter, we present a formal LOTOS specification of the CNAP service. We use ACT ONE [GHM 78] as a notation for describing structured information and describe the control structure of the CNAP service and the processes of which the specification is composed. Using this specification prototype, we can apply validation technique described in the next chapter.

Chapter 6 Validation of CNAP

6.1 Overview of the validation method

Given a formal specification representing a design in LOTOS, we can apply many validation techniques [Bri88][CKM93][AmLo] to establish its correctness (i.e. absence of logical error) or to find ambiguities in the requirements. Many tools, such as LOLA and ELUDO, exist in this aspect. In this thesis, we use a number of validation techniques that include simulation, probe testing and graphic MSC generation. During this process, some ambiguities of CNAP scenarios from the WIN pre-balloting draft are discovered.

6.2 Tool support

In this section, we are going to introduce some software tools for the capture, the editing, the maintenance and the testing of descriptions. Figure 6-1 shows the tools and internal files used by the UCM scenario analysis, LOTOS specification and validation procedure.

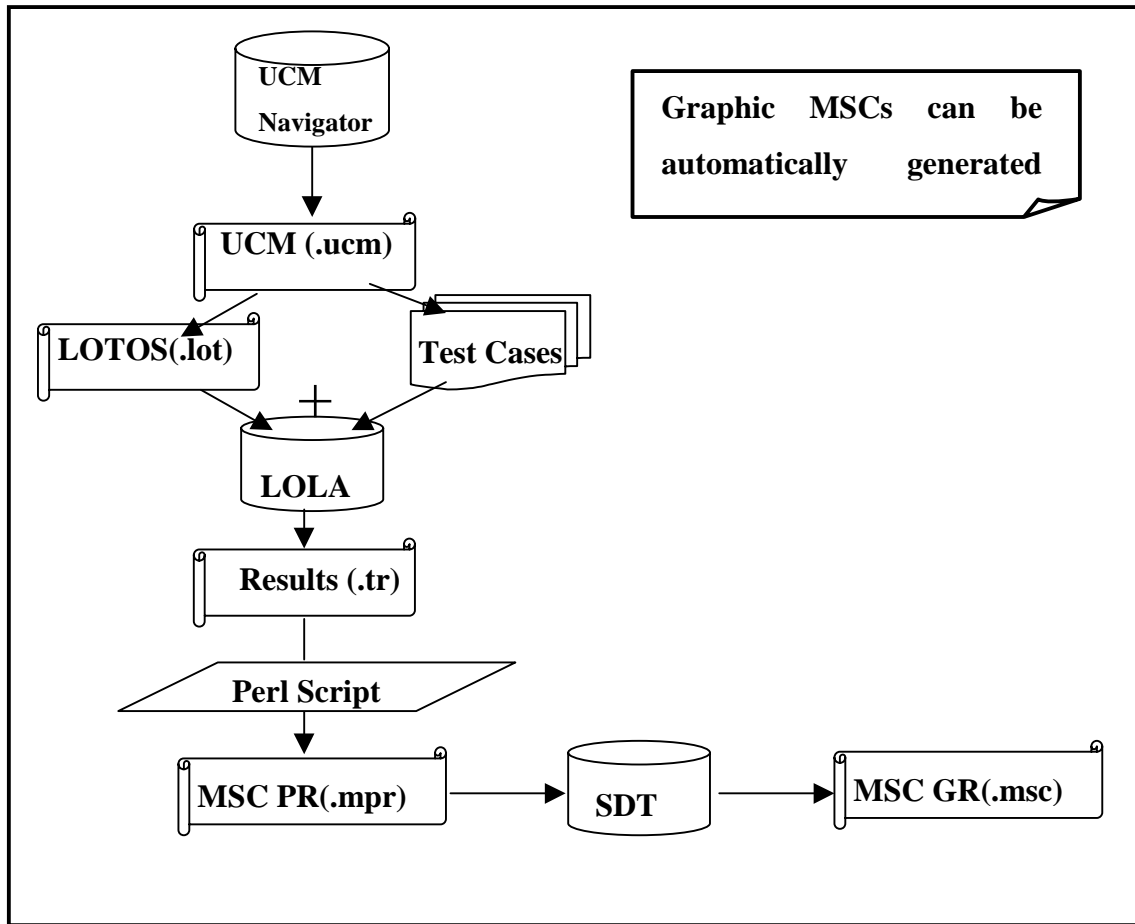


Figure 6-1 Supporting Tools and Files

- UCM Navigator is a UCM editor. It is under development at Carleton University. Versions are currently available for many platforms (Sparc/Solaris, HP/Unix, i386/Linux). As a tool for creating and modifying UCMs, it is designed to handle any unbound or bound UCM. It is also capable of creating multi-level maps in which sub-maps of a lower level are expressed as stubs in a higher-level map. Using this tool, we get the informal visual notation scenario saved in the format of a file (.ucm) for the WIN system.
- LOLA is a step-by-step executor, a tool for obtaining a state transition system of a LOTOS specification and a tool for testing[PL91]{PLR95}. It was developed at the

Technical University of Madrid. The LOLA tool can be utilized for automating a number of validation techniques such as step-by step simulation, testing, etc. Test cases from the specification are formalized as LOTOS test processes. The tested result can be saved as a file (.tr) which contains traces leading to a success event or a deadlock.

- Perl is a popular script language, which has powerful text-manipulation functions. In our case, we use perl to transform a format of the test result (.tr) into plain representation MSC format (.mpr).
- SDT is a well-known tool package produced by Telelogic. It is used for processing specifications written in SDL. SDT has an MSC editor to provide support for editing, managing and generating descriptions of MSCs in both plain representation format (.mpr) and graphic representation format (.msc). The plain MSC representation format (.mpr) can be taken by a SDT MSC editor in order to generate a graphic MSC (.msc).

6.3 Simulation, Testing and New Scenario Generation

6.3.1 Simulation

We use step-by-step execution to simulate possible events in the current environment. In the process of simulation, at every step, a user who plays the role of the environment selects an event among a set of events allowed by the system. According to the given response, the system computes the new state, and provides the successor events for the next selection. This process may continue until no further event is possible. The simulation makes the CNAP specification executable. It helps the designer to determine that the generated scenarios correspond to the requirements.

By doing simulation, the designer can investigate the behavior of the system scenarios on detailed scenarios that may be ignored in the requirements but can be discovered as problems in later design or implementation stages. LOLA is the tool that was used to

execute the *CNAP* specification. In this model of execution, the environment (or the specific user) may be required to supply values for variables. During the execution, the user may backtrack to any event in the trace and execute different branches, or the same branch with different values. Executed events and the order in which they occurred are recorded in *traces* by the system. A trace generated by LOLA simulation is shown in Figure 6-2. This trace shows a scenario where the system creates three new subscribers and registers these subscribers into the WIN Database. Actions 1, 2, and 3 create respectively user *amy*, *bill* and *claudio*, they assign the first two users to HLR1, and the third user to HLR2. All three users subscribe to the *CNAP* service with the *RND* option permanently enabled. *amy* and *claudio* forward their calls to *bill*, and *bill* forwards his call to *claudio*.

More in detail, let us see the first action in Figure 6-2. It is at gate '*createuser*' and the first element '*1 of hlrld*' indicates that the newly created user belongs to HLR1. The following experiment is the data subscription information of the user as explained in Section 5.2. In particular, the *hlrprofileitem* refers that the name of the new user is *amy*, that the MDN is *a*, the MIN is 1, the HLR is 1 and the MSC is also 1. The following two *insert* operations indicate the fact that the user subscribes to *CNAP* with *RND* option, and the status of these features is permanent. The address operation is the forwarded person's address, who is *bill* (*b*), with MIN2, HLR1 and MSC1. Finally, for user *amy*, serving MSC and VLR are 1. The following two actions are similar thus we omit the explanation. Moreover, three internal actions follow the trace to add the three created users to the HLR database. They are at gate '*i*' and are not controlled by the environment.

It can be easily seen that LOTOS traces are quite hard to read and hence the need to translate them into MSCs (see section 6.3.3)

```

(* Traces generated by step-by-step *)

[ 1] - createuser ! 1 of hlrid !
hlrprofileitem(mdnstatus(amy,address(a,1,1,1),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),address(b,2,1,1)),1,1);
[ 1] - createuser ! 1 of hlrid !
hlrprofileitem(mdnstatus(bill,address(b,2,1,1),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),address(c,3,2,2)),1,1);
[ 1] - createuser ! 2 of hlrid !
hlrprofileitem(mdnstatus(claudo,address(c,3,2,2),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),address(b,2,1,1)),2,2);
[ 1] - i; (* registration ! adduser ! 1 of hlrid !
hlrprofileitem(mdnstatus(amy,address(a,1,1,1),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),address(b,2,1,1)),1,1) *)
[ 4] - i; (* registration ! adduser ! 1 of hlrid !
hlrprofileitem(mdnstatus(bill,address(b,2,1,1),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),address(c,3,2,2)),1,1) *)
[ 6] - i; (* registration ! adduser ! 2 of hlrid !
hlrprofileitem(mdnstatus(claudo,address(c,3,2,2),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),address(b,2,1,1)),2,2) *)

```

Figure 6-2 Trace generated by Simulation

6.3.2 Specification Testing

Testing is most commonly defined as a process where an implementation is checked for conformance with respect to a specification. If we have an executable specification, a similar process can be used to check that the specification corresponds to requirements. It is possible to execute test scenarios step-by-step, by using the process described above, or it is possible to create a LOTOS specification including one or more scenarios and execute it in parallel with the specification as a “test process”. For each such execution, there are three possible responses: *must pass*, *may pass*, or *reject*.

