From Requirements to Scenarios through Specifications:

A Translation Procedure from Use Case Maps to LOTOS

Ruoshan Guan

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
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

University of Ottawa
Ottawa, Ontario, Canada
September 2002

© Ruoshan Guan, Ottawa, Canada, 2002

Abstract

The precise specification of communication systems is a crucial part of their successful development and implementation. Different methodologies for generating high-level formal specifications from informal requirements have been used by researchers and industrial groups. This thesis proposes an automatic approach, which is used for the early stages of the design of communication systems.

In the first part of the thesis, we describe our automatic approach, which is based on Use Case Maps for the capture and design of the requirements and on LOTOS for simulation and validation.

In the second part of the thesis, we discuss the design of two tools for our automatic approach. *Ucm2LotosSpec* is a tool that supports the automatic translation from UCMs to LOTOS specifications. *Ucm2LotosScenarios* is a tool that generates LOTOS scenarios from UCMs.

Finally, our proposed approach is applied to a case study of the Location Based Service in the Wireless Intelligent Network, a standard that was under development when this work was done. Stage 2 scenarios for this standard are obtained from informal requirements represented by Use Case Maps. It is concluded that our approach is feasible, based on this experience carried out on a realistic example.

Acknowledgment

I would like to express my deep gratitude to my supervisor Professor Luigi Logrippo who provided his guidance, encouragement and support during this research, reviewed my drafts of the thesis accurately to improve its contents and presentation. His dedication to his work is always going to inspire me.

I would like to thank the members of the University of Ottawa LOTOS group for their useful discussion and helpful comments. Particularly, I would like to thank Jacques Sincennes and Dr. Daniel Amyot who gave me precious advice and suggestions through lots of discussion. I have been very fortunate to work with these wonderful people. I also extend my thanks to my committee, Dr. Daniel Amyot and Dr. Murray Woodside for reviewing and commenting this work.

I would also like to thank the Communications and Information Technology Ontario (CITO), the Natural Science and Engineering Research Council (NSERC) for their financial support. Nortel Networks, in particular Mr. John Visser, should be credited for having motivated this research, providing funding, as well as technical information.

Finally, I would like to express my eternal gratitude to my parents who give me endless love and support through all my life. I would like to dedicate this thesis to my husband, Zhijun Qiu, who always encouraged me and shared my challenges and achievements during my graduate studies.

Table of Contents

Chapter 1 Introduction 8
  1.1 Background and Motivation 8
  1.2 Three Stages in the Standardization Process 9
  1.3 Contributions of the Thesis 10
  1.4 Organization of the Thesis 11
  1.5 Related Work 12
Chapter 2 Wireless Intelligent Network 14
  2.1 Background 14
  2.2 WIN Architecture 15
    2.2.1 The WIN Reference Model 15
    2.2.2 WIN Distributed Functional Plane 17
  2.3 WIN Standard 18
Chapter 3 Review of Selected Description Techniques 20
  3.1 Use Case Maps 20
  3.2 LOTOS 23
    3.2.1 Processes, events and gates 24
    3.2.2 Basic LOTOS 24
    3.2.3 Data types 26
    3.2.4 Full LOTOS 26
    3.2.5 LOLA 26
  3.3 MSC 27
Chapter 4 Description of Our Approach 29
  4.1 Overview of *SPEC-VALUE* 29
  4.2 The Proposed Approach 30
  4.3 Conclusion 33
Chapter 5 Ucm2LotosSpec: Automatic Generation of LOTOS Specification From Use Case Maps 34
  5.1 Purpose 34
  5.2 Overview 35

5.3 Subset of UCM Notation supported 36
5.4 Basic Path Elements Translation For Unbound UCMs 37
5.4.1 Principles of Translation 37
5.4.2 Start points, end points and responsibilities. 38
5.4.2.1 Start Point and End Point 38
5.4.2.2 Responsibilities 39
5.4.3 OR-Fork 39
5.4.4 OR-Join 40
5.4.5 AND-Fork 40
5.4.6 AND-Join 41
5.4.7 Generic Version of AND-Fork and AND-Join 42
5.4.8 OR-Join with OR-Fork or AND-Fork 43
5.4.9 Waiting place 45
5.4.10 Interacting Paths 46
5.5 Translation for UCMs with Stubs and Plugins 47
5.5.1 Principle of Translation for UCM with Stubs and Plug-Ins 49
5.5.2 Binary Trees for LOTOS Processes 57
5.6 Entities, Components and Bound Use Case Map 58
5.6.1 Principle of Translation for bound UCM 58
5.6.2 Bound UCM with Stub and Plug-ins 62
5.6.3 Nested Components 63
5.6.4 Use of Empty Points 64
5.7 Design of Ucm2LotosSpec 65
5.7.1 XML Representation for UCMs 65
5.7.2 Internal Representation of the UCMs in *Ucm2LotosSpec* 67
5.7.2.1 Classes Diagram for *Ucm2LotosSpec* 67
5.7.2.2 Internal Representation for UCM paths 67
5.7.3 Outline of the Algorithm 69
5.8 Degree of Automation 77
5.9 Conclusion 77

Chapter 6 Ucm2LotosScenario: Automatic LOTOS Scenario Generation from Use Case Maps 79
6.1 Overview 79
6.2 Design of Ucm2LotosScenario....80
6.2.1 Translation of Basic Path Element 80
6.2.1.1 Start Point, End Point and Responsibility 80
6.2.1.2 OR-Fork....81
6.2.1.3 OR-Join....81
6.2.1.4 AND-Fork and AND-Join (Synchronization) 82
6.2.1.5 Waiting Place....82
6.2.1.6 Stub....83
6.2.1.7 Loop....84
6.2.2 Scenarios for Unbound UCMs 86
6.2.3 Scenarios for Bound UCMs....86
6.3 Basic Algorithm for *Ucm2LotosScenario* 87
6.4 Comparison with *UCMNav* 88
6.5 Eliminating Unfeasible Scenarios 89
6.6 Conclusion....90
Chapter 7 Case Study: Wireless Intelligent Network Location Based Service System 92
7.1 Initial Requirements of WIN LBSS and UCM Presentations 92
7.1.1 Location Based Services: General Description 92
7.1.2 Fleet and Asset Management (FAM) 93
7.1.3 Enhanced Call Routing (ECR) 97
7.2 LOTOS Specification for Unbound UCM....99
7.3 Bound UCM for WIN LBSS 100
7.3.1 Bound UCM for Fleet and Asset Management 100
7.3.2 Bound UCM for Enhanced Call Routing 102
7.4 LOTOS specification for Bound UCMs....103
7.5 Message Sequence Charts for WIN LBSS 103
7.6 Conclusion....106
Chapter 8 Conclusions and Future Work 107

8.1 Contributions 107
8.2 Future Work 108
REFERENCES: 111
ACRONYMS 115
APPENDIX A: 117
APPENDIX B: 129

Chapter 1 Introduction

1.1 Background and Motivation

Since late 1980s, the telecommunication industry has developed dramatically. Architectures, services, functionalities and protocols of telecommunication systems become more and more complex, especially in the context of wireless system. This has led to the need of high quality standards. With a correctly specified standard, the implementation is simplified and possibly even automated; reliance on testing and last minute 'fixes' is reduced; inter-working between different implementations has a better chance; standard updates can be traced to affected code [Lo00].

The precise specification and accurate verification and validation of new telecommunication products are essential for their successful development and implementation. Currently, in the early stages of the design and standardization process, communication features, services, functionalities and protocols are described using informal operational and declarative descriptions, tables, and visual notations. As these descriptions evolve, they quickly become error-prone and difficult to manage. This approach has been found to have the following potential problems [Am01]:

- In the early stage of design, the focus should be on system and functional views. However in this approach, it is on details, which may be at a lower level of abstraction or may belong to later stages of the design process. Consequently, these irrelevant details hide many requirements and obstruct high-level design decisions.
- This approach is insufficient to describe a complex telecommunication system or its services. Inadequate descriptions may hide ambiguities, inconsistencies or interactions between levels of abstractions of a given service, or between services. It is difficult to detect these with conventional inspection methods, and they often remain hidden until errors are revealed after implementation. At this point, late correction can be very costly.

- Imprecise standard documents may be interpreted differently by different implementers and this may cause interpretation problems between different implementations.

Hence, it is necessary to develop new methodologies to describe and design telecommunication systems in a more robust way. Formal Description Techniques (FDTs) such as LOTOS, SDL and Estelle, have been known in the standards world for some time. They provide specification, validation and verification methodologies for improving standard quality as size and complexity of software systems grow. But there is still a gap between the stages where services are described informally and the first formal specification of the system [Am01].

To describe and design telecommunication system more precisely, a new methodology, called *Specification-Validation Approach with LOTOS and UCMs* (*SPEC-VALUE*) [Am01], was proposed by Daniel Amyot. *SPEC-VALUE* aims to improve the maturity of design processes based on formal specifications by introducing a semiformal description (UCM) between informal requirements and design-oriented formal specifications (LOTOS) [Am01].

In this thesis, we implement and automate a crucial part of the *SPEC-VALUE* methodology and we demonstrate its application for producing a protocol for the Location Based Service in the Wireless Intelligent Network. Our automatic approach is a contribution towards assisting in the development of new telecommunication systems or services. This automatic approach can also free the designers from the complexity and difficulty of LOTOS at least in part. In general, this automation improves maturity of SPEC-VALUE according to FM-CMM model.

1.2 Three Stages in the Standardization Process

As suggested by the I.130 and Q.65 methodologies [ITU-I130][ITU-Q65], the process of design and standardization of telecommunication systems and services usually consists of three major stages:

1. Stage 1 specifies the overall requirements, service description, and main functionalities from the service subscriber and user's standpoint;
2. Stage 2 identifies the functional capabilities, functional entities and information flows needed to support the service described in Stage 1; Protocol scenarios are expressed in the form of sequence diagrams or Message Sequence Charts (MSC);
3. Stage 3 defines the signaling system protocols and switching functions needed to implement the services described in Stage 1. In other words, detailed description of the protocol will be provided.

Using different visual notations and FDTs in different developing stages can help produce high quality standards for new services in WIN. Also, using the FDTs can lead to the discovery of inconsistencies and omissions between the different stages [Am99D]. In this thesis, different techniques are used for the stages.

1.3 Contributions of the Thesis

In the context that we have described, this thesis offers the following contributions:

Contribution 1: Generate a tool supporting automatic translation from UCMs to LOTOS specifications (*Ucm2LotosSpec* tool)

We propose and implement a method for the automatic generation of LOTOS specifications from UCMs. The key idea of the translation is to establish a relationship between UCM and LOTOS and then automatically translate UCMs to LOTOS specification. The tool supports the translation from Bound and Unbound UCMs to LOTOS specifications.

Contribution 2: Generalize and improve a tool generating LOTOS scenarios from UCMs (*Ucm2LotosScenario* tool)

Using the same mapping as the one used for the translation from UCM to LOTOS, a tool is produced to generate LOTOS scenarios from UCMs. These LOTOS scenarios can be used to get LOTOS traces. Then, Message Sequence Charts are generated from these LOTOS traces. This tool is a generalization and improvement (in fact, a reimplementation) of a tool reported in [Ch01].

Contribution 3: Implement and Automate the methodology *SPEC-VALUE* and apply it to Location Based Service System (LBSS) in the Wireless Intelligent Network

Based on *SPEC-VALUE*, an approach is proposed to produce high quality descriptions for Stage 1 and Stage 2 of WIN LBSS. This approach is based on our two other contributions listed above. We describe the requirements of LBSS in UCM for Stage 1. Then, a LOTOS specification is automatically generated from their requirements in UCMs by using the *Ucm2LotosSpec* tool. Also, LOTOS scenarios describing the behavior of the system are automatically generated from requirements (*Ucm2LotosScenario*). Once the LOTOS specification runs against LOTOS scenarios correctly, LOTOS traces are produced. By inputting LOTOS traces into *LOTOS2MSC* [SteLo], MSCs for stage 2 are generated.

1.4 Organization of the Thesis

This thesis is organized as follows:

Chapter 2: Wireless Intelligent Network

In this chapter, first, we look back upon the background of the emergence of the Wireless Intelligent Network. Then we introduce the architecture, services and standards of the Wireless Intelligent Network.

Chapter 3: Review of Selected Description Techniques

This chapter reviews the description techniques used in the thesis. We present Use Case Maps and LOTOS by introducing their operators and giving examples. Then we introduce Message Sequence Charts briefly.

Chapter 4: Description of Our Approach

This chapter presents one of our contributions. First, we give a brief description of the *SPEC-VALUE* method. Then, based on *SPEC-VALUE*, the automatic approach is described.

Chapter 5: Ucm2LotosSpec: Automatic Generation of LOTOS Specifications from Use Case Maps

This chapter presents the purpose and detailed techniques for automatic translation of UCMs into LOTOS specifications, which is the main contribution of the thesis.

Chapter 6: Ucm2LotosScenario: automatic LOTOS Scenarios Generation from Use Case Maps

This chapter introduces the design of a tool to generate LOTOS Scenarios, which is used for automatic generation of scenarios represented in the form of Message Sequence Charts. This is another contribution.

Chapter 7: Case Study: Wireless Intelligent Network Location Based Service System

In this chapter, our proposed approach is applied to the Location Based Service in the Wireless Intelligent Network. The details are given about how to capture the requirement of LBS in UCMs and generate LOTOS specifications to execute the UCMs and then generate the Message Sequence Charts required in Stage 2.

Chapter 8: Conclusion and Future Work

The last chapter gives the conclusion of our research, and some further work on this topic is proposed.

1.5 Related Work

This research follows other work done at University of Ottawa on specifying and validating telecommunication systems and features, using the semiformal graphical notation Use Case Maps and the formal specification language LOTOS.

Daniel Amyot's work

Daniel Amyot proposed a methodology called *SPEC-VALUE* [Am01] to describe and design distributed systems and telecommunication systems through formal prototyping

and validation. The approach is presented in section 4.1. *SPEC-VALUE* is the methodological foundation of this thesis. We automate and implement parts of this technique in the thesis.

In his master thesis [Am94], Amyot presented a methodology for the semi-automated generation of LOTOS specifications from unbound UCMs. Unbound UCMs were translated manually in the *Timethread Map Description Language* (TMDL) and then LOTOS specifications were generated automatically from TMDL. But TMDL is unfit for the synthesis of specification for complex telecommunication systems. It lacks support for stubs and components [Am01].

Zhimei Yi's work:

In her master thesis [Yi00], Zhimei Yi described CNAP (Call Name Presentation), one of the services in the Wireless Intelligent Network, in the form of UCMs. Then, she generated a LOTOS specification for CNAP manually. Her method is one of the applications of *SPEC-VALUE*. But it needs much manual work and can be improved as shown in this thesis.

Leila Charfi's work:

In her thesis, Leila Charfi [Ch01] introduced a new tool *Ucm2LotosTests,* which was added to the *UCMNav* tool [UCM] for the automatic generation of LOTOS scenarios from UCMs. This tool is similar to the tool *Ucm2LotosScenario* presented in this thesis. But the tools here have some differences. Firstly, their purposes are different. *Ucm2lotosTests* is used for testing purposes while *Ucm2LotosScenario* is intended for Message Sequence Charts generation. Secondly, their implementation is different. *Ucm2lotosTests* is implemented as an extension of the *UCMNav* tool and it will be affected if the tool is changed. *Ucm2LotosScenario* is independent of any tools. It generates LOTOS scenarios from the XML File Format for UCMs. Thirdly, in terms of functionality, *Ucm2LotosScenario* is an improvement and generalization of *Ucm2lotosTests*.

Chapter 2 Wireless Intelligent Network

2.1 Background

Before the 1980s, telecommunications services were switch-based, which means that the logic for controlling telecommunications services was located in traditional switching points. This type of architecture resulted in long development times and large investments to deploy new services since new software releases of the switching systems had to be developed for new services.

Beginning in the early 1980s, a new concept was developed in the evolution of networks: *Intelligent Networks* [IN]. In an Intelligent Network (IN), the logic programs for controlling telecommunications services migrate from traditional switching equipment to computer-based, service-independent elements, known as Service Control Points (SCPs). These service logic programs work in collaboration with the switching equipment based upon a common definition of call models and protocols. They may utilize data resources and physical resources that also reside outside of the switching equipment. This type of architecture provides network operators with an open platform provisioned with generic service components that can interoperate with elements from different vendors, based on published, open-interface standards. Also, it makes it possible to develop new and different services in a cost-effective way.