MUST PASS:

All the possible executions were successful. They reached the *Success* event by all means.

MAY PASS:

Some executions were successful, some unsuccessful. This happens mostly because of non-deterministic choice existing in the system. Note that timeouts in our system are represented as non-deterministic choices.

REJECT:

All the executions failed. They all deadlocked somewhere in the system.

LOLA implements this testing mechanism. There is a *TestExpand* command that is used to synchronize test cases with a LOTOS specification:

TestExpand -l success <test_process> <-y> <-i> <-a> <-s>

The *TestExpand* command makes a complete state exploration and calculates the types of response. The test result gives us all the traces leading to a success and/or a deadlock. We can use the *Print* command to see these traces. The four options of this *TestExpand* command are listed below.

-a: Generate executions leading to success.

-s: Generate executions leading to stop.

-y: Force analysis of all possible executions in MAY tests.

-i: Redundant internal actions are NOT removed.

If there are too many traces, we might want to remove the -y option and the -a option in order to focus on traces that lead to a deadlock. After the execution of this test command, we have to reload the original specification.

In the next sections, we are going to talk about two issues: how to obtain functional test cases, and how to measure the LOTOS structure coverage based on the generated test cases.

6.3.2.1 Derivation of Test Cases from Unbound UCM

Test cases can be chosen based on the externally visible behavior or based on the internal structure of a specification. Synchronizing a system external behavior traces, or a specification including traces, with the system is a form of black-box testing. In our thesis, black box test suites are generated from the *Originator* and *Terminator* UCMs described in Section 4.2.1 in order to validate the specification against the functional requirements.

Scenario(s) on a single chosen path:

A UCM path might express several routes between a start point and an end point. We can choose any alternative route within a UCM to build our black-box test cases. In Figure 6-3, we show again two UCMs from Chapter 4, but only the paths used for generating our tests are shown as continuous lines. Note that the path shown in continuous black line in Figure 6-3 (a) is part of the path shown in continuous black line in Figure 6-3 (b). This latter path is also shown in continuous line in Figure 6-4. Similar remains for paths shown in gray lines. These paths are shown in isolation in Figure 6-4.

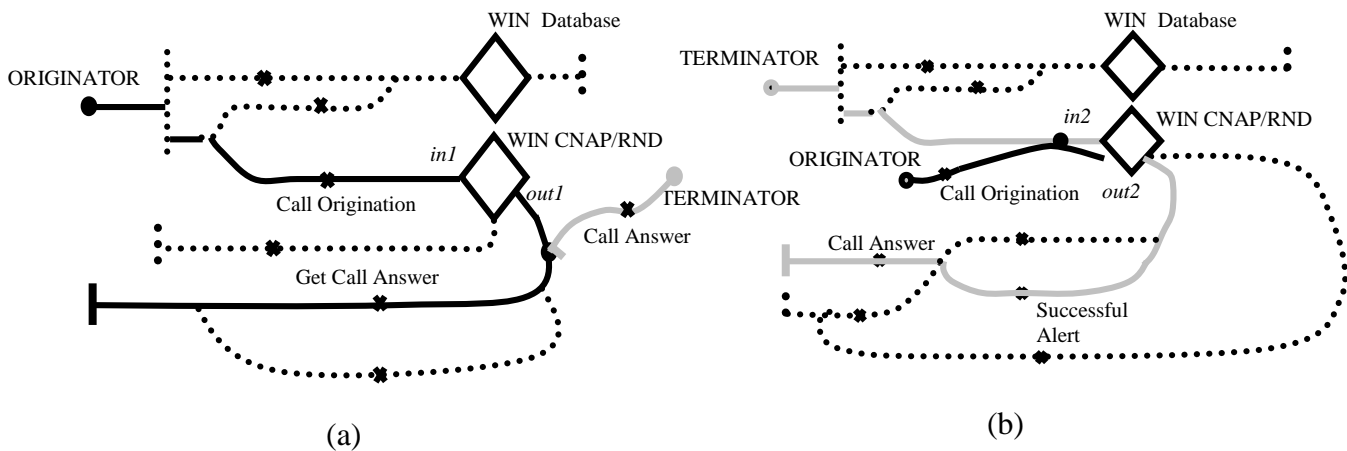



Figure 6-3 A Chosen Path from CNAP Service Description UCM

In Figure 6-4, the chosen path is symbolized by a  (not part of the map notation). Imagine putting such a symbol at the start of a path and then moving it along the path

from point to point until the end is reached and the symbol is removed. The path traced is a scenario. The interpretation of a path as a scenario is the link to test cases. A test case can get inspiration from the description of scenarios or related sets of them and can be formalized into a LOTOS test process.

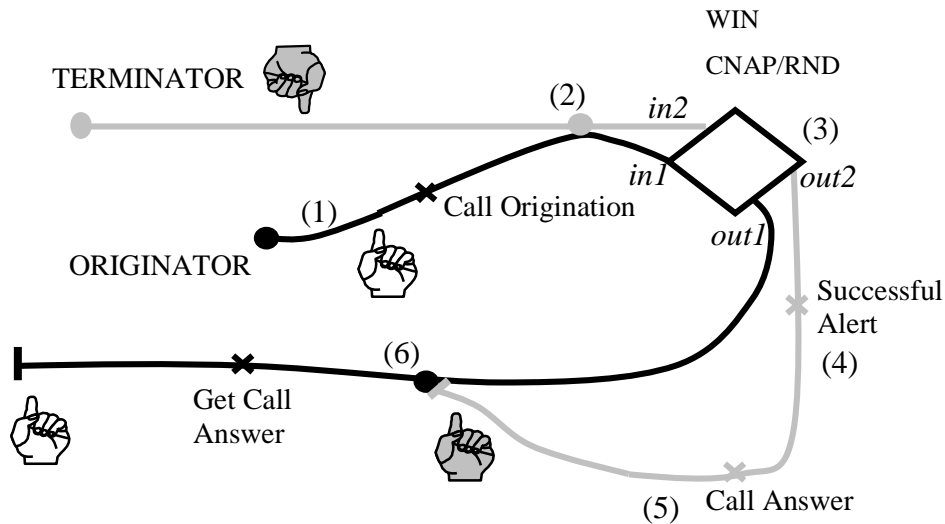


Figure 6-4 Single Scenario on Alternative Path

In Figure 6-4, we first create a user instance *A* on the originator's path and a user instance *B* on the terminator's path. The following scenario is then generated when the two instances move along the two paths:

- (1) Originator *A* calls terminator *B*;
- (2) This waiting point is needed because the terminator cannot be alerted before the originator establishes the call;
- (3) This is the stub shown in Figure 4-5;
- (4) Successful alert follows;
- (5) The terminator *B* answers the call;
- (6) The call proceeds;

Process *Test_1* below is a derived test case in LOTOS from the scenario mentioned above. In this process, action *CreateUser* generates two users *originator A* and *terminator B* for our validation purpose. According to the created *HLRProfileitem*, originator *A*'s name is *Amy*, whose address is *address(a,1,1,1)*. *Amy* subscribes to two features *CNAP* and *RND*. These two feature activation statuses are set as permanently

active. Terminator *B*'s name is *Bill*, whose address is *address(b, 2,1,1)*. Billy subscribes to the same features as Amy's. *AUm* is the gate through which we synchronize the formalized LOTOS test case process shown below with the WIN system. Numbers (not part of the trace notation) are used to show the correspondence of the trace with the UCM. Note that **(2)** and **(3)** do not appear in the test because they are internal.

```

process Test_1[CreateUser, AUm, Success]:noexit:=
CreateUser ! adduser ! 1 of hlrid !
hlrprofileitem(mdnstatus(amy,address(a,1,1,1),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),1,1));

CreateUser ! adduser ! 1 of hlrid !
hlrprofileitem(mdnstatus(bill,address(b,2,1,1),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),1,1));

(1) aum ! callorigination ! address(b,2,1,1) !
cni(address(a,1,1,1),amy) ! 1 of mscid;
(4) aum ! Alert !amy !b !1 of mscid;
(5) aum ! callanswer ! b ! 1 of mscid;
(6) aum ! Getcallanswer ! a ! 1 of mscid;
success;
stop
endproc (* test1 *)

```

In general, more than one scenario can be in progress at the same time. As shown in Figure 6-5, multiple symbols can move along the paths. In process *Test_2*, we create two instances for each path. Originators *Amy* and *David* are two user instances of the originator's path, while terminator *Bill* and *Claudio* are two user instances of the terminator's path. *Amy* can call *Bill*, while *David* can call *Claudio*. This multiple scenario is represented by two behavior expressions `aum ! callorigination` with a parallel operator in process *Test_2*.

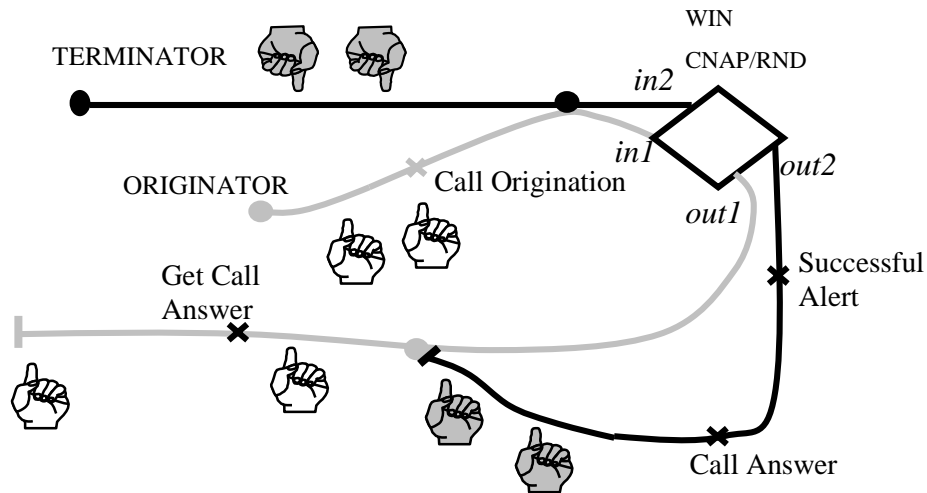


Figure 6-5 Multiple Scenarios on Alternative Path

```

process Test_2[CreateUser,AUm, Success]:noexit:=

CreateUser !...
CreateUser !...
CreateUser !...
CreateUser !...

aum ! callorigination ! address(b,2,1,1) ! cni(address(a,1,1,1),amy) !
! 1 of mscid;
...
success;
stop
|||
aum ! callorigination ! address(c,3,2,2) ! cni(address(a,1,1,1),David)
! 1 of mscid;
...
success;
stop
endproc (* test1 *)

```

Scenario(s) on different Chosen Paths:

A UCM may contain concurrent paths. In both Terminator and Originator’s UCM, there is an *And-Fork* notation which represents multiple paths for a given user instance. Suppose that there is a user instance *A* on the originator’s path and a user instance *B* on

the terminator's path. We can derive the test case shown below, where the terminator *B* concurrently follows two paths: one is the same as described above; the other is to roam into another area.

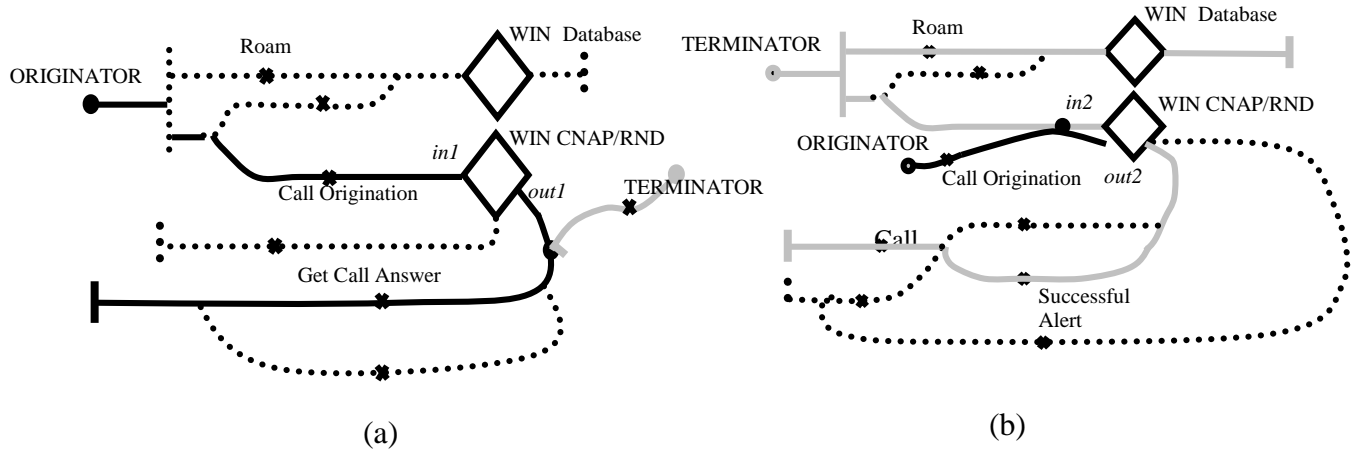


Figure 6-6 Chosen Paths from CNAP Service Description UCM

```

process Test_3[CreateUser, AUm, Success]:noexit:=
CreateUser ! adduser ! 1 of hlrid !
hlrprofileitem(mdnstatus(amy,address(a,1,1,1),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),1,1);

CreateUser ! adduser ! 1 of hlrid !
hlrprofileitem(mdnstatus(bill,address(b,2,1,1),insert(cnap_permanent,insert(rnd_permanent,{ })),insert(cnap,insert(rnd,{ })),1,1);

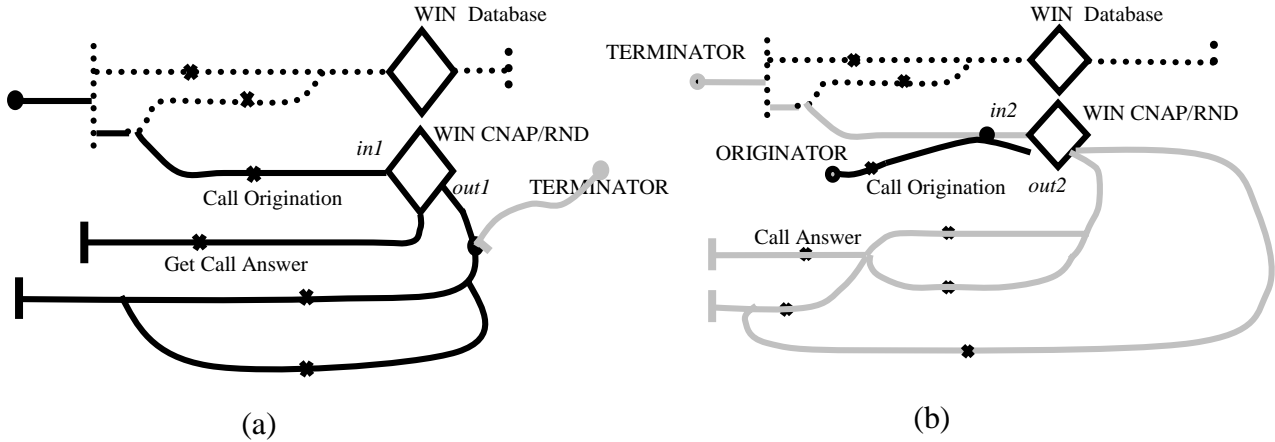
aum ! callorigination ! address(b,2,1,1) ! cni(address(a,1,1,1),amy) !
1 of mscid;
aum ! Alert !amy !b !1 of mscid;
aum ! callanswer ! b ! 1 of mscid;
aum ! Getcallanswer ! a ! 1 of mscid;
success;
stop
|||
roam !b !2 of mscid !2 of vlrid;
success;
stop
endproc (* test1 *)

```

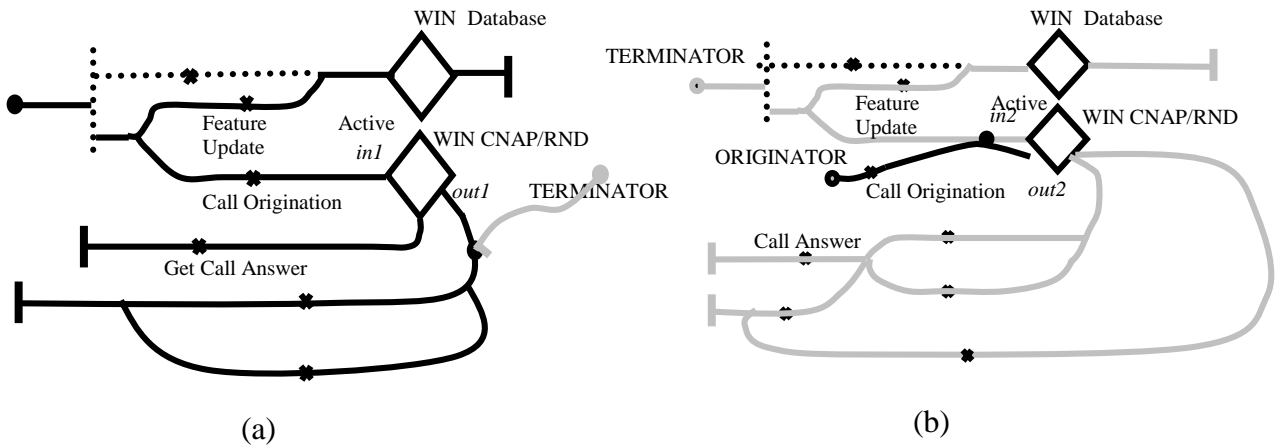
Generally, we separate our derived test cases into three groups.