Wireless telecommunications have grown dramatically since their inception in the early 1980s. Nowadays, wireless subscribers have become more dependent on wireless communications and more demanding of features and functionalities that enhance the value of their basic wireless service. To respond to this increasing demand within an increasingly competitive marketplace, wireless carriers have sought to find better ways to respond to market demands for enhanced services and features.

The development of the Wireless Intelligent Network (WIN) standard is a primary result of this quest for better ways to serve wireless subscribers. WIN, based on IS-41 which is

a well-established standard for wireless telephony in North America, supports the use of IN capabilities to provide seamless terminal services, personal mobility services and advanced network services in the mobile environment [WIN00].

2.2 WIN Architecture

This section describes the WIN architecture from different points of view. Based on [WIN98], the Network Reference Model and the Distributed Functional Plane are introduced in turn.

2.2.1 The WIN Reference Model

The Network Reference Model (NRM) provided by IS-41 is the foundation for the WIN architecture. The network entities of the NRM of WIN are shown in Figure 1. Introduction of new network elements may be required by new services. Generic network entities and the entities used in Location Based Services are show in Figure 1. The entities in shaded are those for Location Based Services.

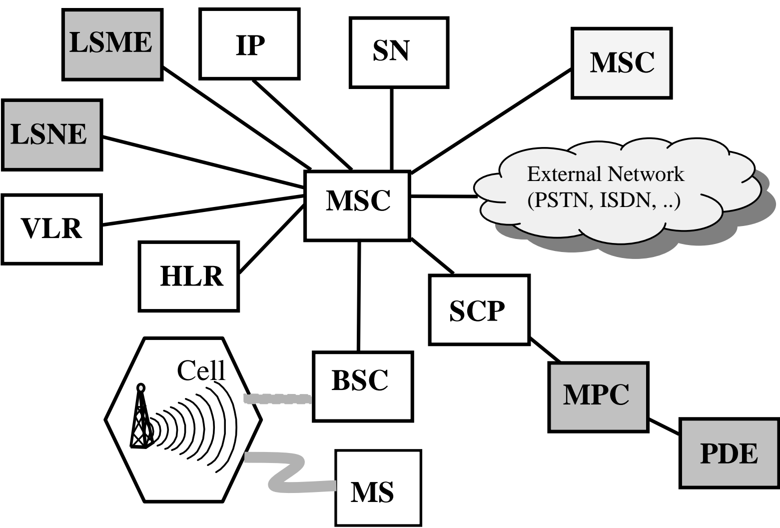


Figure 1 WIN Network Reference Model

As shown in WIN NRM, there are several network entities in WIN [WIN00][WIN98]:

Mobile Switching Center (MSC) – The MSC serves as Service Switching Point (SSP). In the IN, SSP provides the switching function in the network. The Mobile Switching Center provides this function in WIN.

Service Control Point (SCP) – The SCPs are centralized elements in the network. They act as real-time databases and transaction processing system to provide service control and service data functionality. The SCP provides the mechanism for new services independent of the switching system. This mechanism accelerates and simplifies the creation and deployment of new services.

Location Registers – These are used to supplement MSCs with subscriber information. The number of users that a MSC supports changes as roamers move in and subscribers move to other MSCs. The database of active users changes very dynamically. A MSC cannot have one database only for all of its potential users. Two location registers help to solve this problem:

> **Home Location Register (HLR) –** Each subscriber is associated with a single HLR, which retains the subscriber's record. Information (e.g., profile information, current location, authorization period, etc.) on a roamer is obtained from that subscriber's HLR.
>
> **Visitor Location Register (VLR)** – A VLR maintains the subscriber information for visitors or roamers to a MSC. Every MSC or group of MSCs will have a VLR.

Mobile Station (MS) – A MS is the terminal equipment. When the subscriber roams to another switch, the VLR queries the subscriber's home HLR to get information about that subscriber. It provides the users with the capabilities to access network services.

Intelligent Peripheral (IP) – The IP is an entity that performs specialized resource functions such as playing announcements, collecting digits, performing speech-to-text or text-to-speech conversion, recording and storing voice messages, facsimile services, data services and so on.

Service Node (SN) – The SN is an entity that provides service control, service data, specialized resources and call control functions to support bearer related services. It is to accommodate implementors that for any reason do not want to implement MSC, SCP or IP. It is superfluous if MSC, SCP and IP are properly implemented.

Position Determining Entity (PDE) – The PDE determines the precise position or geographic location of a wireless terminal when the MS starts a call or while the MS is engaged in a call.

Mobile Positioning Center (MPC) – The MPC serves as the interface to the wireless network for the location network. The MPC is the entity that retrieves, forwards, stores, and controls position data within the location network. It can select the PDE(s) to use in position determination and forwards the position to the requesting entity or stores it for subsequent retrieval. The MPC may restrict access to position information.

Location Service Message Entity (LSME) – The LSME could be the application that runs a Fleet Management or Information service, or it could be the wireless network operator's Network Management center.

Location Service Network Entity (LSNE) – The LSNE routes and processes the voice band portion of those calls that are subjected to location-based routing.

2.2.2 WIN Distributed Functional Plane

The WIN NRM describes WIN architecture in terms of network entities while the WIN Distributed Functional Plane (DFP), based on the ITU-T Q.1224 recommendation for IN CS-2, defines the WIN architecture in terms of functional entities (FEs). Each FE performs distinct actions in the network and cooperates with others. Several FEs can be included in a single network entity. FEs can be grouped into four categories [Win98][Lo02]: Management related FEs, Service related FEs, Call related FEs and Wireless Access related FEs. Management related FEs, call related FEs and service related FEs in WIN DFP are the same as the ones in IN DFP. Wireless access related FEs in WIN DFP provide functionality specifically related to wireless access and mobility. The relevant portion of the WIN Distributed Functional Model is shown in Figure 2 [Win98][Lo02].

We will not mention the Management related FEs, which are not relevant to the subject of this thesis.

Service related FEs includes:

- Service Control Function (SCF), Service Data Function (SDF)

Call related FEs includes:

- Service Switch Function (SSF)
- Call Control Function (CCF)
- Specialized Resource Function (SRF)

Wireless Access related FEs include:

- Location Registration Function (LRF)
- Authentication Control Function (ACF)
- Mobile Station Access Control Function (MACF)
- Radio Control Function (RCF)
- Radio Terminal Function (RTF)
- Radio Access Control Function (RACF)

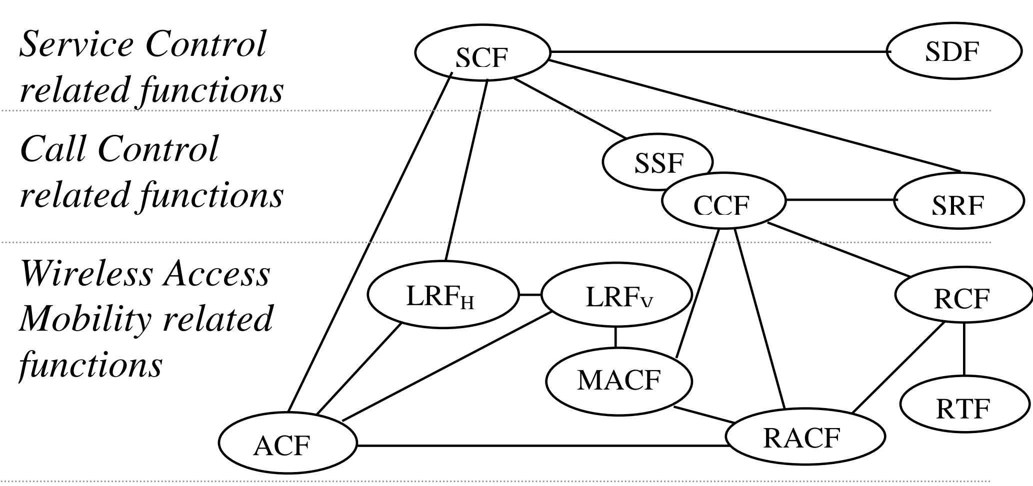


Figure 2 WIN Distributed Functional Model

2.3 WIN Standard

The Wireless Intelligent Network (WIN) standard is an open industry standard that enables equipment from different suppliers to interoperate successfully, and allows

automatic roaming between various networks. The following three WIN phases have been or are being developed by the TIA (Telecommunications Industry Association) TR-45 Engineering Committee:

- WIN Phase I defines the architecture, provides the first batch of capabilities for basic call origination and call termination, and supports basic services such as Calling Name Presentation (CNAP), Incoming Call Screening (ICS) and Voice Controlled Services (VCS) [TIA/EIA/IS-771].
- WIN Phase II adds the second batch of capabilities, and supports charging related services such as Prepaid Charging (PPC), FreePhone (FPH), Premium Rate Charging (PRC), and Advice of Charging (AOC) [TIA/EIA/IS-826][TIA/EIA/IS-848].
- WIN Phase III will add the third batch of capabilities, and support Location Based Services (LBS) such as Location Based Charging (LBC), Fleet and Asset Management (FAM), Location Based Information Service (LBIS), and Enhanced Calling Routing (ECR). This phase is under development. The standard documentation will be published as TIA/EIA/IS-843.

In this thesis, Location Based Services will be an example of applying our automatic approach to specify the new telecommunication services precisely in the early stages of the design and standardization process. Chapter 7 gives details of this case study.

Chapter 3 Review of Selected Description Techniques

3.1 Use Case Maps

Use Case Maps [Bur96][Bur98] is a scenario based semi-formal and visual notation. It aims to capture operational requirements of communicating and distributed systems. It describes scenarios by using ***UCM paths*** that causally link ***responsibilities***. ***Responsibilities***, which can be bound to underlying structures of abstract components with the evolution of the system under study, represent actions, operations, tasks and functions to be performed, messages to be sent or received and so on. Causal relationships represent causal orderings of responsibilities that may be in sequence, as alternatives or in parallel. ***Components*** are functional entities of the system under study. They can be software entities such as objects, processes, databases, and servers as well as non-software entities such as hardware. In this thesis, the word *entity* is used to refer to system entities and the word *component* to refer to the representation of entities in Use Case Maps.

UCMs can be hierarchical. Top-level UCMs are called root maps. All levels of UCM can include some containers (called stubs) for sub-maps (called plug-ins) [Am99B]. Plug-ins can be used and reused in appropriate stubs. A map including a stub is called the ***parent map*** of the plug-ins that can be contained in this stub.

There are two kinds of stubs [Am99B].

- **Static stubs**: represented as plain diamonds. They can contain only one plug-in
- **Dynamic stubs**: represented as dashed diamonds. They can contain several plug-ins, whose selection is determined at run-time according to a selection policy.

In this thesis, static stubs are treated as special cases of dynamic stubs; that is, static stubs are dynamic stubs that can contain only one plug-in.

UCMs can represent systems at different levels of abstraction [Bur96]. A UCM can describe features of a system in general terms at a very early stage even when the architecture of the system is unclear. At such a stage, UCMs are called Unbound UCMs since no components are defined. Unbound UCMs are very useful in Stage 1 description of service functionalities, which focuses on causality and responsibilities without reference to architecture or components [Am99A].

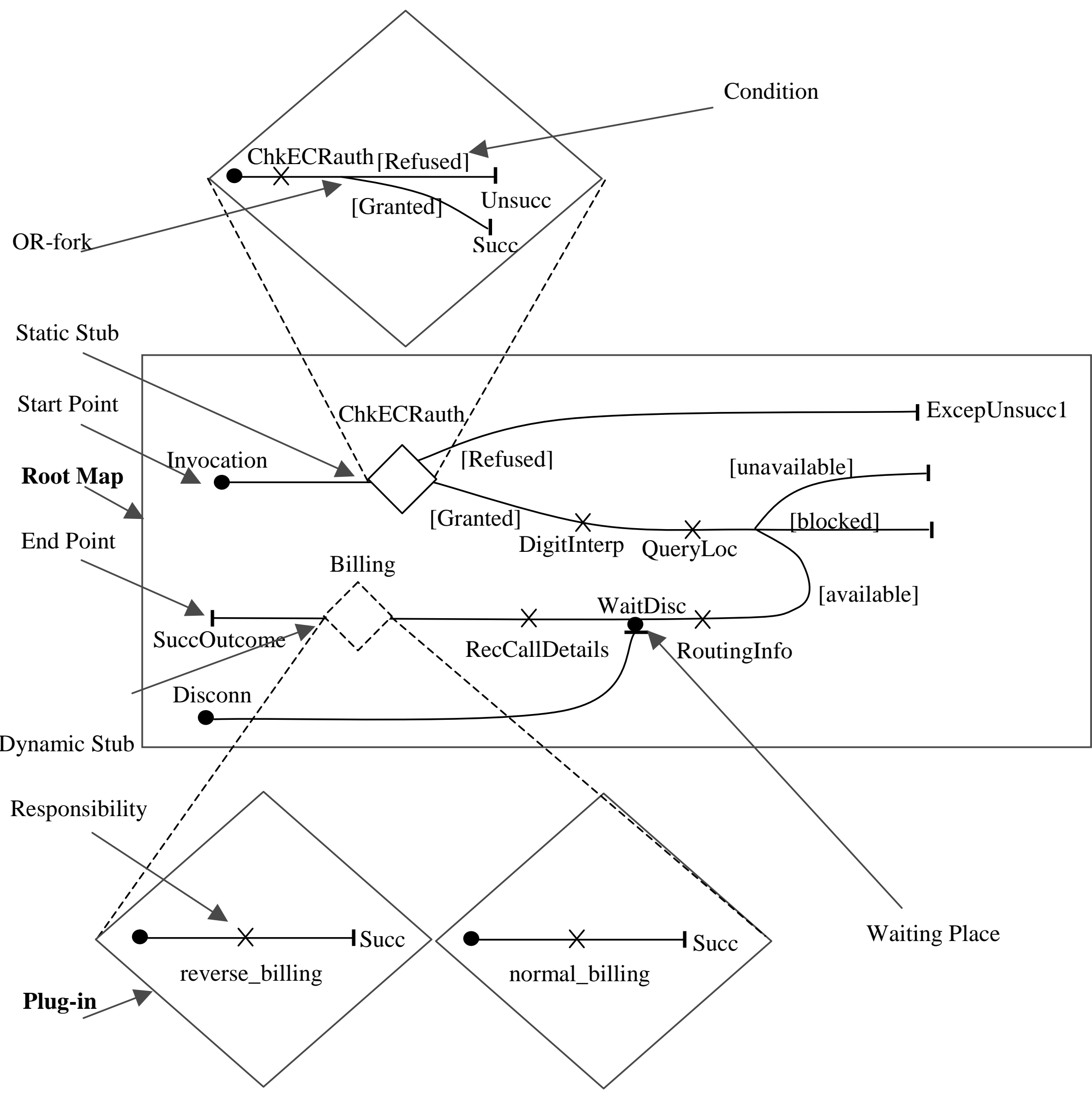


Figure 3 An Unbound UCM for the ECR Service

For example, the ECR (Enhanced Call Routing) service of WIN LBSS in Stage 1 can be represented in the UCMs in Figure 3, which is an Unbound UCM. This UCM will be described fully in section 7.1.3. Figure 3 also includes information on UCM terminology. UCMs can also represent a system where all the responsibilities are refined and assigned to specific components. UCMs with components are called bound UCMs. Therefore, UCMs are very suitable to bridge the gap between stage 1 and stage 2, where a tentative distribution of system behaviors over a structure is being introduced [Am99A][Am99C]. The UCMs in Figure 4 is a Bound UCM for the ECR service and describes the ECR service where responsibilities have been assigned to different components.