- The first group is to validate the call processing without considering that the user can control the feature activation or the user may roam to another area. In total, 17 test

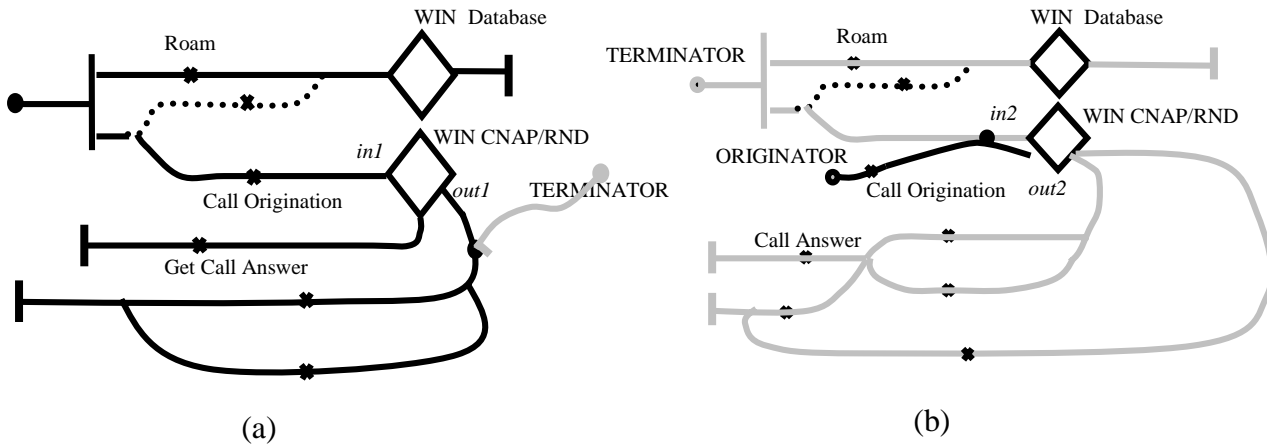
cases are included in this group. These test cases are chosen on the basis of the call processing paths shown in the continuous solid lines in the *Originator* and *Terminator* UCMs.



- The second group is to validate the feature de-activation / activation functionalities. In total, 5 test cases are included in this group. These test cases are chosen on the basis of the combination of the feature de-activation / activation path and the call processing path in the *Originator* and *Terminator* UCMs.



- The third group is to validate the service provision while the user is roaming. In total, 5 test cases are included in this group. These test cases are chosen on the basis of both the roaming path and the call processing path concurrently.



6.3.2.2 Structure Coverage

After generating the set of test suites from the service description UCM, we want to measure the efficiency of the set in terms of structural coverage. In [AmLo99], *structure coverage means a technique for monitoring and identifying portions of the specification and to measure the completeness of the test suite with respect to the syntactic structure of the specification.* In that report, a technique for coverage measurement is introduced. This technique is based on probe insertion in the specific context of the formal language LOTOS. We are going to adopt this idea to measure the efficiency of the validation test suite generated. The reader is referred to [AmLo99] for a thorough discussion of the technique.

6.3.2.2.1 Probe Insertion in LOTOS

Probe insertion is a well-known white-box technique for identifying what is the percentage of the code that has been exercised [Probert82]. Usually, a probe is a statement associated with a counter, which is initially set to 0. In our LOTOS specification, we first define 32 probe identifiers (P1-P32). We can insert the probes into specific locations. When a test case synchronizes with the specification, it executes the probes along the path. The executed probes indicate what part of the code is reachable and how many times they are reached. The data type of Probe is defined as below:

```

type TypeProbeTag is NaturalNumber, Boolean
sorts ProbeTag
opns

P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20,
P21, P22, P23, P24, P25, P26
  :->ProbeTag
...

```

There is a counter associate with each probe. The counter indicates the number of times the probe is reached. We design a *ProbeBag* as a set of probes and their associated counters. In the *ProbeBag* data type, *insertTag (ProbeTag, Nat, ProbeBag)* is a constructor operation where *Nat* is the counter. There is another operation *addTag (ProbeTag, ProbeBag)* in the *ProbeBag* data type. Each time a probe is executed by a test cases, if the probe has never been executed before, the probe is inserted into the *ProbeBag*. If at that time this probe already exists in the *ProbeBag*, the operation *addTag* only increases the probe's associated counter. The *ProbeBag* data type is shown below:

```

type TypeProbeBag is NaturalNumber, TypeProbeTag
sorts ProbeBag
opns
{ } :-> ProbeBag (* constructor *)
insertTag: ProbeTag, Nat, ProbeBag -> ProbeBag (* constructor *)
addTag: ProbeTag, ProbeBag -> ProbeBag

```

After we synchronize the test cases with the specification, we want to observe the probes and the counters reached by the test case. A process *Observer* is designed to collect probe statistic information. When we compose test cases with the inserted probe specification, the reached *ProbeTag* along the trace is sent to the *Observer* through an internal gate *Probe*. After the *Observer* process receives a *ProbeTag*, it recursively instantiates and adds the received probe (*addTag(tag, pb)*) into its *ProbeBag* parameter. We can observe the current *ProbeBag* through an external gate *Observe*. Checking which probes have been inserted in the *ProbeBag*, we know which part of the specification has been visited by test cases. Probes with high-count numbers indicate the frequently visited part of the specification. Such probes may show the possible bottlenecks in the system. Probes that have not been visited might indicate that the test case is not complete or that part of specification is not reachable. The LOTOS implementation of *Observer* process is shown in Figure 6-7:

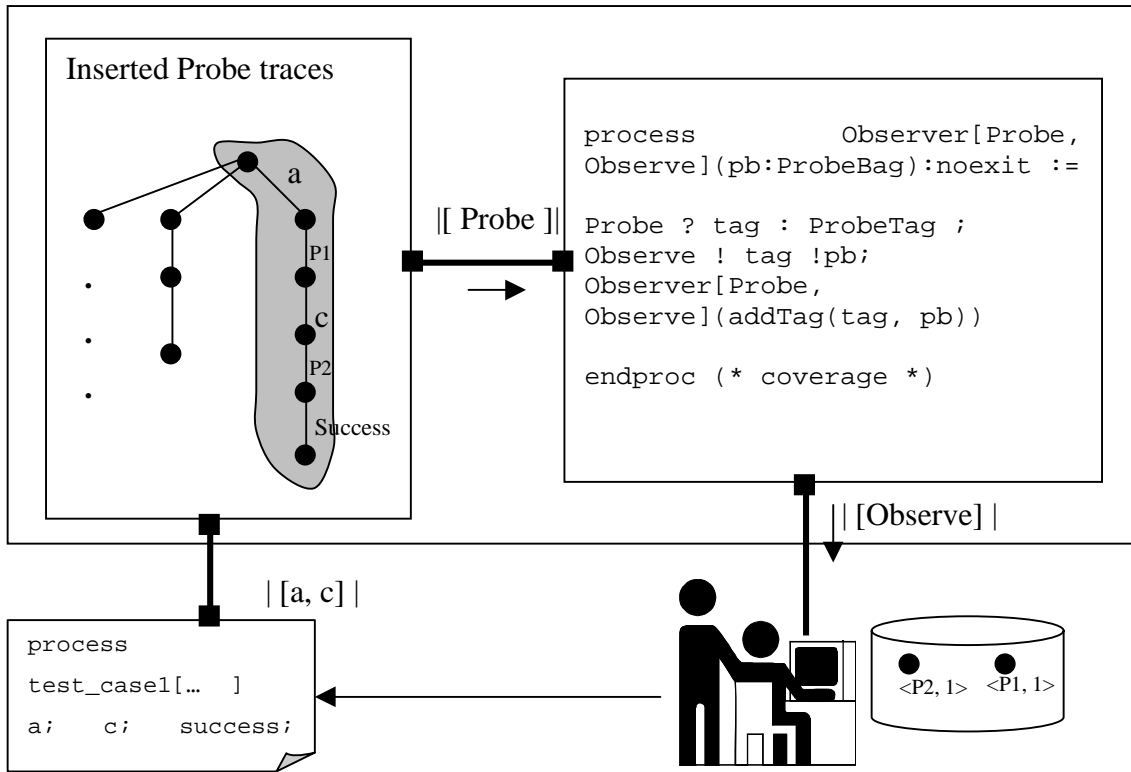


Figure 6-7 Probe Observer

The next issue is where to insert probes into our specification. To insert the defined probes, we adopt the *improved Probe Insertion Strategy* from [AmLo99]. Before we explain this strategy, we need to introduce the simple Probe Insertion Strategy.

In [AmLo99], a basic behavior expression (BBE) and a behavior expression (BE) are defined. A BBE can be either an inaction *stop*, a successful termination *exit*, or a process instantiation ($P[...]$), while a behavior expression (BE) can be one of the followings:

- A BBE (such a BE is also called a simple BBE).
- A BE prefixed by a unary operator, such as the action prefix ($;$), a hide, a let, or a guard.
- Two BEs composed through a binary operator, such as choice ($[]$), an enable (\gg), a disable (\gg), or one of the parallel composition operators ($[...]$, $||$, or $|||$).
- A BE in parentheses

In the simple probe insertion strategy, for each event e and each behavior expression BE , the expression $e; BE$ is transformed into $e; Probe ! p_id; BE$ where $Probe$ is a hidden gate and P_id is a unique identifier. This strategy can measure the event coverage because a probe that is visited guarantees, by the action prefix inference rule, that the prefixed event has been performed. In this case, if all the probes are visited by at least one test case in the validation test suite, then we have achieved a total event coverage. However, such simple probe insertion strategy has two problems. First, the number of probes required can be too high. Secondly, such approach does not cover a simple BBE containing no actions. Therefore, in [AmLo99], the improved Probe Insertion Strategy is introduced to solve the above two problems.

The new strategy can reduce the number of probes. If there is a sequence of actions prefixes, we insert only one probe just before the ending BBE . If $*$ is one the LOTOS binary operators, in a generic pattern $BE*BE$, where BE is represented as $e; B$ and B is not a simple BBE , the probe is not required after the event e . This is because the behavior expression B will contain probes itself, and a visit to any of these probes ensures that event e is reached. Table 6-2 presents one example of such case.