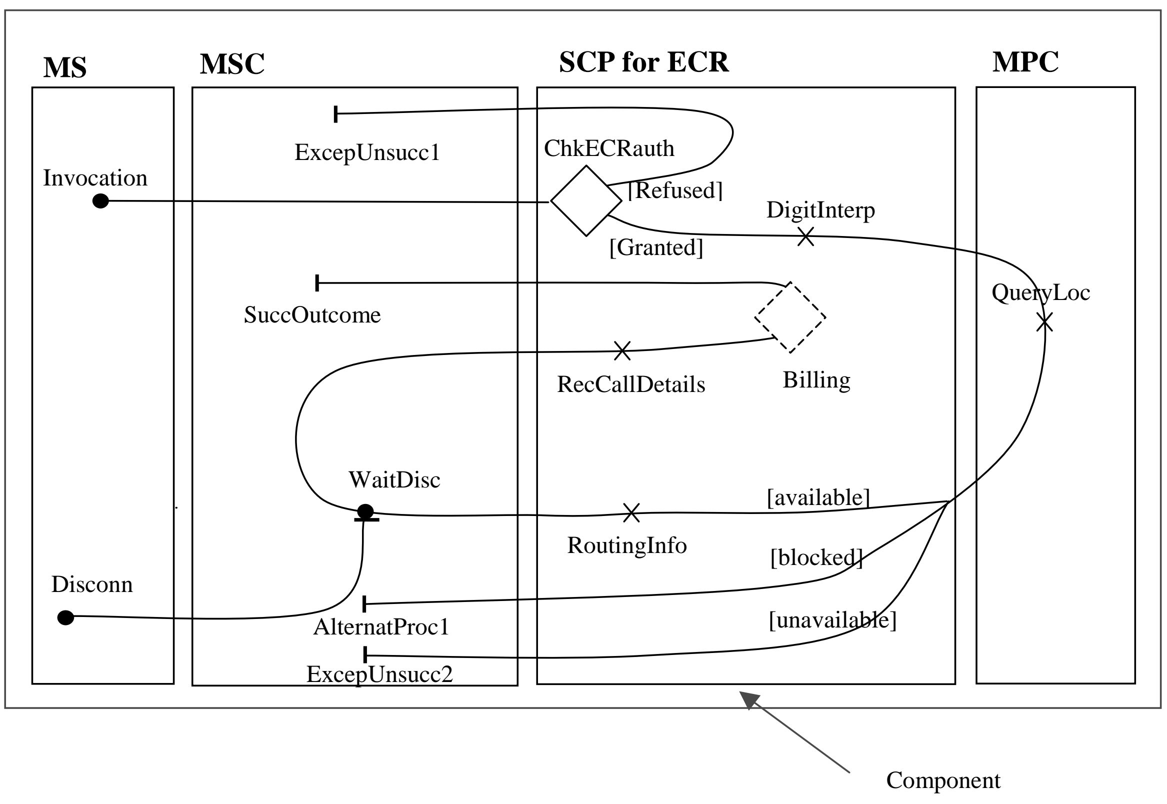


Figure 4 A Bound UCM for the ECR Service

Basic UCM notation elements, including start point, end point, responsibility and component, have been shown in Figure 3 (Unbound UCM) and Figure 4 (Bound UCM). Other UCM path elements are presented in Table 1 (UCM Basic Notation).

Name	**Notation**	**Description**	**Name**	**Notation**	**Description**
OR-Fork		A single path segment splits into several alternative path segments	OR-Join		Several alternative path segments join in to a single path
And-Fork		A single path segment splits into several concurrent path segments	And-Join		Several parallel or concurrent path segments synchronize into a single path
Abort		Top path aborts bottom path	Timer	Timeout path	If a timeout occurs, the path goes on the timeout path.
Waiting Place		Top path waits a triggering event at some point for carrying on the path	Loop		Part of path can be repeated before carrying on the rest of the part

Table 1 UCM Basic Notation

3.2 LOTOS

LOTOS (Language of Temporal Ordering Specifications) [OSI89] is an algebraic specification language and a FDT standardized by ISO for the formal specification of open distributed systems. LOTOS is executable and its models allow the use of different validation and verification techniques. Several tools can be utilized for automating these techniques (e.g. LOLA, ELUDO). Therefore, LOTOS can be applied in all stages of standardization and design of telecommunication systems. The LOTOS notation has two components:

Behavior Notation

- Based on Milner's Calculus of Communicating Systems (CCS) [Mi89] and Hoare's Communicating Sequential Processes (CSP) [Ho85]
- Deals with the description of process behavior and interactions (control flow)

Abstract Data Type Notation

- Based on the formal theory of abstract data types, especially the approach of equational specification of data types
- Inspired by ACT ONE [Ehr85]
- Deals with the description of data structures and value expressions (data flow)

3.2.1 Processes, events and gates

In LOTOS, a distributed system is described as a *process*, possibly consisting of several sub-processes. In general, a LOTOS specification describes a system via a hierarchy of process definitions. A process is an entity capable of performing internal, unobservable actions, and to interact with other processes (forming its environment) by *events*. Events imply process synchronization and they are atomic. An event is considered as occurring at an interaction point called *gate*. Figure 5 represents the structure of a typical LOTOS process that represents a system with two sub-processes. Process 1 (and the system) communicates with the environment via a gate, and so does Process 2. Process 1 and 2 communicate with each other by gate 3. Gate 3 is *hidden*.

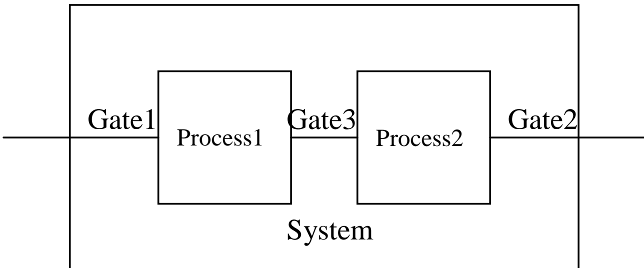


Figure 5 A LOTOS Process

3.2.2 Basic LOTOS

Basic LOTOS is a subset of the language employing a finite alphabet of observable actions. Data types cannot be specified. The syntax of basic LOTOS behavior expressions is given in Table 2 below (where B, B1, B2 are variables for behavior expression, g, g1, … are variables for gate names and p is a variable for process name). From Table 2, we know that basic LOTOS includes nullary, unary, and binary operators.

Stop, exit and process instantiations are basic behavior expressions. Other behavior expressions can be formed by combining internal actions, actions on gates and behavior expressions by means of operators.

Name	Syntax	Meaning
Inaction	stop	Process stops
Action prefix (unobservable, internal)	i; B	Internal action is offered before B
Action prefix (observable)	g; B	Action on gate *g* is offered before B
Choice	B1 [] B2	Actions in B1 or in B2 are offered
Parallel composition (general case)	B1 |[g_1, …, g_n]| B2	B1 and B2 synchronize on gates g_1, …, g_n
Parallel composition (pure interleaving)	B1 ||| B2	B1 and B2 interleave
Parallel composition (full synchronization)	B1 || B2	B1 and B2 synchronize fully
Hiding	hide g_1, …, g_n in B	Gates g_1,..,g_n are hidden in B
Process definition	p[g_1, …, g_n]	P is a process with parameters
Successful termination	exit	Process exits
Sequential composition (enabling)	B1 >> B2	If B1 exits, B2 follows
Disabling	B1 [> B2	B2 can disable B1
Comment	(* …… *)	

Table 2 LOTOS Basic Notation

3.2.3 Data types

The representation of values, value expressions and data structures in LOTOS is derived from the specification language for abstract data types ACT ONE. LOTOS includes the following features for specifying abstract data types:

1. Using a library of previously defined specifications;
2. Combining and extending already existing specifications;
3. Parameterizing specification, and actualizing parameterized specifications;
4. Renaming specifications.

The most basic form of data type specification in LOTOS consists of a *signature*, which gives all the information required to build syntactically correct terms (or value expressions), and possibly a list of *equations*, which are used to state that two syntactically different terms denote the same value. Examples of abstract data type will be seen in Appendix B.

3.2.4 Full LOTOS

A detailed introduction to full LOTOS can be found in [Lo91] and [Bo87]. By using the facilities for the description of data structures and values to give a finer structure to observable actions and process interactions in full LOTOS, we are able to enrich synchronizations with value passing, thus providing inter-process communication. A full LOTOS specification will be shown later.

The concept of LOTOS *trace* will play an important role in this thesis. A trace is a finite-length sequence of LOTOS actions, which represents a scenario of possible interactions between the system and its environment [GaLo].

3.2.5 LOLA

LOLA (LOtos LAboratory) [QFM87][QPF89][PL91] is a transformational and state exploration tool for the simulation and testing of LOTOS specification. In LOLA, test cases are specified as LOTOS processes. They are composed in parallel with the LOTOS specification, synchronizing in the union of the gate sets of both. Each test process contains a termination event after all the events. For example, one test process is shown

in the following box. It represents a WIN FAM scenario where a subordinate manually initiates sending of location status information to the supervisor. The event **Scenario** in the process is the termination event. The successful termination of a test in a given execution consists in reaching a state where the termination event (**Scenario**) is offered. A deadlock is an unsuccessful termination [QFM87].

In this thesis, LOLA is used for the simulation and testing of LOTOS specifications.

```
Process example [Start, Resp, End, Scenario]: exit: =
  Start! UpdPos;
  Resp! GetLocation;
  Resp! UpdLoc;
  Resp! NotifySV;
  End ! Succ;
  Scenario;
  exit
EndProc
```

3.3 MSC

MSC (Message Sequence Chart), standardized by ITU-T [ITU-Z120], is a graphical and textual language for the description and specification of the interaction scenarios between system entities. Its main application is visualization of the communication behavior of real-time systems, in particular telecommunication switching systems. MSCs provide a clear description of communication behavior between system entities and their environment by means of message exchange. MSCs may be used for requirement specification, simulation and validation, test-case specification and documentation [Am99A]. In this thesis, only the simplest kind of MSC is used, which represents only one system scenario.

In Stage 2 of the standardization, telecommunication services are specified with sequences of messages between different functional entities. MSCs mainly concentrate on message interchanged by communicating entities and their environment [ITU-Z100].

Therefore, MSCs are used for Stage 2 to describe information flows that are needed to support the service described in Stage 1 and defined in details by Stage 3.

At the University of Ottawa, a tool called *Lotos2Msc* was developed, that allows producing MSCs for given LOTOS traces [SteLo]. This tool was used for the work of this thesis, as will be shown in Chapter 4.

Chapter 4 Description of Our Approach

The three stages in the standardization process have been discussed in Section 1.2. In this chapter, the *SPEC-VALUE* is briefly described. Then an approach is proposed to produce correct Message Sequence Charts for Stage 1 and Stage 2 of new telecommunication services.

4.1 Overview of *SPEC-VALUE*

As mentioned above, this thesis places itself in the framework of *SPEC-VALUE* methodology. *SPEC-VALUE* is an iterative and incremental approach (in spiral form) that allows rapid prototyping of abstract behavior and test case generation directly from scenarios [Am01]. Figure 6 (adapted from [Am99D]) presents the approach. In SPEC-VALUE, firstly, the description of system structures ① and scenarios ② are obtained from the requirements. They can be obtained separately. A structure contains the abstract system components, which are mostly software but can also be hardware. Scenarios are represented in Use Case Maps, which focus on the causal relationships among the responsibilities that compose services and large-grain functionalities. The responsibilities defined in the UCMs are then allocated to the components in the selected underlying structure ③. Each component will perform the responsibilities allocated to it. Next, the scenarios are combined to construct a LOTOS specification (manually in Amyot's thesis) ④, which becomes the executable prototype enabling formal validation of abstract behaviors of the system under study ⑥. Simultaneously, test cases can be generated from the individual scenarios (manually in Amyot's thesis) ⑤ to ensure that the specification conforms to each intended functionality [Am01]. In this thesis, step ④ (LOTOS specification construction) and part of step ⑤ (test cases generation) is automated by developing *Ucm2LotosSpec* to construct LOTOS specifications and developing *Ucm2LotosScenario* to generate LOTOS scenarios from which test cases can be obtained.

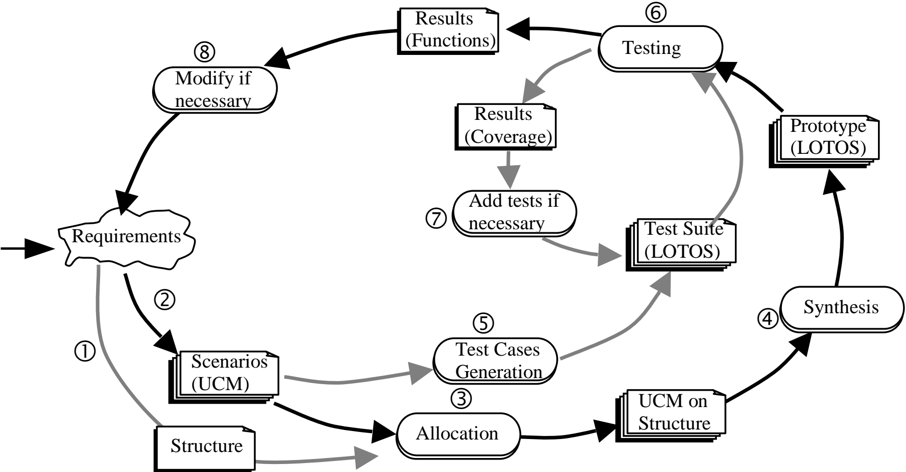


Figure 6 ***SPEC-VALUE*** **(adapted from Figure 2 of [Am99D])**

4.2 The Proposed Approach

As suggested in *SPEC-VALUE*, in our proposed approach, UCMs, which can fill the gap between informal requirements and formal specifications, are used to capture the requirements in Stage 1. Then, the new services are formally specified in LOTOS and Message Sequence Charts are generated in Stage 2. The proposed approach not only can assist in the development of new telecommunication service standards but also can help reduce the early stage design errors by generating scenarios showing possible system behaviors.

Figure 7 presents the essential steps of the proposed approach. Grey boxes represent the tools used in this approach. Tools in double boxes are the ones developed in this thesis.

(1) Initial Requirements of new telecommunication services are described in UCMs. By the use of UCM Navigator [UCM], the UCMs representing the requirements of new services are drawn and saved in XML format.

(2) Using *Ucm2LotosSpec* (introduced in Chapter 5), the requirements represented in UCMs are translated into LOTOS specifications when analysis needs to be performed.

(3) Using *Ucm2LotosScenario* (introduced in Chapter 6), LOTOS scenarios are generated automatically from the UCMs.

(4) LOLA (LOtos LAboratory), which can help to analyze the behavior of a system specified in LOTOS by executing and testing LOTOS specifications, is used to simulate the generated LOTOS specification and test it against the generated LOTOS scenarios.