Original LOTOS specification	Probe inserted, using the improved strategy
<pre> process SSF_CCF1[c, e,..., Probe]:exit:= c ! LocReq ...; c ! ReLocReq ...; ... exit [] e ...; exit endproc (* setup_Channel *) </pre> <p style="text-align: right;">→ BE</p> <p style="text-align: right;">→ BE</p>	<pre> process SSF_CCF1 [C, ..., Probe]:exit:= c ! LocReq ...; c ! ReLocReq ... ; ... Probe ! p1; exit [] e;... Probe ! p2; Exit endproc (* setup_Channel *) </pre> <p style="text-align: right;">→ BBE</p> <p style="text-align: right;">→ BBE</p>

Table 6-1 Probe Insertion in case of “BE*BE”

Second, for the case of a simple BBE (without any action prefix), if we are to prefix the BBE with a probe in the generic pattern BBE * BE and BE * BBE, we must be careful not to introduce any new non-determinism:

- BBE is stop: No probe is required
- BBE is a process instantiation P[...]: A probe before the BBE can be safely used except when * is the choice operator[], or when * is the disable operator ([>]) with the BBE on its right.
- BBE is exit: The constraint and solution is the same as for the process instantiation.

Table 6-2 shows an example of such case.

Original LOTOS specification	Probe inserted, using the improved strategy
<pre> process MS_Subscribers[AUm,createuser,... Probe]:exit:= </pre> <div style="border: 1px dashed black; padding: 5px; margin: 5px 0;"> <pre> createuser ! ...; ... MS[...] </pre> </div> <p style="text-align: right; margin-right: 20px;">→ BE</p> <div style="border: 1px dashed black; padding: 5px; margin: 5px 0;"> <pre> MS_Subscribers[...] </pre> </div> <p style="text-align: right; margin-right: 20px;">→ BBE</p> <pre> endproc (* MS_Subscribers *) </pre>	<pre> process MS_Subscribers[AUm,createuser,... Probe]:exit:= </pre> <pre> createuesr ! ...; ... Probe ! p14; MS[...] </pre> <p style="text-align: center;"> </p> <pre> Probe ! p15; MS_Subscribers[...] </pre> <pre> endproc (* MS_Subscribers *) </pre>

Table 6-2 Probe Insertion in case of “BE*BBE”

6.3.2.2.2 Coverage Results

Table 6-3 demonstrates the coverage results of a given set of test scenarios. These test scenarios were derived with the guidance of the UCMs as described in section

6.3.2.1. We actually derived a total of 27 test cases, but we only show in Table 6-3 a subset of them that covered all probes in our LOTOS specification. There are more test cases that cover other scenarios but these are not all shown because all the probes are covered by the listed scenarios.

In the following test scenario descriptions, *Oa* is an abbreviation of *Originator a*, *Tb* is an abbreviation of *Terminator b*, *Tc* is an abbreviation of *Terminator c*. The covered probes are shown in the next column where the probes are highlighted in bold the first time they appear.

Test Group #	Test Scenarios	Probe Covered
1	<i>Tb</i> subscribes CNAP; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> is alerted with CNI(<i>Oa</i>); <i>Tb</i> answers the call; <i>Oa</i> gets Call Answer	<p 12 ,1>; <p2,3>; <p7,1>; <p 23 ,1>; <p1,2>; <p5,2>; <p 16 ,1>; <p 19 ,1>; <p 24 ,1>; <p 27 ,1>; <p 28 ,1>; <p 30 ,1>
1	<i>Tb</i> subscribes CNAP; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> is alerted with CNI(<i>Not Avail</i>); <i>Tb</i> answers the call; <i>Oa</i> gets Call Answer	<p 8 ,1>; <p12,1>; <p2,3>; <p7,1>; <p23,1>; <p1,2>; <p5,2>; <p16,1>; <p19,1>; <p24,1>; <p27,1>; <p28,1>; <p30,1>
1	<i>Tb</i> subscribes CNAP; <i>Tc</i> subscribes CNAP and RND; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> is alerted with CNI(<i>Oa</i>); <i>Tb</i> does not answer the call; <i>Tb</i> forwards call to <i>Tc</i> ; <i>Tc</i> is alerted with CNI(<i>Oa</i> , <i>Tb</i>); <i>Tc</i> answers the call. <i>Oa</i> gets Call Answer;	<p 13 ,1>; <p2,3>; <p7,1>; <p1,4>; <p23,2>; <p5,5>; <p16,2>; <p19,2>; <p27,2>; <p24,2>; <p28,2>; <p30,2>; <p 25 ,1>; <p 10 ,1>; <p 26 ,1>; <p 20 ,1>
1	<i>Tb</i> subscribes CNAP; <i>Tc</i> subscribes CNAP and RND; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> is alerted with CNI(<i>Oa</i>); <i>Tb</i> does not answer the call; <i>Tb</i> forwards call to <i>Tc</i> ; <i>Tc</i> is alerted with CNI(<i>Oa</i> , <i>Tb</i>); <i>Tc</i> does not answer the call. <i>Oa</i> does not get Call Answer;	<p 22 ,1>; <p2,3>; <p7,1>; <p23,2>; <p1,4>; <p16,2>; <p5,5>; <p 17 ,2>; <p24,2>; <p27,2>; <p28,2>; <p30,2>; <p25,1>; <p10,1>; <p26,1>;
1	<i>Tb</i> subscribes CNAP; <i>Tc</i> subscribes CNAP and RND; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> is alerted with CNI(<i>Oa</i>); <i>Tb</i> does not answer the call; <i>Tb</i> forwards call to <i>Tc</i> ; <i>Tc</i> is alerted without CNI; <i>Tc</i> answers the call; <i>Oa</i> gets Call Answer.	<p13,1>; <p2,3>; <p7,1>; <p1,4>; <p23,2>; <p5,5>; <p16,2>; <p19,2>; <p27,2>; <p24,2>; <p28,2>; <p 29 ,2>; <p25,1>; <p10,1>; <p26,1>; <p20,1>

1	<i>Tb</i> subscribes CNAP; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> is alerted with CNI(<i>Oa</i>); <i>Tb</i> does not answer; <i>Oa</i> does not get answer;	<p11,1>, <p7,1>, <p23,1>, <p2,3>, <p1,2>, <p16,1>, <p5,2>, <p19,1>, <p24,1>, <p27,1>, <p28,1>, <p30,1>
1	<i>Tc</i> subscribes CNAP; <i>Oa</i> calls <i>Tc</i> ; <i>Tc</i> is alerted with CNI(<i>Oa</i>); <i>Tc</i> does not answer; <i>Oa</i> does not get answer;	<p5,1>, <p7,1>, <p23,1>, <p2,3>, <p1,1>, <p16,1>, <p5,2>, <p19,1>, <p24,1>, <p27,1>, <p14,1>, <p22,1>, <p28,1>, <p30,1>
1	<i>Tc</i> subscribes CNAP; <i>Tb</i> subscribes CNAP and RND; <i>Oa</i> calls <i>Tc</i> ; <i>Tc</i> is alerted with CNI(<i>Oa</i>); <i>Tc</i> does not answer; <i>Tc</i> forwards call to <i>Tb</i> ; <i>Tb</i> is alerted with CNI(<i>Oa,Tc</i>); <i>Tb</i> does not answer; <i>Oa</i> does not get answer	<p30,1>, <p7,1>, <p23,2>, <p2,3>, <p1,4>, <p16,2>, <p5,5>, <p19,2>, <p24,2>, <p27,2>, <p21,1>, <p25,1>, <p28,2>, <p30,1>, <p15,1>, <p26,1>, <p12,1>
1	<i>Tc</i> subscribes CNAP; <i>Oa</i> calls CNI(<i>Tc</i>); <i>Tc</i> is alerted with <i>Oa</i> ; <i>Ob</i> calls <i>Tc</i> ; <i>Ob</i> gets busy tone; <i>Tc</i> answers; <i>Oa</i> gets call answer;	<p5,1>, <p7,2>, <p23,2>, <p2,3>, <p1,2>, <p16,2>, <p5,4>, <p19,1>, <p24,2>, <p27,2>, <p28,1>, <p30,1>, <p6,1>, <p9,1>, <p13,1>, <p18,1>, <p17,1>, <p20,1>
2	<i>Tb</i> subscribes CNAP; <i>Tb</i> de-activates the CNAP; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> is alerted with CNI (<i>Not_Avail</i>); <i>Tb</i> answers the call. <i>Oa</i> gets Call Answer.	<p12,1>; <p2,3>; <p4,1>; <p7,1>; <p23,1>; <p1,2>; <p16,1>; <p5,2>; <p27,1>; <p19,1>; <p24,1>; <p28,1>; <p32,1>
2	<i>Tc</i> subscribes CNAP and RND; <i>Tc</i> de-activates the RND; <i>Tb</i> forwards call to <i>Tc</i> ; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> is alerted with CNI(<i>Oa</i>); <i>Tb</i> does not answer the call. <i>Tc</i> is alerted with CNI (<i>Oa</i>); <i>Tc</i> answers the call. <i>Oa</i> gets call answer.	<p6>, <p7,1>, <p23,2>, <p2,3>, <p4,1>, <p1,4>, <p16,2>, <p5,5>, <p19,2>, <p24,2>, <p27,2>, <p28,2>, <p30,1>, <p25,1>, <p6,1>, <p10,1>, <p26,1>, <p31,1>, <p13,1>
3	<i>Tb</i> subscribes CNAP; <i>Oa</i> calls <i>Tb</i> ; <i>Tb</i> roams to the other area. <i>Tb</i> is alerted with CNI (<i>Oa</i>); <i>Tb</i> answers the call; <i>Oa</i> gets Call Answer.	<p12,1>; <p2,3>; <p3,1>; <p7,1>; <p23,1>; <p1,2>; <p5,2>; <p16,1>; <p19,1>; <p27,1>; <p24,1>; <p28,1>; <p30,1>

Table 6-3 Coverage Result of Generated Test Cases

6.3.3 Automatic Graphic Scenario Generation

By applying synchronization between testing processes and the LOTOS specification, LOLA can generate system scenarios with many internal steps in a symbolic format. Each scenario shows a single trace of LOTOS actions. Since this format is not easy to read without understanding its syntax, we wrote a perl script to translate the trace format into MSC PR format. The format of MSC PR has been described in Chapter 3 “Selected Techniques”. The SDT MSC editor can recognize the PR syntax and generate a corresponding graphical MSC. These generated graphic MSCS are compatible with the MSC notation used in the standard. As the result, we are able not only to verify the MSCs covered within the standard, but also to generate more MSCs for additional analysis. Figure 6-8 shows the procedure of graphic MSC generation combined with test process in LOTOS. In Figure 6-9, it shows a concrete example of generated graphic system MSC for the test process Test_1 that was described in Section 6.3.2.1

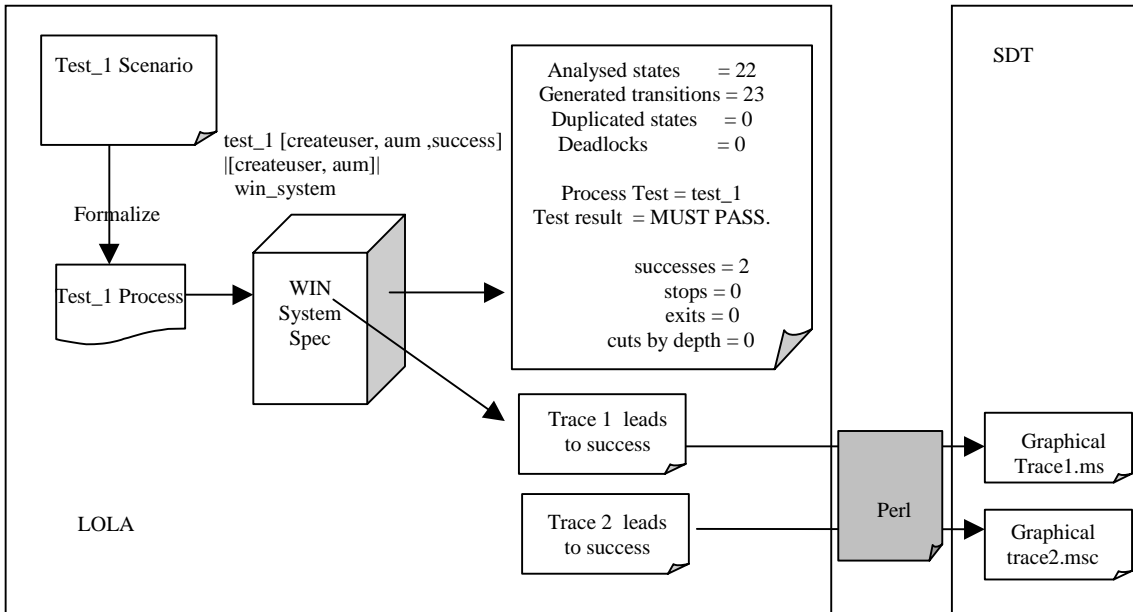


Figure 6-8 Graphical MSC Generation

Generated System Scenario:

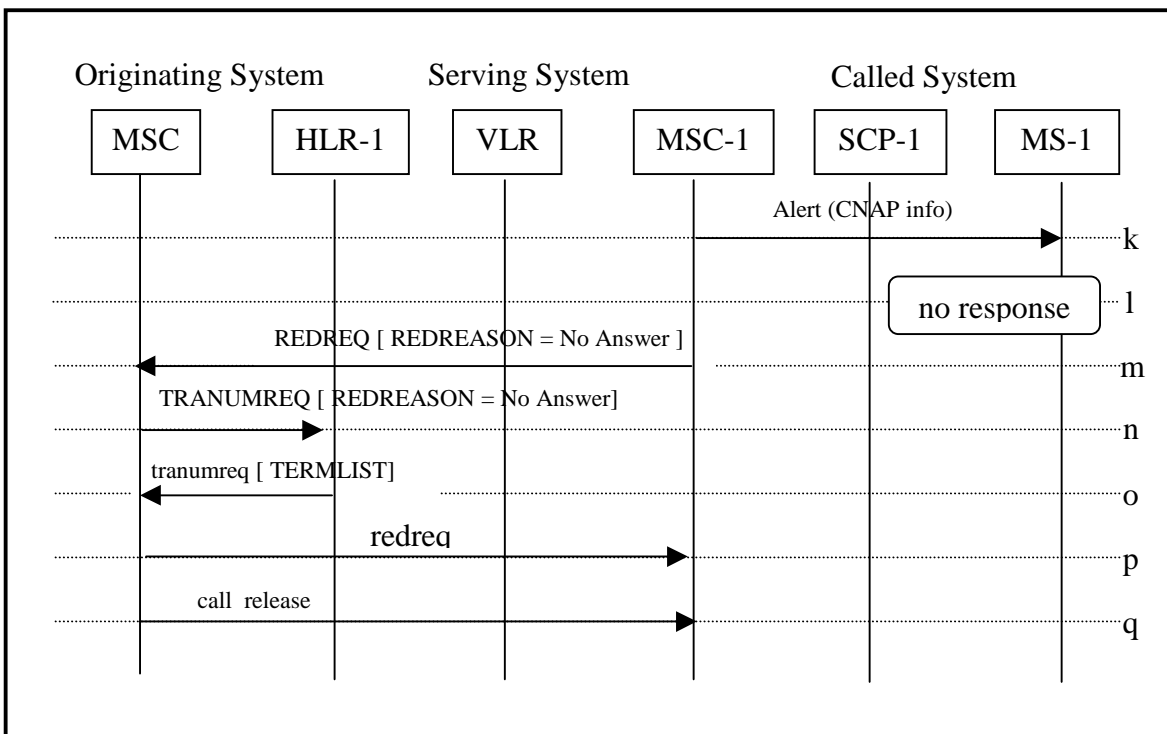
Figure 6-9 Generated Graphical System MSC for Test_1

6.4 Discovery of a number of ambiguities and inconsistencies in the Draft Standard

There were five WIN scenarios (scenario 8.1.1 to scenario 8.1.5) included in the standard when this thesis was initiated. After applying simulation, testing and graphic MSC generation, several ambiguities and inconsistencies existing in the WIN standard scenarios were found. These are symptoms of possible design errors, and were fed back to the standardization committee for their consideration.

All the scenarios show below assume that MS calls MS-1 that is in a different location area, and is served by MSC-1. All scenarios also assume that MS-1 subscribe to CNAP and it has placed itself in Call Forward No Answer (CFNA) to another MS-2. The address to which the call must be forwarded for MS-1 is in HLR-1.

6.4.1 Scenario 8.1.3



- Possible Consistency Problem

In the scenario 8.1.3, step *m*, if MS-1 does not respond, the serving MSC-1 would send a REDREQ, including the redirection reason “No Answer”. But in the Scenario 8.1.5 shown in section 6.4.2, the redirection reason becomes “No response”.

- Possible Incompleteness Problems

This incompleteness shows in the above scenario 8.1.3 and is illustrated in Figure 6-10.