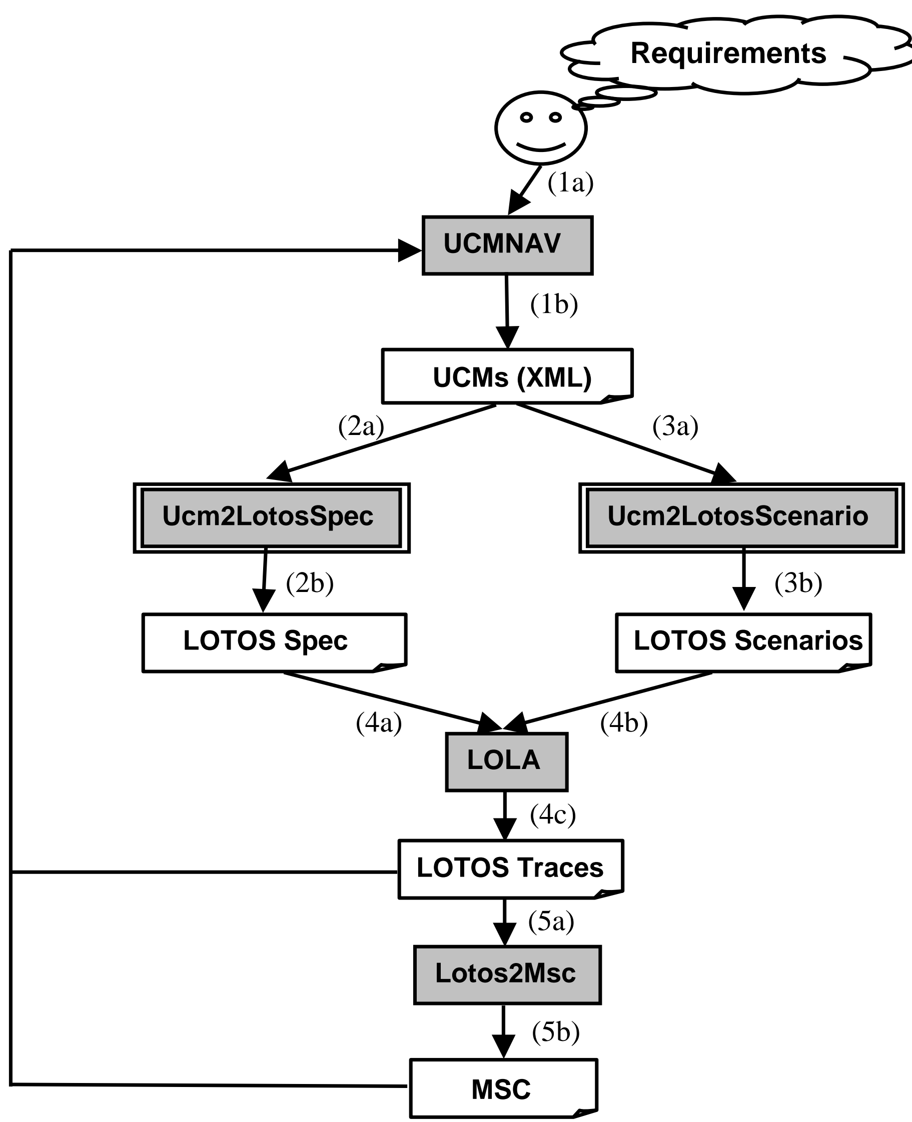


Figure 7 The Proposed Approach

If there is any problem in the simulation or testing of the LOTOS specification, UCMs have to be corrected and step (1)(2)(3) and (4) are redone until the specification is valid.
As the description of the services evolves, more detailed information about actions performed and messages exchanged between components is added to the UCMs. Some responsibilities will be refined and Steps (1) (2) (3) (4) will be repeated.
LOTOS traces are generated by LOLA when all the LOTOS scenarios run against the LOTOS specification successfully.
(5) Message Sequence Charts, which describe the scenarios of the new telecommunication services, are produced by *Lotos2Msc* [SteLo]. Again, these can be inspected and compared with the intended behaviors. Any correction may require repetition of Steps (1) - (5).

A question could be raised at this point regarding step 4: since both traces and specifications are derived from the same set of UCMs, how can it happen that a specification cannot execute a trace? This is mainly due to the complexity of UCM integration.

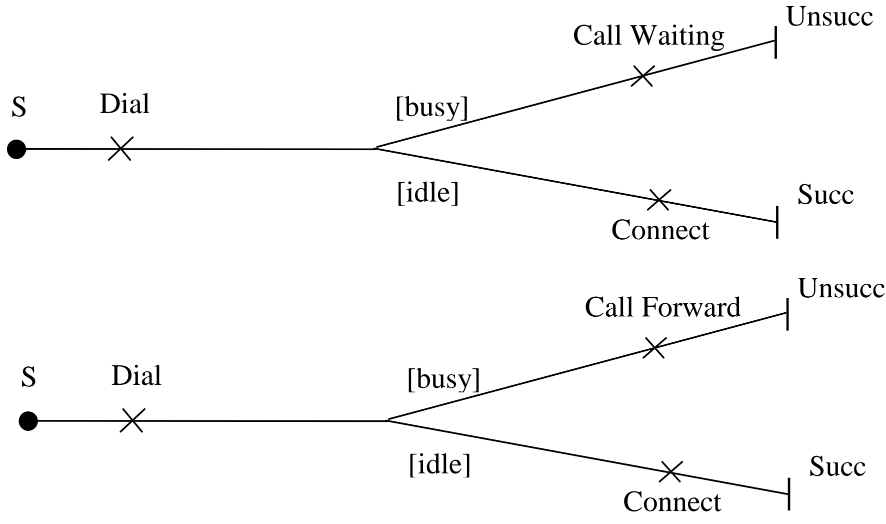


Figure 8 Nondeterminism in UCMs

Given a simple example in Figure 8, suppose that a system under study has two scenarios. One is that when a caller dials and the line is busy, the call is forwarded. Another is that when a caller dials and the line is busy, the call waiting function is invoked. If the system is described in two UCMs as in Figure 8, when the LOTOS specification and LOTOS scenarios are obtained from these UCMs, then by running either scenario with the LOTOS specification, a *may pass* result will be obtained, rather than a *must pass*. A problem of nondeterminism in the LOTOS specification was found, i.e. which function, call waiting or call forward, should be invoked after the event *dial*? This implies that there is an error in the UCM design. In fact, in this specific case, the error found is a feature interaction. [Ca93]

Therefore, although the LOTOS scenarios and LOTOS specification are derived from the same set of UCMs, by running the LOTOS scenarios against the LOTOS specification, design errors may still be found.

It should be pointed out that the main purpose of generating LOTOS scenarios in this work is to generate Message Sequence Charts. Other techniques to validate the design should be used in addition.

4.3 Conclusion

This chapter first reviewed the *SPEC-VALUE* methodology (section 4.1). Then the approach was described (section 4.2), which is based on ideas of the *SPEC-VALUE* methodology. In the following chapters, the tools that support the approach are discussed and the application of the approach in WIN LBSS project is shown.

Chapter 5 Ucm2LotosSpec: Automatic Generation of LOTOS Specification From Use Case Maps

5.1 Purpose

Use Case Maps help designers capture the requirements of the system and to visualize the system design in early stages before implementation starts. Use Case Maps mostly focus on system functionality, and they are easily learnable and understandable. However, Use Case Maps have no formal semantics. They are not created for presenting detailed information such as protocol messages. Therefore, they are not suitable for detailed analysis and design. On the other hand, LOTOS provides a solid platform for detailed analysis of functionality and protocols. It permits to perform validation and verification. Although LOTOS has many benefits, its complexity keeps designers from using it.

Both Use Case Maps and LOTOS are used for the WIN project and similar work was done in other projects by other people [Am01][Ch01][An00]. Initially, UCMs were used to describe the system and the LOTOS specification was generated (manually) for validation and verification. In practice, some problems come up. One of them is that the LOTOS specification doesn't have much correspondence with the UCM model. Traceability suffers and if the LOTOS specification is changed, it is difficult to make the corresponding change in the UCM model. Therefore, once the design evolves in more detail, UCMs are put aside and not used any more. It is necessary to improve our methodology to use these two techniques to the best of their potential. The key idea is to establish a semantic relationship between UCM and LOTOS and then automatically translate UCMs to LOTOS specifications as completely as possible in order to free designers from the complexity and difficulty of LOTOS. The generated LOTOS specification can be used to verify and validate system designs. If there are any design errors, the UCM model is modified and the LOTOS construction and verification are

redone. This idea was presented first in [Am01] but an automatic translation has not been available until now.

The prerequisite of the automatic translation is a UCM model for the system under study (this is usually a collection of UCMs). The output is a LOTOS specification that captures the functional requirements and the high-level design, where the structure of components may or may not be considered.

5.2 Overview

As mentioned, Use Case Maps are a graphical language for requirement capture and system design at the early stages of system design. Initially, unbound UCMs are used to represent the behavior of the system if the structure is still not clear. As the description of the system evolves, more detailed information such as entities of the system, actions performed are added to UCMs. At this point, unbound UCMs become bound UCMs where all tasks and responsibilities (actions) are assigned to different UCM components. While unbound UCMs describe the functionality of the system, bound UCMs further highlight how and when messages need to be exchanged between entities. Therefore, one can say that unbound UCMs describe a service and bound UCMs describe a protocol. This difference between unbound UCMs and bound UCMs has an impact on the type of LOTOS specification that can be induced. To keep the consistency, unbound UCMs are considered as a special case of bound UCMs, i.e. there is only one component in an unbound UCM.

The UCM notation is mainly composed of path elements and components. Therefore, the translation from UCM to LOTOS will be described in three phases:

- In the first phase, it is shown how to translate into LOTOS from a basic UCM notation, including only basic UCMs path notation elements. (Section 5.4)
- In the second phase, the translation from a UCM model, including stubs and plug-ins, is discussed. (Section 5.5)
- In the third phase, UCM components are taken into account and the method for LOTOS construction from the bound UCM model is presented. (Section 5.6)

The data structures and basic algorithms for implementation of translation are outlined in section 5.7.

Some BNF-like notation used in the following sections is shown in Table 3.

Name	Notation	Description	Example
Optional Element	[ ]	Put an optional fragment in square bracket []	[expr]
Multiple Choice	(|)	Put elements of which one must be used in round brackets, separated by bars	(+|*)

Table 3

5.3 Subset of UCM Notation supported

UCM is a very general notation, with many possibilities. Translating the full notation into LOTOS is perhaps possible, but there would be undesirable effects, such as:

1. Complex translation process
2. LOTOS behavior very difficult to read and to trace back to UCM
3. Having to take decisions in situation where the meaning of the UCM notation is not clear

For these reasons, the tool only accepts UCMs that abide by the following constraints:

1. The UCMs must be constructed from the UCM Notation Element shown in Table 4.

Name	**Notation**	**Name**	**Notation**	**Name**	**Notation**
Start/End Point		OR-Join		Static Stub	
Responsibility		And-Fork		Dynamic Stub	
OR-Fork		And-Join		Waiting Place	

Table 4

The UCM notation *timer*, *abort*, *loop*, *dynamic components and responsibilities* are not supported in this tool.

2. The loop notation is not supported, but a loop can be constructed by using OR-Fork and OR-Join. Section 5.4.8 will give details.
3. In UCM, the number of entry points/exit points of a stub could be different from the number of start points/end points of the plug-in map that it contains. Because of the complexity of implementing this feature, in the UCMs accepted by the tool, the number of start and end points must match in the stubs and related plug-ins.
4. If the UCMs have stubs and plugins, the binding information must be provided. Furthermore, the UCMs cannot include any path that gets out of a stub and loops back to it.

5.4 Basic Path Elements Translation For Unbound UCMs

In this section, basic UCM path elements translation that was originally presented in [Am94] and its implementation are discussed.

5.4.1 Principles of Translation

As mentioned in section 3.2, a LOTOS process is an entity capable of performing internal, unobservable actions, and to interact with other processes (its environment) by *events* occurring at gates. Therefore, a LOTOS process is described with events, each of which consists of a formal gate and optional experiments, and internal actions if there are any. Whenever a process is instantiated, a list of actual gates is used, providing for the reuse of the same process with different sets of gates.

At this point only unbound UCMs are considered, and so it is assumed that there is only one component for the whole UCM model. The LOTOS specification generated from this UCM model only has one top-level process, which may contain several sub-processes.

In the very early stages of system design when there is lack of detailed information, three gates are predefined: Start, End and Resp, which are used respectively for start point, end point and responsibility. When the design of a system evolves and more information is added, start point, end point and responsibilities will be refined. Correspondingly, LOTOS behaviors are modified by replacing labels of start point/end point/responsibilities with triggering event/resulting event/execution-sequence. Once responsibilities are assigned to components, actual channels are used besides predefined ones. This issue will be discussed in later sections.

5.4.2 Start points, end points and responsibilities.

5.4.2.1 Start Point and End Point

A path in UCMs begins at a start point and finishes at an end point. A start point defines preconditions and triggering events for the path. Preconditions can be translated to selection predicates or guarded behavior in terms of LOTOS.

A trigger event can be represented as experiment at gate "start". At the early stages of system design, a start point in a UCM is only described by its name. In this case, start points may be simply represented with a gate "start" with value expressions named after the label of the start point. That is, a start point translated into LOTOS becomes

[Preconditions->] Start! (Triggering Event| name of start point);

(rest of the path*)*

Similarly, an end point stipulates post-conditions and resulting events for the path, which may be used to signal the continuation of the path at its parent stub level. An end point can be translated to the following LOTOS behavior:

*(*head of the path*)*

End! (Resulting Event| name of end point); exit

According to the translation rules for start point and end point, a simple path with one start point and one end point becomes the following LOTOS behavior:

[Preconditions->] Start! (Triggering Events| name of start point);

(body of the path*)*

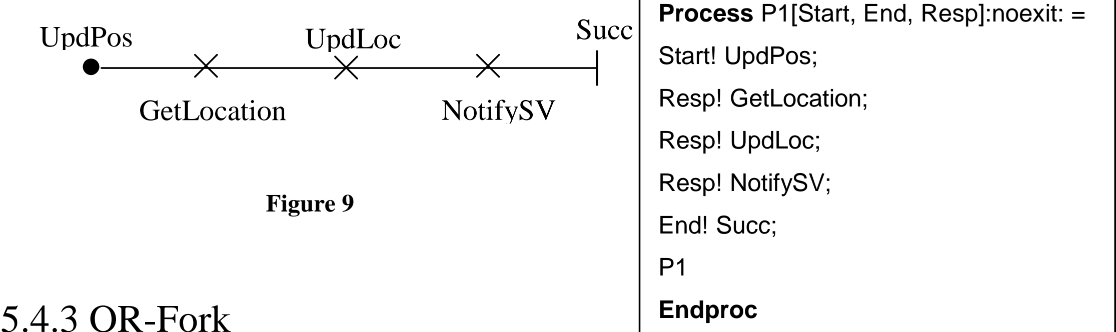
End! (Resulting Event|name of end point); exit

5.4.2.2 Responsibilities

Responsibilities along a path represent actions, tasks and functions to be performed. In the order given, the closer they are to the start point along the path, the earlier they are executed. Sequences of responsibilities can be captured by the LOTOS action prefix operator. Similar to start points and end points, responsibilities are given generic names, without any details about the sequence of actions that need to be performed in order to implement them. The translation of a responsibility can be:

Resp! the label of the responsibility;

Therefore, the use case map in Figure 9 are represented in LOTOS as follows:




```
Process P1[Start, End, Resp]:noexit: =
Start! UpdPos;
Resp! GetLocation;
Resp! UpdLoc;
Resp! NotifySV;
End! Succ;
P1
Endproc
```

Figure 9

5.4.3 OR-Fork

OR-Fork is another common notation element in UCMs. It describes alternative paths and it corresponds to the choice operator in LOTOS in terms of its meanings. An OR-Fork may contain information for branch selection. In this case the OR-Fork can be represented in the LOTOS specification with guarded behaviors.

Figure 10 will be represented in LOTOS as follows:

```
Process P1[Start, End, Resp]:noexit: =
  Start! Invocation;
  Resp! ChkECRauth;
  (
   [Refused] -> End! ExcepUnsucc1;
                stop
   []
   [Granted] -> Resp! DigitInterp;
                Resp! QueryLoc;
                End! AlternatProc;
                stop
  )
Endproc
```

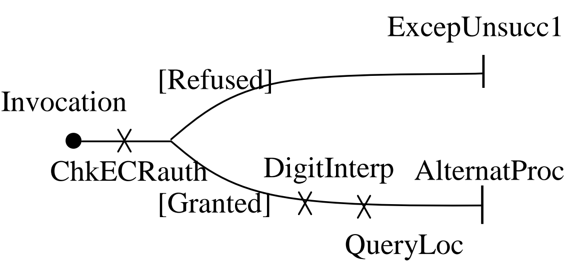
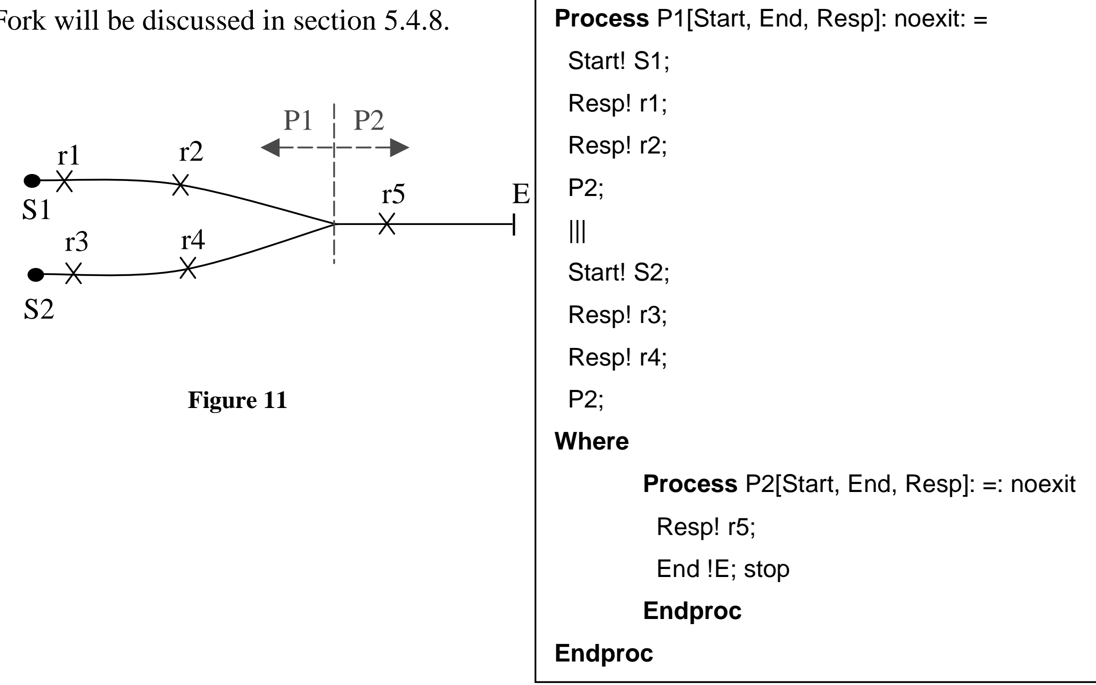


Figure 10

5.4.4 OR-Join

An OR-Join in a Use Case Map means that two or more independent path segments share the sequence of responsibilities that follows the join. If an OR-Join doesn't follow an OR-Fork or AND-Fork in a UCM, the independent path segments before an OR-Join will be considered as concurrent. The cases where an OR-Join follows an OR-Fork or AND-Fork will be discussed in section 5.4.8.




```
Process P1[Start, End, Resp]: noexit: =
 Start! S1;
 Resp! r1;
 Resp! r2;
 P2;
 |||
 Start! S2;
 Resp! r3;
 Resp! r4;
 P2;
Where
      Process P2[Start, End, Resp]: =: noexit
       Resp! r5;
       End !E; stop
      Endproc
Endproc
```

Figure 11

The shared path segment (responsibility r5 and end point E) is considered as a separate sub-process (Process P2) in order to provide modularity and reusability. Section 5.4.8 will show that this is necessary.

5.4.5 AND-Fork

An AND-Fork specifies that two or more path segments must be processed in parallel (interleaving) from some point of a path. For example, in Figure 12, path segment 1 and 2 (as marked) execute independently after r1 completes. This case can be represented by the LOTOS interleave operator (|||). The translation is:

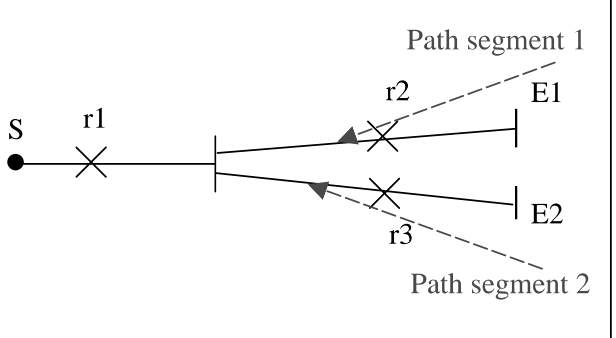


Figure 12

```
Process P1[Start, End, Resp]: noexit: =
 Start! S;
 Resp! r1;
 (
  Resp! r2;
  End! E1;
  stop
  |||
  Resp! r3;
  End! E2;
  stop
 )
Endproc
```

5.4.6 AND-Join

The AND-join is used to describe that two or more path segments that join on a point of the path must complete before the rest of the path can carry on. For instance, in Figure 13, r3 has to wait for the completion of r1 and r2 before it can be executed. The LOTOS parallel composition (|[]|) can be used to describe this behavior. For the sake of uniformity with the OR case, the path segment after the AND-Join is implemented in a sub-process. That is:

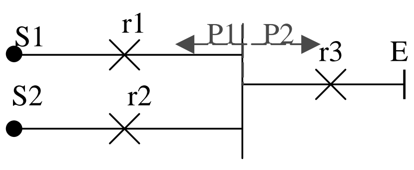


Figure 13

```
Process P1[Start, End, Resp]: noexit: =
hide sync in
(
  (Start! S1;
   Resp! r1;
   sync; P1
   |[sync]|
   Start ! S2;
   Resp! r2;
   sync; P1)
  |[sync]|
  sync; P2
)
Where
        Process P2[Start, End, Resp]: noexit: =
          Resp! r3;
          End! E; stop
        Endproc
Endproc
```

5.4.7 Generic Version of AND-Fork and AND-Join

AND-Fork and AND-Join are special cases of synchronization. AND-Fork is one kind of synchronization that only has one path segment before the synchronization point, while AND-Join is one kind of synchronization that only has one path segment after the synchronization point. Figure 14 shows a more general example of synchronization. In this example, r1 and r2 must be completed before r3, r4 and r5 execute independently. In order to provide modularity, the part of the path following the synchronization point is encapsulated in a separate process. The operation of synchronization can be described in LOTOS by the LOTOS parallel composition (|[]|).

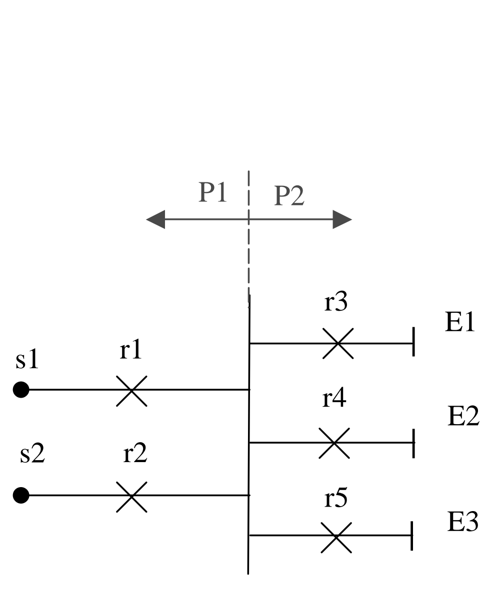
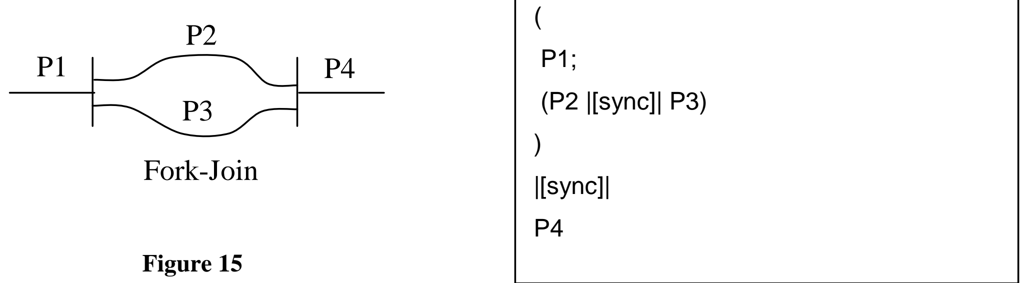


Figure 14

```
Process P1[Start, End, Resp]: noexit: =
hide sync in
(
  (Start! s1;
   Resp! r1;
   sync; stop
   |[sync]|
   Start!  s2;
   Resp! r2;
   sync; stop)
  |[sync]|
  sync; P2
)
Where
        Process P2[Start, End, Resp]: noexit: =
          Resp! r3;
          End! E1; stop
          |||
          Resp! r4;
          End! E2;stop
          |||
          Resp! r5;
          End! E3; stop
        Endproc
Endproc
```

There are some variations of AND-Fork/AND-Join as shown in Figure 15 and Figure 16. The translations of these variations can be obtained by combining the translations of AND-Fork and AND-Join. Note that P1, P2, …Pn in Figure 15 and Figure 16 are sub-processes. They can contain any sequence of responsibilities.

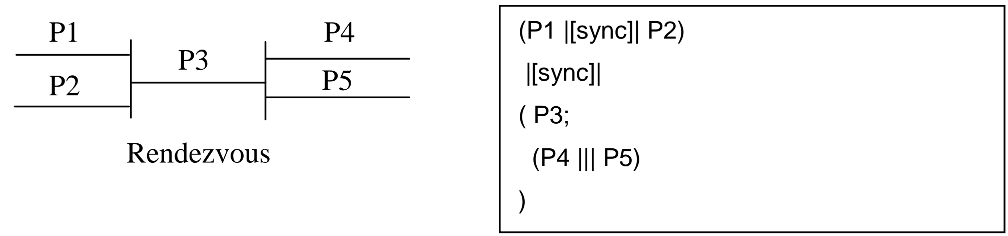
For example, the Fork-Join as in Figure 15, in which each segment can be represented by a sub-process such as P1, P2, P3 and P4, is translated in LOTOS behavior as follows:




```
(
 P1;
 (P2 |[sync]| P3)
)
|[sync]|
P4
```

Figure 15

Similarly, Rendezvous in Figure 16 translated as follows:




```
(P1 |[sync]| P2)
 |[sync]|
( P3;
  (P4 ||| P5)
)
```

Figure 16

5.4.8 OR-Join with OR-Fork or AND-Fork

When an OR-Join follows an OR-Fork or AND-Fork, its meaning depends on the context. In both cases, the shared path segment is considered as a separated sub-process, which follows the actions before it. For example, the UCM in Figure 17 can be translated as follows. The translation for the UCM in Figure 18 is the same except that the interleaving operator (|||) replaces the choice operator ([]).

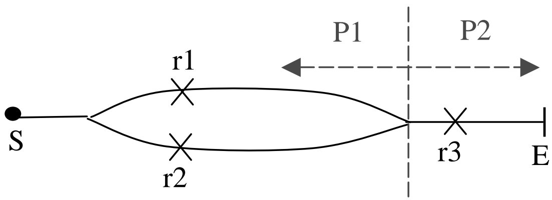


Figure 17

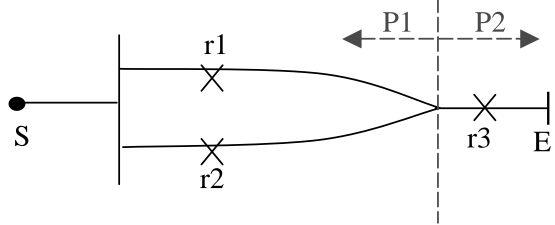


Figure 18

```
Process P1[Start, End, Resp]: noexit: =
 Start! S;
 (
  Resp! r1; P2;
  []
  Resp! r2; P2;
 )
Where
        Process P2[Start, End, Resp]: noexit: =
         Resp! r3;
         End !E;
         stop
        Endproc
Endproc
```

It is necessary to consider the shared path segment as a separate sub-process in order to provide modularity and reusability, especially in some cases. For example, the UCM in Figure 19 has a loop that is constructed by an OR-Join operator and an OR-Fork operator.

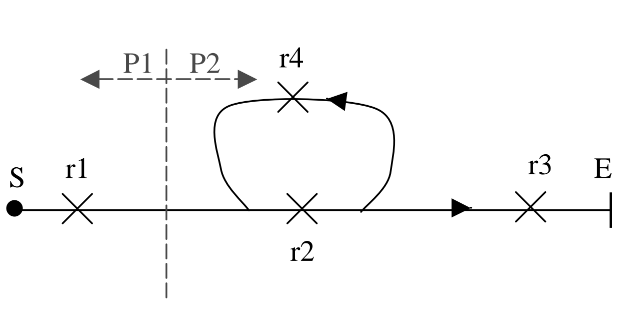


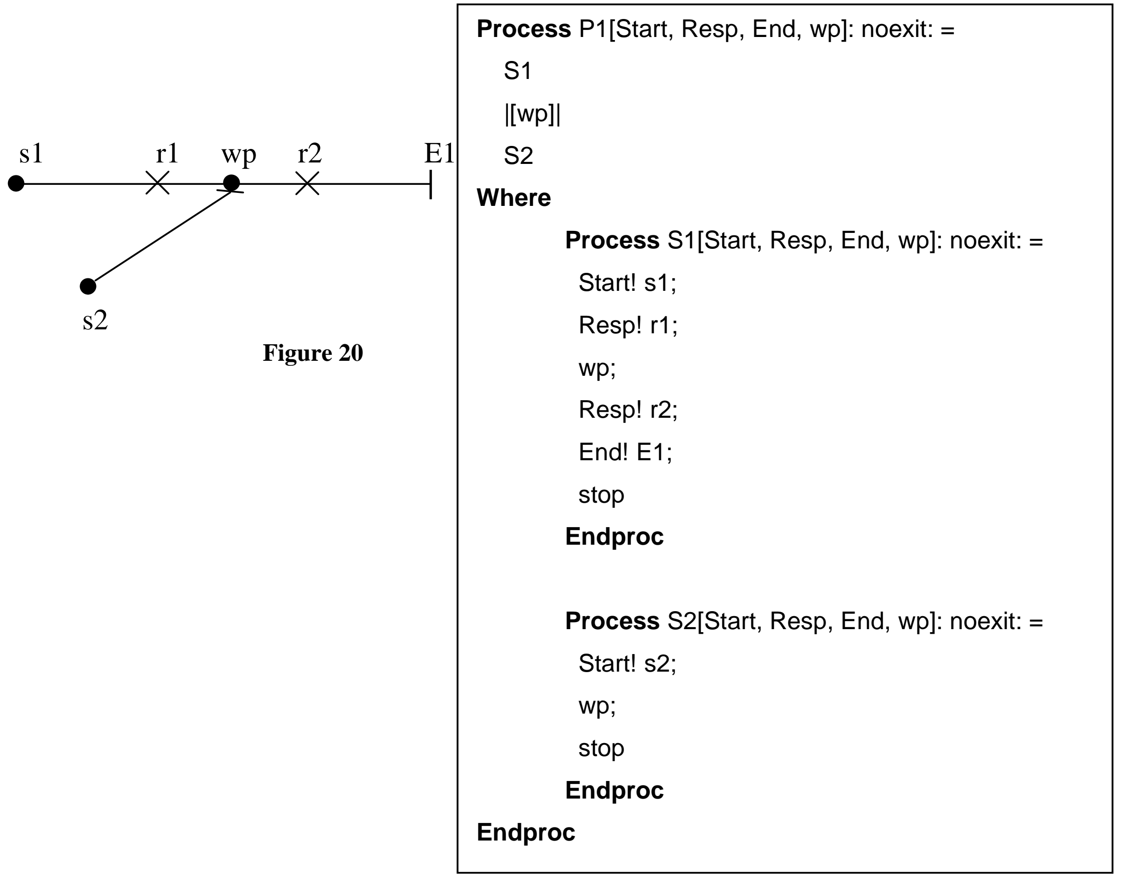
Figure 19

```
Process P1[Start, End, Resp]: noexit: =
 Start! s;
 Resp! r1;
 P2
Where
        Process P2[Start, End, Resp]: noexit: =
         Resp! r2;
         (
          Resp! r4;
          P2
          []
          Resp! r3;
          End! E;
          stop
         )
        Endproc
Endproc
```

5.4.9 Waiting place

A waiting place suspends the execution of an ongoing path until a connected path arrives. There are two cases after triggering the ongoing path. In one case, the connected path ends once it triggers the waiting place. In the second case, the connected path has other tasks to perform after triggering the ongoing path. In both cases, the waiting place is implemented by means of the LOTOS selective synchronization operator; that is, the ongoing path and the triggering path synchronize on the waiting place.