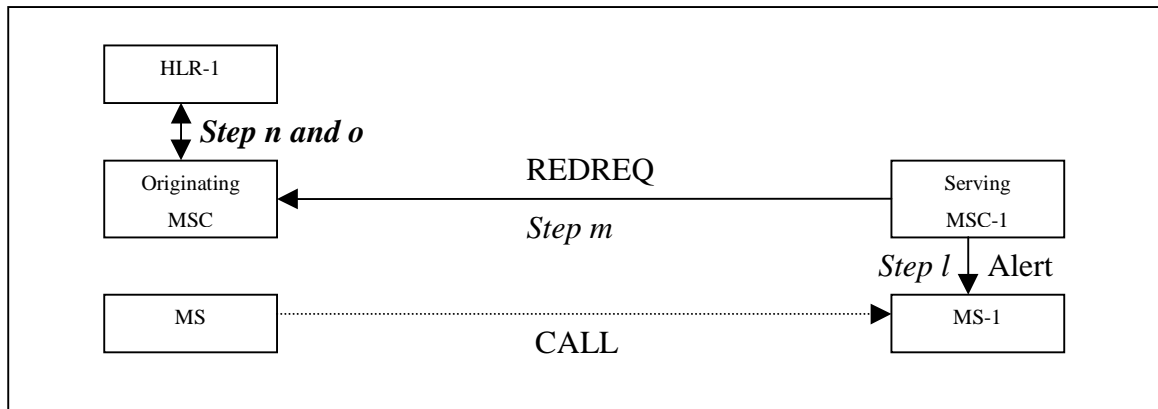


Figure 6-10 Redirection Scenario Of CNAP

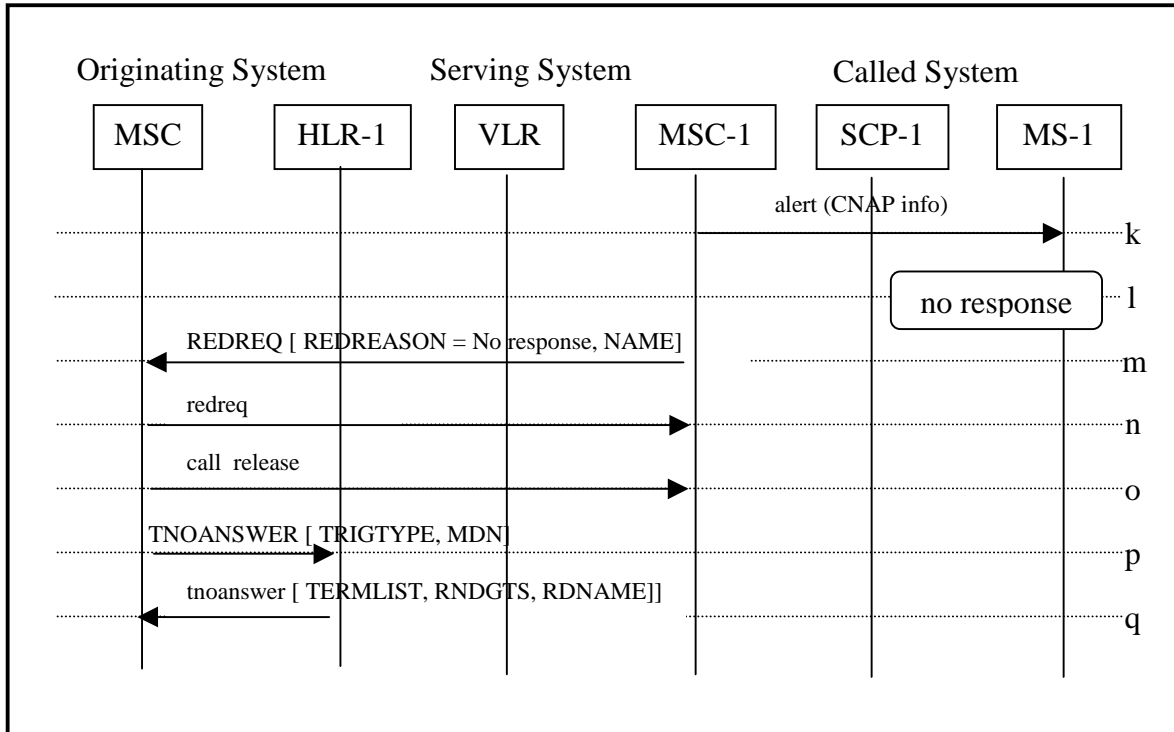
There are two different problems here.

MSC-1 (serving MSC) alerts MS-1, which does not respond (step *l*). It then asks the originating MSC to redirect the call (step *m*). The originating MSC queries HLR-1 for the redirecting information (step *n*). However, there is an incompleteness problem in step *n*, since the message TRANUMREQ does not include the MS-1 identifier, and so HLR-1 will not be able to search the address of the party to which his call should be redirected.

In addition, suppose that in some way the problem mentioned above can be overcome and that at step *o*, HLR-1 provides the redirect party’s address in the format of TERMLIST. However, HLR-1 does not provide the name of MS-1. Later on, the originating MSC must establish a call to the forwarded party. Since the name of MS-1

was not provided by HLR-1, this name cannot be displayed at the forwarded MS-2. Therefore, even if the latter subscribes to RND, the feature cannot be supported.

6.4.2 scenario 8.1.5



- In scenario 8.1.5, step *m*, when MSC-1 sends the redirection request to the original MSC, NAME represents the originating call party's name information and is included in the message REDREQ. But, since the original MSC already knows such information, it is redundant to include NAME here.

6.5 Conclusion

In this chapter, we apply various validation techniques to the prototype specification of CNAP service. Our validation focuses on the simulation, testing, and graphical MSC generation. As a result, some inconsistencies and ambiguities existing in the CNAP design are found. It should be noted that none of the problems detected use of a major behavior nature. However, these problem were found by using our formal specification

method after the standard had been extensively validated by the committee using only conventional manual methods. Furthermore, several unclear points in the standard that could have led to implementation ambiguities or errors, were clarified by us in the process of formally specifying and validating the standard draft.

One additional advantage of our method is that many CNAP scenarios can be derived from the specification in addition to the five given in the draft standard. The standard committee could have used our specification as a precise prototype of the CNAP procedures. Unfortunately, the RND option was taken out of the subsequent version of the draft.

Chapter 7 Contributions and Future Work

In this chapter, the results that have been reported in this thesis are summarized. Section 7.1 contains an overview of our contribution. In Section 7.2, some possible future research areas are discussed as a complement of this thesis.

7.1 Contribution of the thesis

Generally, the major contributions of this thesis are:

- **Scenario-Oriented Design Approach**
Following the work of [AmAn99], we demonstrate an incremental design approach based on high-level scenarios called Use Case Maps (UCMs). UCMs are at a higher level of abstraction than MSCs and therefore are more appropriate than MSCs at the earliest stage of design. Such scenario analysis helps us in the steps from the informal service description phase towards a formal specification.
- **LOTOS specification of a simplified WIN model with the also simplified CNAP feature.**

A LOTOS specification of a simplified WIN network and its included CNAP feature was given in Chapter 5. Since it is difficult to specify a WIN system of real size, we use this small model for our research goals. The structure of our specification is based on the Network Reference Model (NRM) of WIN. Its behavior is based on the

scenarios analyzed in Chapter 4. To our knowledge, this is the first formal specification of a WIN feature in any language.

- Validation of the CNAP feature.

In Chapter 6, we present a number of validation techniques for our LOTOS specification. Firstly, we apply simulation to verify the consistency between the requirements and the formal specification. Test cases are then generated from the scenario analysis and transformed into LOTOS test processes. Probes are used to verify the structural coverage of the specification. The probe insertion process shows that a complete set of test cases for the structural coverage of the specification can be obtained, with the guidance from UCMs. Moreover, the generated LOTOS traces are transformed into graphic MSC scenarios by using a perl script. During this process, some ambiguities and incompleteness of CNAP scenarios from the WIN pre-balloting draft can be found. The techniques used are not new, however, this thesis provides a new substantial example of their use.

- Validation of incomplete designs

As discussed early in this thesis, much research has been dedicated to the validation of complete protocols where messages, states, etc. are fully defined. This thesis shows that it is possible to capture a protocol design at a preliminary stage, where it still presents considerable looseness, and to validate it at least partially at this stage.

7.2 Future Work

Our WIN model supports unlimited MS subscribers, two HLRs, two VLRs, and two MSCs. It is a subset of WIN that is sufficient to represent all the scenarios included in the specification of the CNAP feature provided at the time this work was initiated. This model can be made more general by including the capability of dynamically generating new network entities and automatically initializing the communication channel between the newly created entities and the existing ones in the WIN model.

In this thesis, only the CNAP feature was considered. In the future, more WIN features, such as Incoming Call Screen and Voice Call Screen, could be integrated in our specification. When multiple features exist in parallel, feature interaction detection can be applied [StLo95][FaLo93].

We also suggest that future work be directed towards a strategy for deriving MSC scenarios from UCM. The derivation part is not done in our thesis since we limited our analysis to the MSCs that already existed in the standard.

REFERENCES

(References are sorted by date)

[WIN ANSI-41.3-C Additions] August 19, 1997.

[WINT] *Wireless Intelligent Network Tutorial* <http://www.webproforum.com/dsc/>

[GHM 78] J. V. Guttag, E. Horowitz, and D. R. Musser, *Abstract Data Types and Software Validation* Communications of the ACM, December 1978, Volume 21, Number 12, University of Southern California.

[Mil80] R. Milner, *A Calculus of Communicating System*, Lecture Notes in Computer Science, (Springer-Verlag, 1980) No.92.

[Probert82] R. L. Probert, *Optimal Insertion of Software Probes in Well-Delimited Programs*, IEEE Transactions on Software Engineering, Vol 8, No1, January 1982, 34-42.

[Hoar85] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985

[EM85] B. Ehrig, B. Mahr, *Fundamentals of Algebraic Specifications*, Springer-Verlag, 1985.

[BE88] E. Brinksma, *A theory for the derivation of tests*. In: S. Aggarwal and K. Sabnani (Eds), *Protocol Specification, Testing and Verification VIII*, North-Holland, 63-74, June 1988.

[HH88] M. Haj-Hussein, *An Interactive System for LOTOS Application (ISLA)*, Master Thesis, University of Ottawa.

[VSSB89] C.A. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma, *Specification styles in distributed systems design and verification* in *Theoretical Comp. Sc.*, 1991, 179-206

[Bo91] R. Boumezbeur, *Design, Specification, and Validation of Telephony System in LOTOS* Thesis of Comp. Sc., September 1991

[PL91] S. Pavón, and M. Llamas, *The testing Functionalities of Lola*. In: J. Quemada, J.A. Mañas, and E. Vázquez (Eds), *Formal Description Techniques, III*, IFIP/North-Holland, 1991, 559-562.