The following UCM shows the first case: the path led by s1 waits for the path led by s2 to arrive to wp in order to resume executing the rest of its path. The LOTOS translation is:




```
Process P1[Start, Resp, End, wp]: noexit: =
   S1
   |[wp]|
   S2
Where
         Process S1[Start, Resp, End, wp]: noexit: =
          Start! s1;
          Resp! r1;
          wp;
          Resp! r2;
          End! E1;
          stop
         Endproc

         Process S2[Start, Resp, End, wp]: noexit: =
          Start! s2;
          wp;
          stop
         Endproc
Endproc
```

Figure 20

The second case is shown in the UCM of Figure 21, r2 must wait for wp, but r3 can proceed independently of wp. Therefore, the translation is:

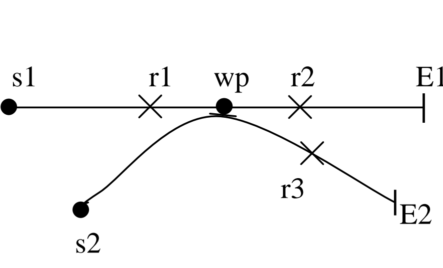


Figure 21

```
Process P1[Start, Resp, End, wp]: noexit: =
   S1
   |[wp]|
   S2
Where
          Process S1[Start, Resp, End, wp]: noexit: =
           Start! s1;
           Resp! r1;
           wp;
           Resp! r2;
           End! E1;
           stop
          Endproc

          Process S2[Start, Resp, End, wp]: noexit: =
           Start! s2;
           (
            wp;
            stop
            |||
            Resp! r3;
            End! E2;
            stop
           )
          Endproc
Endproc
```

5.4.10 Interacting Paths

There are two kinds of interacting paths: synchronous interaction and asynchronous interaction as shown on the left of Figure 22 and Figure 23. The implementation of these interacting paths is the same as the one of their right sides.

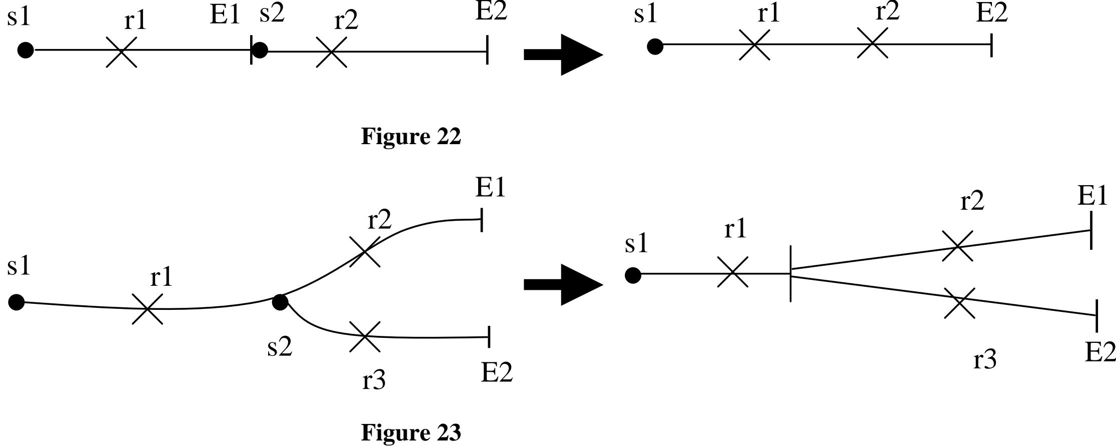


Figure 22

Figure 23

5.5 Translation for UCMs with Stubs and Plugins

In the previous section, basic UCM path notation translation has been discussed. When maps become complex, it is necessary to modularize the UCMs for the sake of readability and reusability. At this point, like procedures in programming languages, the concepts of stub and plug-in are introduced. In this section, the principle of translation for UCMs with stubs and plugins is presented together with a binary tree presention of UCMs for the translation.

When translating UCMs with stub and plug-ins to any other language, two difficult cases have to be dealt with:

1. The plug-in determines the meaning of the parent map. Consider the two cases shown in Figure 24, where there is a static stub with two incoming paths. If the stub has the plug-in on the left, the two incoming path segments are concurrent. If the stub has a plug-in on the right, the two path segments are mutually exclusive.
2. The parent map determines the meaning of the plug-in. For example, in Figure 25, the relationship of two paths in the plug-in depends on the plugin's parent map. When this plug-in is included in the map on the left, the two paths in the plug-in are alternative. When this plug-in is included in the map on the right, the two paths in the plug-in are parallel.

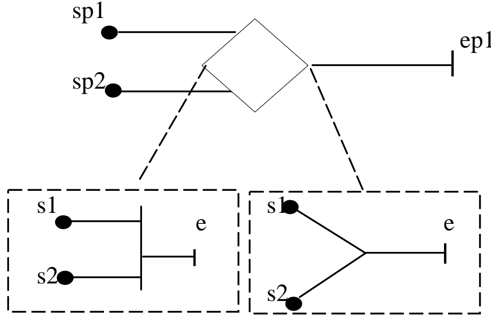


Figure 24

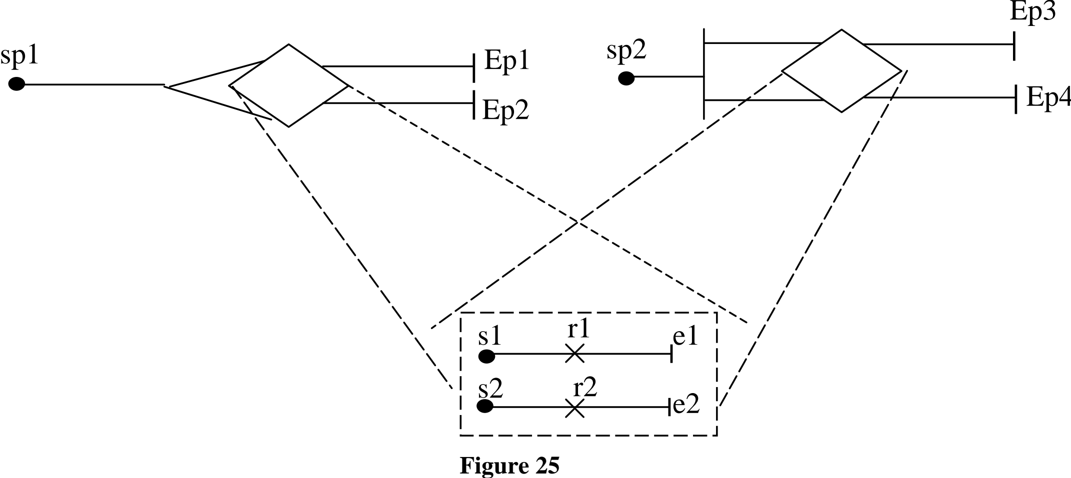


Figure 25

The general solution for these two cases is:

1. Each independent path or disconnected path segment is translated into a process with recursion or without recursion (process **stop** instead of **exit**).
2. The relationship of these paths or path segments is represented as a composition of these interleaving processes.

The example in the next section will show the translation for independent paths using the above solution.

5.5.1 Principle of Translation for UCM with Stubs and Plug-Ins

In the translation, it is assumed that

1. For each stub, only one plug-in map can be activated at one time.
2. All Plugin-Bindings are given. (see section 5.3)

In principle, a stub selectively activates one or more plug-in maps according to selection policies by sending messages. Selection policies can be reflected in LOTOS guarded behaviors.

Each plug-in map is translated as a process interleaving with the process of the root map. Each plug-in map can be available to several stubs. Plug-in maps inform their parent maps of their completion by sending messages. To provide modularity, the behavior before and after a stub is translated in two sub processes, respectively called ***PreStub*** and ***PostStub***.

A process called ***Plugin_handler*** handles messages sent from a stub to the appropriate plug-in process, as well as messages from plug-in processes to the corresponding stub, through the two gates ***to_handler*** and ***from_handler***. The process is as follows.

```
Process plugin_handler[to_handler,from_handler]:noexit:=
        to_handler ?from:datatype ?to:datatype ?para:datatype;
        (
                from_handler !from !to !para; stop
                |||
                plugin_handler[to_handler,from_handler]
        )
Endproc
```

A message is sent to the *Plugin_handler* through gate *to_handler* from the sender. This message specifies the sender (*from* in the process), the receiver (*to* in the process) and the entry/exit point (*para* in the process). Then the message is sent to the receiver through gate *from_handler* from the *Plugin_handler*. This *Plugin_handler* process is recursive in order to handle all the stubs and plug-ins.

However, some LOTOS tools (such as CAESAR [GaSi]) don't support process recursion on the left or on right hand side of |[...]|. If it is desired to obtain a LOTOS specification compatible with these tools, the Plugin_handler is:

```
Process plugin_handler[to_handler,from_handler]:noexit:=
        to_handler ?from:datatype ?to:datatype ?para:datatype;
        from_handler !from !to !para;
        plugin_handler[to_handler,from_handler]
Endproc
```

In general, a plug-in map can be represented in a LOTOS process as follows. The process is recursive in order to make it possible for this map to be activated as many times as needed simultaneously.

```
Process PlugInN[..]: noexit: =
(* Msg from stub; *)
 (
        (*Implementation of PluginN;*)
        (* Msg to stub;*)
        |||
        PlugInN[..]
         )
Endproc
```

For the same reason as mentioned above, if a LOTOS specification needs to be compatible with tools such as CAESAR, a plug-in map can be translated as follows. This plug-in map can be activated repeatedly but not simultaneously with itself:

```
Process PlugInN[..]: noexit: =
 (
        (* Msg from stub; *)
        (*Implementation of PluginN;*)
        (* Msg to stub;*)
 )
 >>
 PlugInN[..]
Endproc
```

When a dynamic stub with multiple entry points is translated, a hidden gate *selection* is used in order to ensure that the stub activates the entry points in the same plugin. Before activating the related start point of the selected plugin map, all the entry points must synchronize on the gate *selection* with the selected plugin. For example, the stub in UCM of Figure 26 has two entry points *in1* and *in2*. The ***preStub*** for the UCM in Figure 26 shows the usage of *selection*. Synchronizing on the hidden gate *selection*, either *PlugIn1* or *PlugIn2* is selected for entry *in1* and *in2*. However, the plugin cannot be processed until all the entry points arrive using this translation.

```
Process PreStub[Start,End,Resp,to_handler,from_handler]:noexit:=
        Hide selection in
        (
                resp !r1;
                (       selection !PlugIn1;
                        to_handler !P1 !PlugIn1 !entry1_plugin1;
                        stop
                        []
                        selection !PlugIn2;
                        to_handler !P1 !PlugIn2 !entry1_plugin2;
                        stop
                )
                |[selection]|
                resp !r2;
                (       selection !PlugIn1;
                        to_handler !P1 !PlugIn1 !entry2_plugin1;
                        stop
                        []
                        selection !Plugin2;
                        to_handler !P1 !Plugin2 !entry2_plugin2;
                        stop
                )
        )
Endproc
```

In our tool, the analysis of UCMs with stubs and plug-ins starts from the innermost plug-in map. The following LOTOS behaviors translate the dynamic stub with two plug-in maps shown in Figure 26.

The LOTOS behaviors are translated based on the rules we discuss above.

1. The processes for all maps including the root map and two plug-in maps are interleaved with each other, i.e. P(for root map) ||| PlugIn1 ||| PlugIn2
2. There is a process *plugin_handler* to handle the stub and the plug-ins
3. Independent paths in the plug-in map (PlugIn1) are translated into interleaving recusive processes.
4. Using gate *selection* enforces that each entry point of the stub activates the same plug-in map.

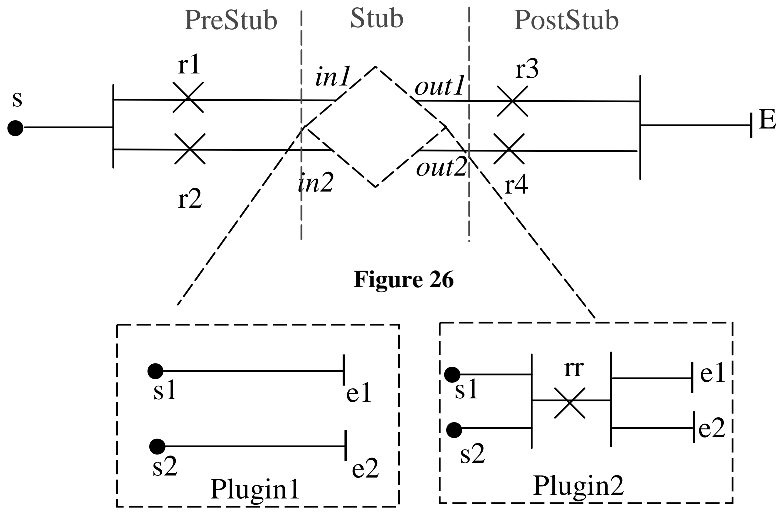


Figure 26

```
Behavior
        C1[Start, End, Resp, to_handler,from_handler]
        |[to_handler,from_handler]|
        Plugin_handler[to_handler,from_handler]
Where
```

```
Process plugin_handler[to_handler,from_handler]:noexit:=
        to_handler ?from:datatype ?to:datatype ?para:datatype;
        from_handler !from !to !para;
        plugin_handler[to_handler,from_handler]
Endproc
Process C1[Start, End, Resp, to_handler,from_handler]: noexit:=
 P1[Start,End,Resp,to_handler,from_handler] (* for root map *)
 |||
 PlugIn1[Start,End,Resp,to_handler,from_handler](*plug-in map on the left*)
 |||
 PlugIn2[Start,End,Resp,to_handler,from_handler](*plug-in map on the right*)
Where
        Process PlugIn1[Start, End, Resp, to_handler,from_handler]: noexit:=
                PlugIn1_0[Start, End, Resp, to_handler,from_handler]
                |||
                PlugIn1_1[Start, End, Resp, to_handler,from_handler]
        Where
                Process PlugIn1_0[Start, End, Resp, to_handler,from_handler]: noexit:=
        (*A message sent from a stub to the plugin to trigger the start point of this plugin*)
                        from_handler ?From:datatype !PlugIn1 !entry1_plugin1;
                        (       Start !s1;
                                End! e1;
                        (*A message sent to the stub to notify it of the end point finished*)
                                to_handler !PlugIn1 !From !exit1_plugin1; stop
                                |||
                                Plugin1_0[Start, End, Resp, to_handler, from_handler])
                Endproc
                Process PlugIn1_1[Start, End, Resp,to_handler,from_handler]: noexit:=
                        from_handler ?From: datatype !PlugIn1 !entry2_plugin1;
                        (       Start !s2;
                                End! e2;
                                to_handler !PlugIn1 !From !exit2_plugin1; stop
                                |||
                                PlugIn1_1[Start, End, Resp, to_handler, from_handler])
                Endproc
        Endproc
```

```
Process PlugIn2[Start, End, Resp, to_handler, from_handler]: noexit:=
hide sync in (
        PlugIn2_0[start,resp,end, to_handler, from_handler,sync]
        |[sync]|
        PlugIn2_1[start,resp,end, to_handler, from_handler,sync])
Where
Process PlugIn2_0[start,resp,end, to_handler, from_handler,sync]:noexit:=
        from_handler ?From:datatype !PlugIn2 !entry1_plugin2;
        (       start !s1;
                sync;
                PlugIn2_2[start,resp,end, to_handler, from_handler,sync]
                |||
                PlugIn2_0[start,resp,end, to_handler, from_handler](From)
        )
Endproc
Process PlugIn2_1[start,resp,end, to_handler, from_handler,sync]:noexit:=
        from_handler ?From:datatype !PlugIn2 !entry2_plugin2;
        (       start !s2;
                sync;
                |||
                PlugIn2_1[start,resp,end, to_handler, from_handler,sync]
        )
Endproc
Process Plugin2_2[start,resp,end, to_handler, from_handler](from:datatype):
noexit:=
        resp !rr;
        (
                end !e1;
                to_handler !PlugIn2 !From  !exit1_plugin2;
                stop
                |||
                end !e2;
                to_handler !Plugin2 !From  !exit2_plugin2;
                stop
        )
Endproc
Endproc
```

```
Process P1[Start,End,Resp,to_handler,from_handler]: noexit:=
        P1_0[Start,End,Resp,to_handler,from_handler]
Where
        Process P1_0[Start,End,Resp,to_handler,from_handler]: noexit:=
        start !s;
        (       PreStub[Start,End,Resp,to_handler,from_handler](*before stub*)
                |||
                PostStub[Start,End,Resp,to_handler,from_handler] (*after stub*)
                |||
                P1[Start,End,Resp,to_handler,from_handler]
        )
        Where
                Process PreStub[Start,End,Resp,to_handler,from_handler]:noexit:=
                        Hide selection in
                        (
                                resp !r1;
                                (       selection !PlugIn1;
                                        to_handler !P1 !PlugIn1 !entry1_plugin1;
                                        stop
                                        []
                                        selection !PlugIn2;
                                        to_handler !P1 !PlugIn2 !entry1_plugin2;
                                        stop
                                )
                                |[selection]|
                                resp !r2;
                                (       selection !PlugIn1;
                                        to_handler !P1 !PlugIn1 !entry2_plugin1;
                                        stop
                                        []
                                        selection !Plugin2;
                                        to_handler !P1 !Plugin2 !entry2_plugin2;
                                        stop
                                )
                        )
                Endproc
```

```
                    Process PostStub[Start, End, Resp, to_handler,from_handler]:
                noexit:=
                    hide sync in
                    (
                        (
                            from_handler !PlugIn1 !P1 !exit1_plugin1;
                            resp !r3;
                            sync;stop
                            []
                            from_handler !Plugin2 !P1 !exit1_plugin2;
                            resp !r3;
                            sync;stop
                        )
                        |[sync]|
                        (
                            from_handler !Plugin1 !P1 !exit2_plugin1;
                            resp !r4;
                            sync;stop
                            []
                            from_handler !Plugin2 !P1 !exit2_plugin2;
                            resp !r4;
                            sync;stop
                        )
                        |[sync]|
                        sync;
                        PostStub_1[Start, End, Resp]
                    )
                    Endproc
                    Process PostStub_1[Start, End, Resp]: exit:=
                        End! E;
                        exit
                    Endproc
                Endproc
        Endproc
Endproc
```

5.5.2 Binary Trees for LOTOS Processes

As mentioned in section 3.2, a LOTOS specification describes a system via a hierarchy of process definitions. When unbound UCMs are translated into a LOTOS specification, only one top-level process is defined, which is called Environment (*Env*). This top-level process contains one or more interleaving processes, which represent root maps and plugin maps. These processes include a new sub-process in any of following cases:

1. Each start point in a map leads to a separate sub-process.
2. The path segment after an OR-Join is translated into a sub-process (see section 5.4.4).
3. The path segment after an AND-Join is considered as a sub-process (see section 5.4.6).
4. The path segment after Stub is represented as a sub-process (see section 5.5)

Since all LOTOS operations except the *hide* are binary operations, LOTOS processes can be stored in a binary tree. Each LOTOS behavior is represented as a node (operator) with two children nodes (operands) in the binary tree. (For *hide* operantion, the left operand is empty.)

Therefore, an unbound UCM can be represented as one or more binary trees. For example, the process for the UCM in Figure 27 is stored in the tree shown in Figure 28.

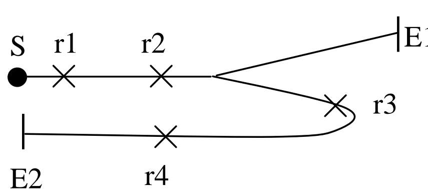


Figure 27 An Unbound UCM

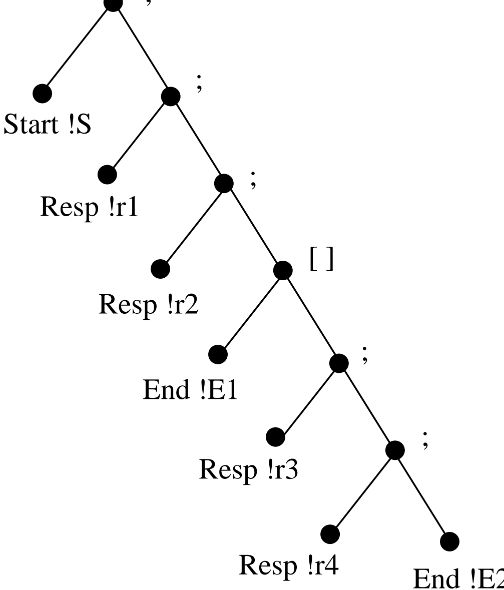


Figure 28 Tree of LOTOS Process for UCM in Figure 27

5.6 Entities, Components and Bound Use Case Map

Until now, we have focused on the interpretation of unbound UCMs, which meant that there was only one component in the whole UCM model. When the tasks are assigned to different entities, unbound Use Case Maps become bound Use Case Maps.

5.6.1 Principle of Translation for bound UCM

In our translation and conforming to the usual implicit implementation of bound UCMs, it is assumed that all the components are connected. Therefore, there are two gates (one for each direction) between any two entities besides predefined gates. And UCM components are specified as processes synchronized on their shared gates. For instance, in Figure 29, there are two gates C1_to_C2 and C2_to_C1 for UCM components C1 and C2. Processes for C1 and C2 are synchronized on gates C1_to_C2 and C2_to_C1. The behavior of Figure 29 is:

```
C1 [Start, End, Resp, C1_to_C2, C2_to_C1]
|[C1_to_C2, C2_to_C1]|
C2 [Start, End, Resp, C1_to_C2, C2_to_C1]
```

To translate from bound UCMs, not containing stubs, to LOTOS specifications, the following rules are used:

1. UCM components are represented as collections of synchronized processes, which may include sub-processes. That is, each distinct component is mapped to one different process that specifies all the responsibilities and events inside the components and their causal relationships. The main behavior in a LOTOS specification becomes a composition of component processes: one process per component.
2. When a path crosses from one UCM component to another UCM component, even if there is no explicitly specified message, it is assumed that there is a message sent from the first UCM component to the second one. For example, in Figure 29, C1 sends a message m1 to C2, and C2 receives this message from C1. In late stages of systems development, when the detailed information about

message exchanges is available; this "token" message may be replaced by the real message. The problem here is how to present a concrete message in UCM. The ideal solution is to create a new UCM element for messages between components such as shown in Figure 30. In the current situation where UCM lacks of elements representing message, a messages can be described within the responsibility. For instance, in Figure 30, responsibility *dial* is defined as *dial !number*, *chk* as *chk ?number*. Here, "*!*" may be interpreted as sending a message and "*?*" as receiving a message.

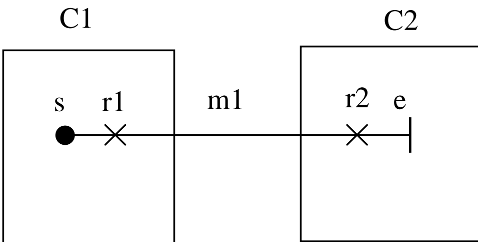


Figure 29

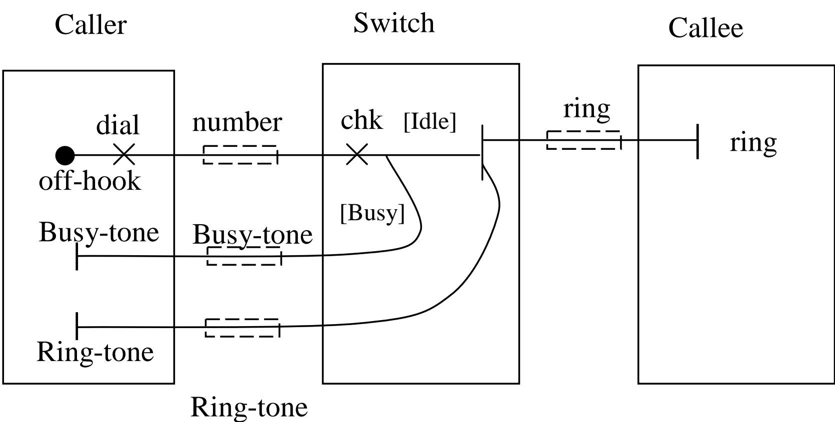


Figure 30 An Bound UCM with New Element for Messages

3. The paths of a UCM are broken down into path segments based on the boundaries of UCM components. The segments within the components are translated into LOTOS behaviors in the corresponding processes. When a bound UCM is translated, the paths are traversed, the UCM notation is translated and the structure is stored in the same way as in an unbound UCM. As mentioned in

section 5.5.2, a LOTOS process for one component is stored in a binary tree. Therefore, there are one or more binary trees for each component. Each tree represents a LOTOS process for one component. The LOTOS process includes the LOTOS behavior in this component and the messages exchanged with the component. For example, the UCM in Figure 31 is one possible bound UCM for the unbound UCM shown in Figure 27. The trees of the LOTOS processes for the components are shown in Figure 32.

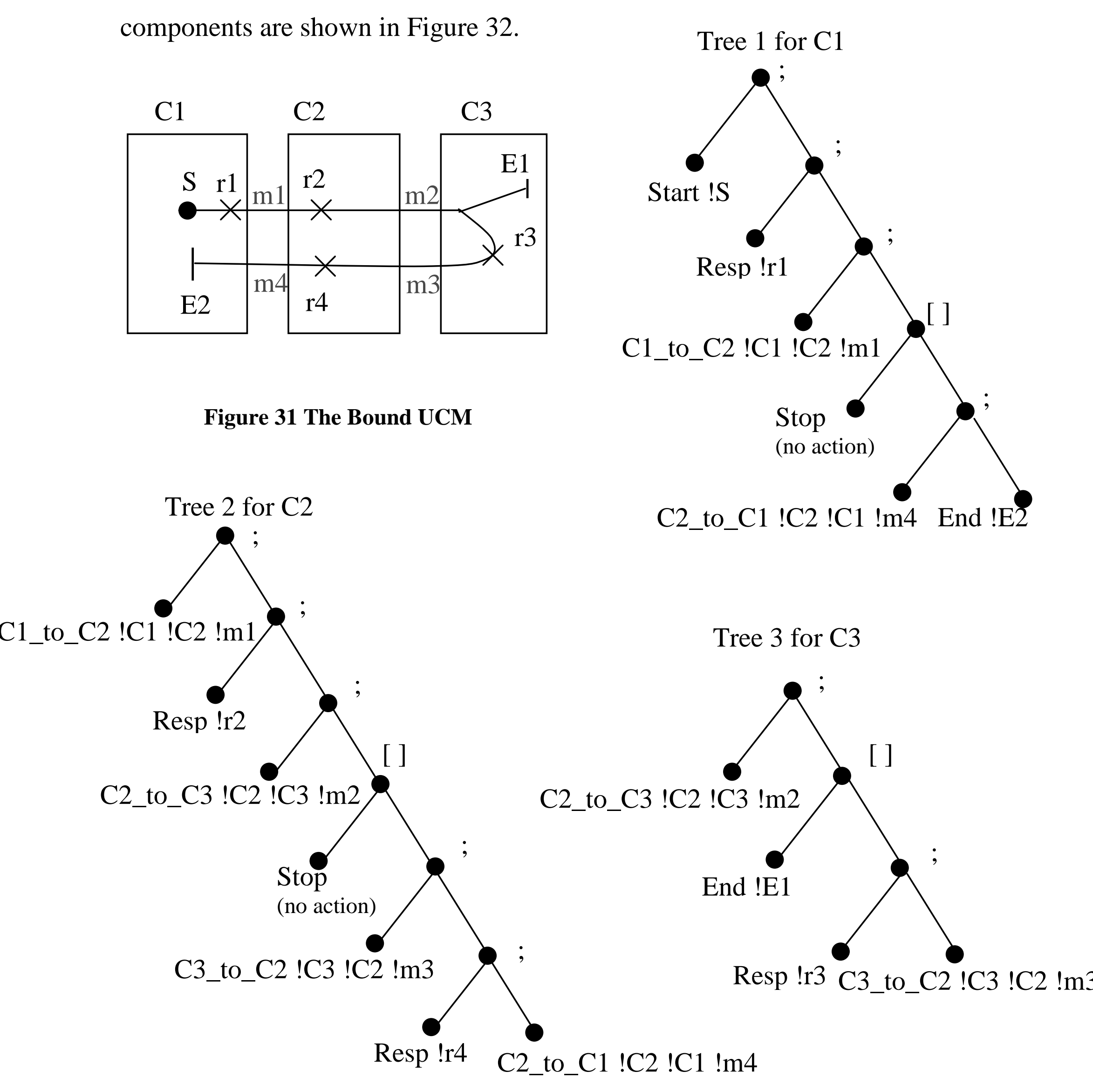


Figure 31 The Bound UCM

Figure 32 Trees of LOTOS processes for the Bound UCM

On the basis of these translation rules, the bound UCM of Figure 31 is represented in LOTOS as follows:

```
Behavior
 (C1 [Start, End, Resp, C1_to_C2, C2_to_C1, C1_to_C3, C3_to_C1]
 | [C1_to_C2, C2_to_C1]|
 C2 [Start, End, Resp, C1_to_C2, C2_to_C1, C2_to_C3, C3_to_C2])
 |[C1_to_C3, C3_to_C1, C2_to_C3, C3_to_C2]|
 C3 [Start, End, Resp, C1_to_C3, C3_to_C1, C2_to_C3, C3_to_C2]
Where
        Process C1[Start, End, Resp, C1_to_C2, C2_to_C1]: noexit: =
                Start !C1 !s;
                (       Resp !C1 !r1;
                        C1_to_C2 !C1 !C2 !m1;
                        (       stop
                                []
                                C2_to_C1 !C2 !C1 !m4;
                                End !C1 !E2;stop
                        )
                        |||
                        C1[Start, End, Resp, C1_to_C2, C2_to_C1]
                )
        Endproc
        Process C2[Start, End, Resp, C1_to_C2, C2_to_C1]: noexit: =
                C1_to_C2 !C1 !C2 !m1;
                (       Resp !C2 !r2;
                        C2_to_C3 !C2 !C3 !m2;
                        (       stop
                                []
                                C3_to_C2 !C3 !C2 !m3;
                                Resp !C2 !r4;
                                C2_to_C1 !C2 !C1 !m4; stop
                        )
                        |||
                        C2[Start, End, Resp, C1_to_C2, C2_to_C1]
                )
        Endproc
```

```
Process C3[Start, End, Resp, C1_to_C2, C2_to_C1]: noexit: =
        C2_to_C3 !C2 !C3 !m2;
        (       (       End !C3 !E1; stop
                        []
                        Resp !C3 !r3;
                        C3_to_C2 !C3 !C2 !m3; stop
                )
                |||
                C3[Start, End, Resp, C1_to_C2, C2_to_C1]
        )
Endproc
```

In our translation, a message is composed as follows:

sender_to_receiver !sender !receiver !message

In above example, gate C1_to_C2 is used to represent a channel from component C1 to C2. Experiments C1 and C2 represent the sender and receiver. This message format is chosen because it is the one that is required by the tool Lotos2MSC[SteLo]. In particular, this tool requires offering separately gates with the names of the sender and receiver of each message.

5.6.2 Bound UCM with Stub and Plug-ins

A UCM model may include some root maps and several plug-in maps. One UCM component may exist in different maps (either root maps or plug-in maps). In our translation, a component process includes component sub-processes, each of which is for one map and all of which are parallel in the same component process. For example, in the UCMs of Figure 33, there is a process C1 for component C1. Because component C1 lies on the root map and the plug-in map, there are two sub-processes interleaving in process C1: sub-process C1_m0 for component C1 in the root map and sub-process C1_m1 for component C1 in the plug-in map.