[BoZu92] W. Bouma and H. Zuidweg, *Formal Analysis of Feature Interactions by Model Checking*. 1992

[LFH92] L. Logrippo, M. Faci, M. Haj-Hussein, *An Introduction to LOTOS : Learning by Examples*, Computer Networks and ISDN Systems 23, 1992

[DaNa93] O. Dahl and E. Najm, *Specification and Detection of IN Service of IN Service Interference Using LOTOS*. Proceedings Forte '93, eds. R. L. tenney, P. D. Amer, M. U. Uyar, Boston 1993, 53-71

[StLo93] B. Stepien and L. Logrippo, *Status-Oriented Telephone Service Specification*. In: T. Rus and C. Rattray (eds) *Theories and Experience for Real-Time System Development*. AMAST series in computing, Vol. 2, World Scientific, 1994, pages 265-286.

[BoLo93] R. Boumezbeur, L. Logrippo, *Specifying Telephone System in LOTOS*. IEEE Communication Magazine, Aug. 1993, 38-45

[CKM93] A. R. Cavalli, S.U. Kim, and P. Maignon, *Improving Conformance Testing for LOTOS*. In: R.L. Tenney, P.D. Amer and M. U. Uyar (Eds), FORTE VI, 6th International Conference on Formal Description Techniques, North-Holland, 367-381, October 1993.

[FaLo93] M. Faci, L. Logrippo, *Specifying Features and Analyzing their Interactions in a LOTOS Environment*. In: L. G. Bouma and H. Velthuijsen (eds.) *Feature Interactions in Telecommunications Systems*. IOS Press, 1994 (proc. Of the 2nd International Workshop on Feature Interactions in Telecommunications Systems, Amsterdam) 136-151. Also published in S. Brlek (ed.) BMW-94, 1994, 167-182

[Am94] D. Amyot, *Formalization of Timethreads Using LOTOS*, Master's Thesis, University of Ottawa, 1994

[Q1201] *ITU-T/ETSI Recommendation Q1201*, 1993.

[Q1202] *ITU-T/ETSI Recommendation Q1202*, 1993

[Q1203] *ITU-T/ETSI Recommendation Q1203*, 1993.

[Q1204] *ITU-T/ETSI Recommendation Q1201*, 1993.

[Q1205] *ITU-T/ETSI Recommendation Q1202*, 1993

[Q1211] *ITU-T/ETSI Recommendation Q1203*, 1993.

[Q1213] *ITU-T/ETSI Recommendation Q1201*, 1993.

[Q1214] *ITU-T/ETSI Recommendation Q1202*, 1993

[Cheng94] K.E. Cheng, *Towards a Formal Model for Incremental Service Specification and Interaction Management Support*. Second International Workshop on Feature

Interactions in Telecommunication Software Systems. Eds. L. G. Bouma and H. Velthuisen, IOS Press 1994, pages 152-166.

[CGR94] D. Craigen, S. Gerhart, T. Ralston, *Industrial applications of formal methods to model, design, and analyze computer systems: an international survey*. Noyes Data Corporation (Publisher), USA. 1994

[ITU95] ITU *Recommendation Z. 105, SDL Combined with ASN.1 (SDL/ASN.1)*. Geneva, 1995

[BLV95] T. Bolognesi, J. van de Lagemaat, and C. Vissers, *LOTOSphere: Software Development with LOTOS*. Kluwer Academic Publishers, The Netherlands, 1995.

[PLR95] S. Pavón, D. Larrabeiti, and G. Rabay, *LOLA-User Manual, version 3.6. DIT*, Universidad Politécnica de Madrid, Spain, LOLA/N5/V10 (February, 1995)

[ABBL95] D. Amyot, F. Bordeleau, R.J.A. Buhr, and L. Logrippo, *Formal support for design techniques: a Timethreads-LOTOS approach*. In: FORTE VIII, 8th International Conference on Formal Description Techniques, Chapman & Hall, 1995, pp. 57-72.

[LaRa95] M. Laitinen, J. Rantala, *Integration of Intelligent Network Services into Future GSM Networks*, IEEE Communication Magazine, June 1995.

[StLo95] B. Stepien, L. Logrippo, *Feature Interaction Detection Using Backward Reasoning with LOTOS*. In: S. Vuong (ed.) Protocol Specification, Testing, and Verification, XIV (Proc. Of the 14th International Symposium on Protocol Specification, Testing, and Verification, organized by IFIP WG6.1, Vancouver), 1995, 71-86.

[Grin96] A. Grinberg, *Seamless Networks*. McGraw-Hill, 1996

[Buhr96] R.J.A. Buhr, *Use Case Map for Object-Oriented Systems*, Prentice Hall, 1996.

[JaKa96] Jalel Kamoun, Master Thesis: *Formal specification and Feature Interaction in the IN*, University of Ottawa, January, 1996

[RaTu96] Randall Tuok, Master Thesis: *Modeling and Derivation of Scenarios for a Mobile Telephony System in LOTOS*, University of Ottawa, 1996.

[ITU-T96] ITU-T, *Recommendation Z.120- Message Sequence Chart (MSC)*, 1996.

[RGF96] E. Rudolf, J. Grabowski and P. Graubmann, *Tutorial on Message Sequence Charts (MSC'96)*, Tutorial of the FORTE/PSTV'96 conference in Kaiserslautern, Germany, Oct. 1996.

[BoBu97] F. Bordeleau and R.J.A. Buhr, *The UCM-ROOM Design Method: from Use Case Maps to Communicating State Machines*. In: Conference on the Engineering of Computer-Based Systems, Monterey (CA), USA, March 1997.

[ALF97] Daniel Amyot, Luigi Logrippo and Pascal Forhan, *Formal Specification and Validation of GPRS Group-Call Using a Scenario-Based Approach*, August 1997.

[FGJ97] I. Faynberg, L. R. Gabuzda, T. Jacobson, *The Development of the Wireless Intelligent Network (WIN) and its Relation to the International Intelligent Network Standards*, Bell Labs Technical Journal, Volume 2, Number 3, Summer 1997.

[WIN98] ANSI/TIA/EIA (1998) *ANSI 771, Wireless Intelligent Networks (WIN). Additions and modifications to ANSI-41 (pre-balloting version)*. July 1998.

[WIN98'] ANSI/TIA/EIA (1998) *ANSI 771, Wireless Intelligent Networks (WIN). Additions and modifications to ANSI-41 (Phase 1)*. TR-45.2.2.4, December 1998.

[GBGO98] Nancy Griffeth, Ralph Blumenthat, Jean-Charles Gregoir, and Tadashi Ohta, *Feature Interaction Detection Contest*. Feature Interactions in Telecommunications and Software Systems V. K. Kimbler and L.G. Bouma (Eds.) IOS Press, 1998

[JaLo98] J. Kamoun and L. Logrippo, *Goal-Oriented Feature Interaction Detection in the Intelligent Network Model*. Feature Interactions in Telecommunications and Software Systems V. K. Kimbler and L.G. Bouma (Eds.) IOS Press, 1998

[LWFH98] Y. Lahav, A. Wolisz, J. Fischer, E. Holz, *Implementability of Message Sequence Charts*. Proceeding of the 1st Workshop of The SDL Forum Society On SDL and MSC. Berlin, Germany, 29th June-1st July 1998

[TuLo98] R. Tuok, L. Logrippo, *Formal Specification and Use Case Generation for a Mobile Telephony System*. Computer Networks and ISDN Systems 30 , 1998. 1455-1063

[AHLF98] D. Amyot, N. Hart, L. Logrippo, and P. Forhan, *Formal Specification and Validation using a Scenario-Based Approach: The GPRS Group-Call Example*. In: ObjecTime Workshop on Research in OO Real-Time Modeling, Ottawa, Canada, January 1998.

[WPJ98] K. Weidenhaupt, K. Pohl, Jarke, Matthias, and P. Haumer, *Scenarios in System Development: Current Practice*. In: IEEE Software, March/April 1998, 34-45

[AALSY98] D. Amyot, R. Andrade, L. Logrippo, J. Sincennes, Z. Yi, *Survey of Several Specification Techniques for the Drafting of the WIN Standard*. Technical Report, University of Ottawa, 1998

[Miga98] A. Miga, *Application of Use Case Maps to System Design with Tool Support*. M.Eng. thesis 1998, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada. http://www.UseCaseMaps.org/pub/am_thesis.pdf

[ALBG99] D. Amyot, L. Logrippo, R.J.A. Buhr, and T. Gray, *Use Case Maps for the Capture and Validation of Distributed Systems Requirements*. In: Fourth International Symposium on Requirements Engineering (RE'99), Limerick, Ireland, June 1999.

[AALSY99] D. Amyot, R. Andrade, L. Logrippo, J. Sincennes, and Z. Yi, *Formal Methods for Mobility Standards*. In: IEEE 1999 Emerging Technology Symposium on Wireless Communications & Systems, Dallas (TX), USA, April 1999.

[AmAn99] D. Amyot and R. Andrade, *Description of Wireless Intelligent Network Services with Use Case Maps*. In: 17th Brazilian Symposium on Computer Networks (SBRC'99), Salvador, Brazil, May 1999.

[AmLo99] D.Amyot and L. Logrippo, *Structural Coverage of LOTOS Specification Through Probe Insertion*, Technical report, University of Ottawa, 1999.

[AmLo] Daniel Amyot and Luigi Logrippo, *Use Case Map and LOTOS for the Prototyping and Validation of a Mobile Group Call System*, To appear in Computer Communications.