Although a stub belongs unambiguously to a given component, the plug-in maps it contains may or may not be related to the same component the stub is in. In addition, the

plug-in map may be referred to by multiple stubs. To enable feasibility and reusability of plug-in maps in the LOTOS translation, a stub activates a plug-in map by sending a message through the ***Plugin_handler*** (see section 5.5). For example in Figure 33, the start point of the plug-in is in the same component (C2) as the stub is. The stub sends a message from component C2 in root map (C2_m0) to the component C2 in the plug-in map (C2_m1), i.e, *to_handler !C2_m0 !C2_m1 !entry1_plugin;* the plug-in receives the message, i.e. *from_handler ?from:datatype !C2_m1 !entry1_plugin*. The end point of the plug-in is in the different component (C3) as the stub is (C2). The plug-in map sends a message from component C3 in plug-in map (C3_m1) to component C2 in root map (C2_m0), i.e. *to_handler !C3_m1 !C2_m0 !exit1_plugin.*

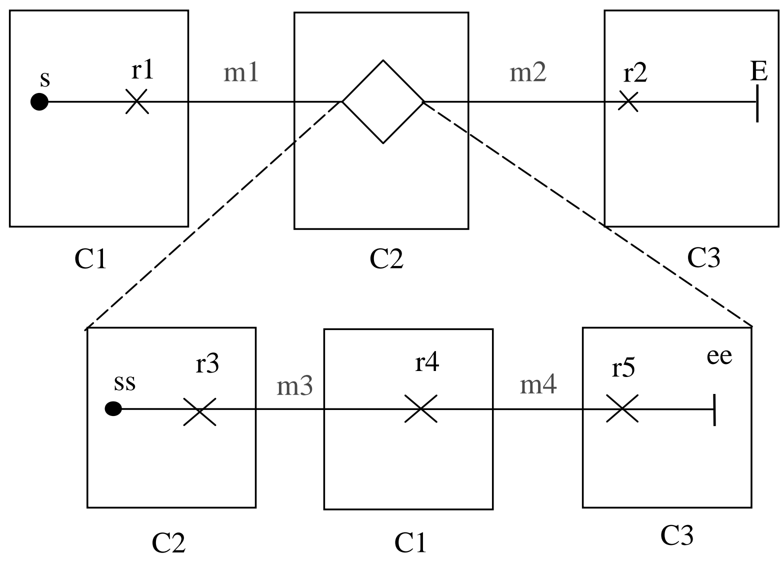
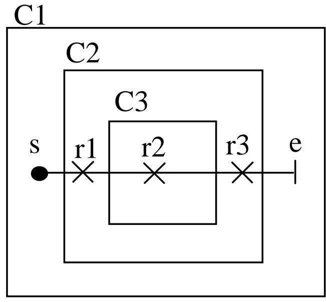


Figure 33 A Bound UCM with Stub

5.6.3 Nested Components

In UCMs, components can be nested inside other components such as the UCM in Figure 34. From the designers' points of view, the UCM in Figure 34 is not equivalent to the UCM in Figure 35 since C3 is part of C2 and C2 is part of C1 in the UCM of Figure 34 while the UCM in Figure 35 does not present this relationship.



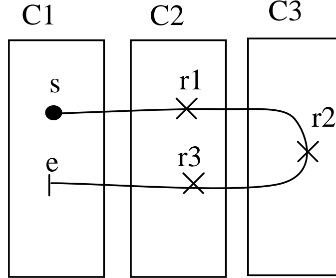


Figure 34 UCM with nested components **Figure 35 UCM without Nested Components**

But in terms of LOTOS, the two UCMs can be specified as the same LOTOS behaviors. In our tools, the maximum nesting level is one. The relationship of any two components is if there is channel between them. Therefore, the LOTOS specification generated for the UCM in Figure 34 is same as the one for the UCM in Figure 35.

5.6.4 Use of Empty Points

Determining whether a path goes across one component or not generally involves complex geometry analysis. To alleviate this difficulty, Empty Points are used instead. In UCMNav, Empty Points are used to shape UCM paths. They don't have semantic meaning. However, in our tools, an empty path segment with one Empty Point inside a component means that the component relays the message from the previous component to the next one along the path. That is, the Empty Point is used to anchor an empty path to a component.

5.7 Design of Ucm2LotosSpec

The principle of translation from UCMs to LOTOS has been discussed in above sections. This section introduces the data structures and the basic algorithms for the tools *Ucm2LotosSpec*.

5.7.1 XML Representation for UCMs

UCMs are represented in XML by the tool *UCMNav*. According to the XML Data Type Definition for UCMs, Figure 36 and Figure 37 show the data structure for UCM in XML (only the parts in which we are interested are shown). The elements in XML are shown in boxes and their attributes are listed below them.

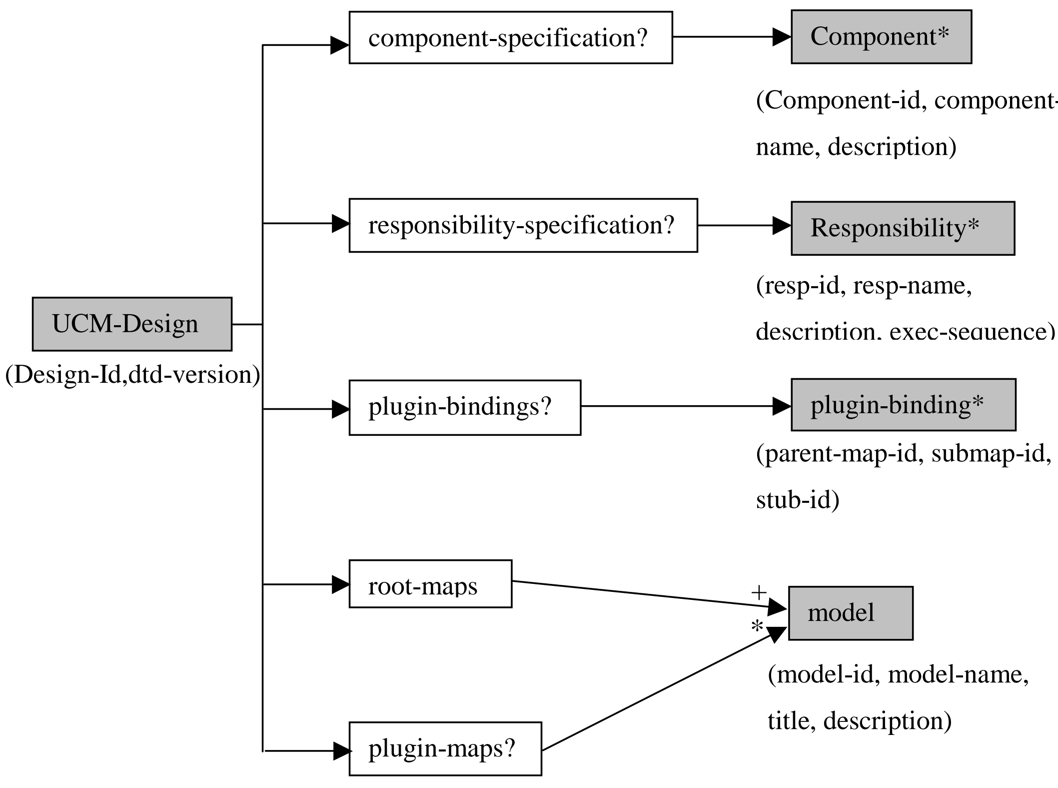
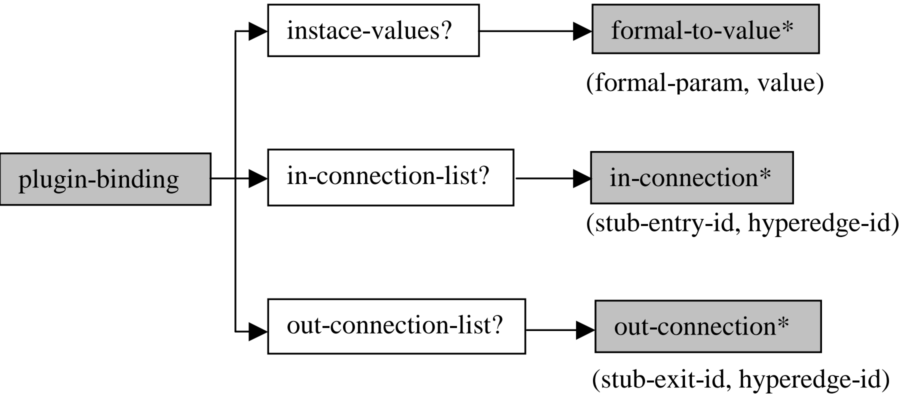


Figure 36 XML representation for UCM(1)



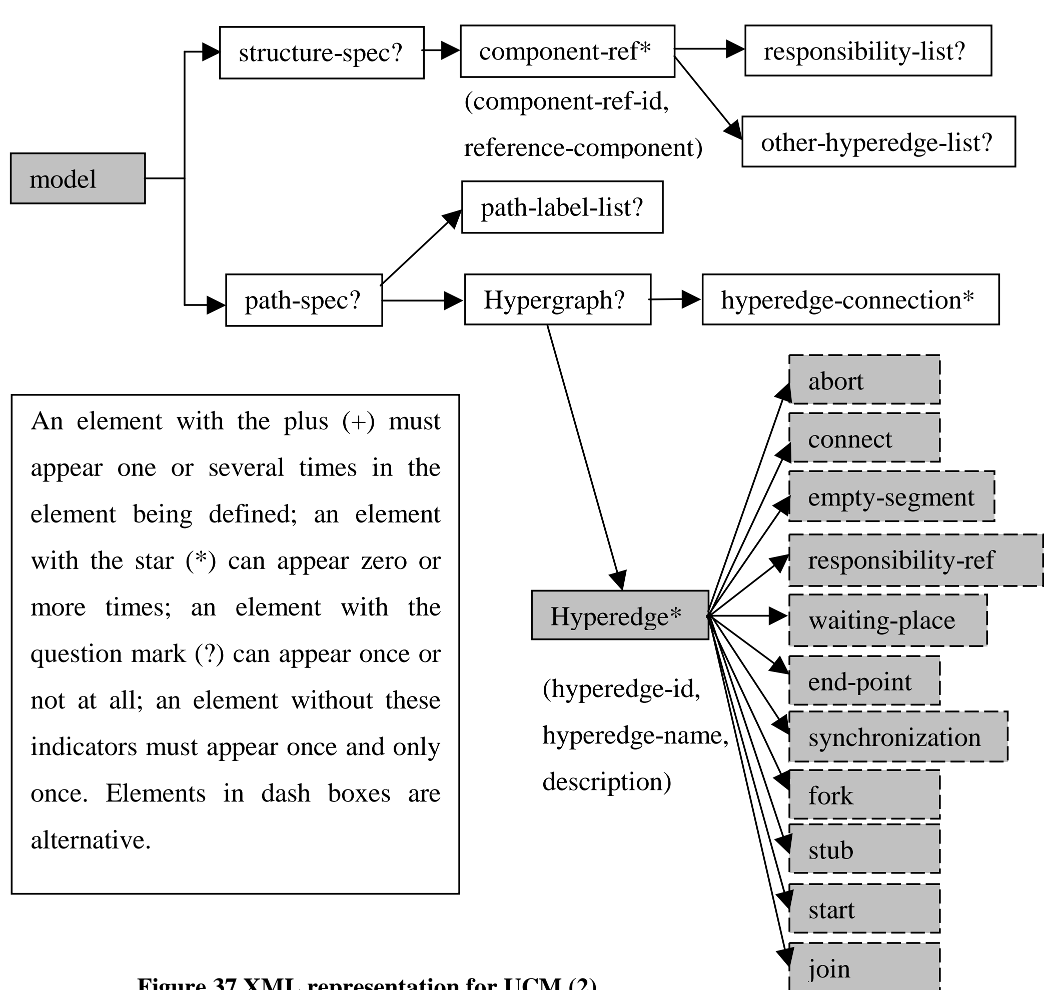


Figure 37 XML representation for UCM (2)

A UCM model is called an *UCM-Design* in XML. It contains root-maps and plugin-maps, which are represented as *Models* in XML. In the *UCM-Design*, *Components*, *Responsibilities* and *Plugin-bindings* are also specified. Each *Model* is made up of a collection of nodes (*hyperedges*) connected together (*hyperedge-connection*).

5.7.2 Internal Representation of the UCMs in *Ucm2LotosSpec*

5.7.2.1 Classes Diagram for *Ucm2LotosSpec*

Based on the XML data type definition for UCM, one class is defined for each element in the grey box of Figure 36 and Figure 37. Class Ucm2lotos is for the XML element *UCM-Design*; Class Map is for the XML element *Model*; Class Components is for the XML element *Component*; Class Resp is for the XML element *Responsibility*; Class PluginBinding is for the XML element *Plugin-binding*; Class Nodes is for element *Hyperedge*. Class AbortNode, ConnectNode, EmptyNode, RespNode, WaitNode, EndNode, SyncNode, ForkNode, StubNode, StartNode, JoinNode for *abort*, *connect*, *empty-segment*, *responsibility-ref*, *waiting-place*, *end-point*, *synchonization*, *fork*, *stub*, *start*, *join* respectively are the sub-classes of Class Nodes. Class Tree and TNode are used for storing information during the translation. The Class Diagram for *Ucm2LotosSpec* is shown in Figure 38.

5.7.2.2 Internal Representation for UCM paths

In the XML representation for UCM, UCM paths are represented by *hypergraphs* which contains sets of *hyperedges* and *hyperedge-connections*. Following the *hyperedge-connections*, all paths in a UCM can be traversed. In our tools, the *hyperedge-connection* is represented as a bi-directional linked list of Nodes in a map. Therefore, for each node, there is one array for its previous node(s) and one array for its next node(s). All paths can be traversed from start points following the *next* link. For example, a UCM in Figure 39 is represented in our tools as in Figure 40.

Starting from the StartNode in Figure 40, three paths can be extracted: StartNode:offHook -> RespNode:dial -> RespNode:chk -> ForkNode -> EndNode:Busy-tone;

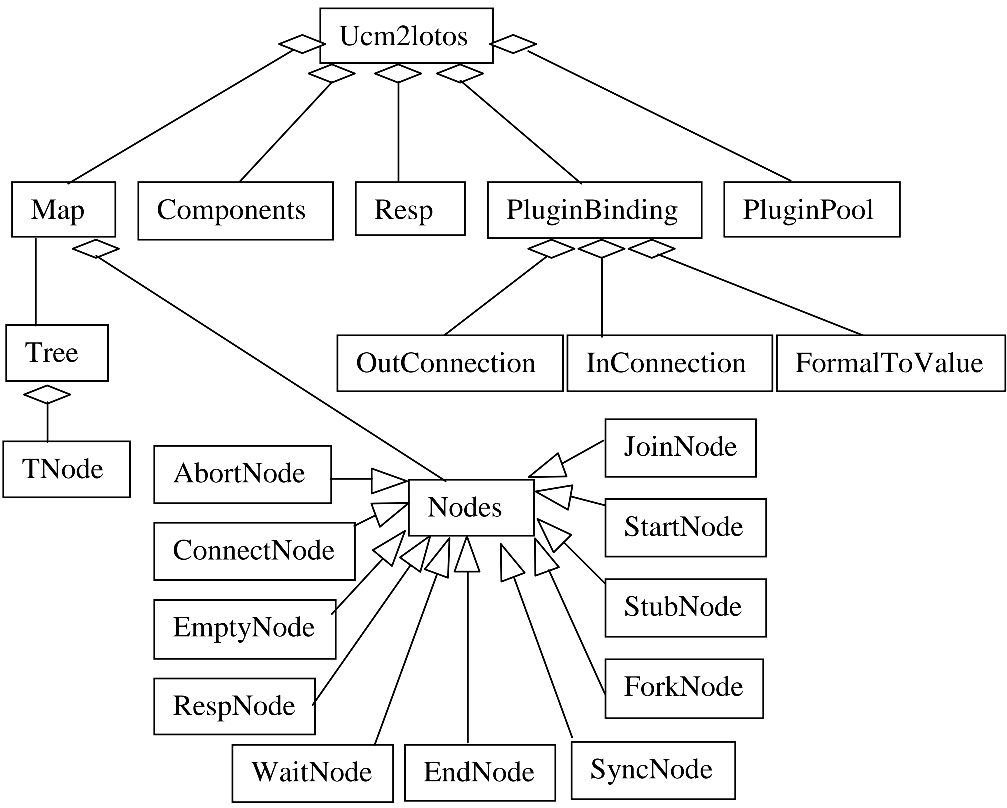


Figure 38 Class Diagram for Ucm2LotosSpec and Ucm2LotosScenarios

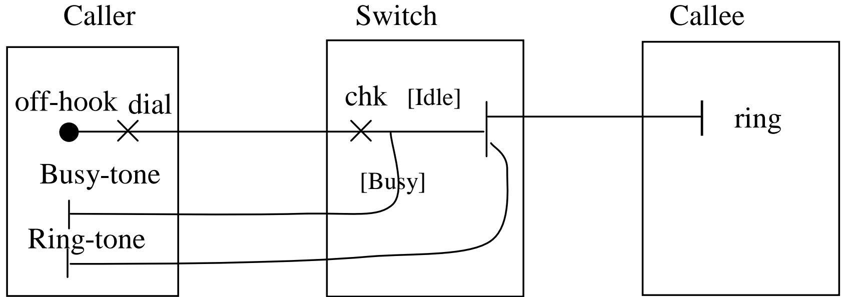


Figure 39 A UCM

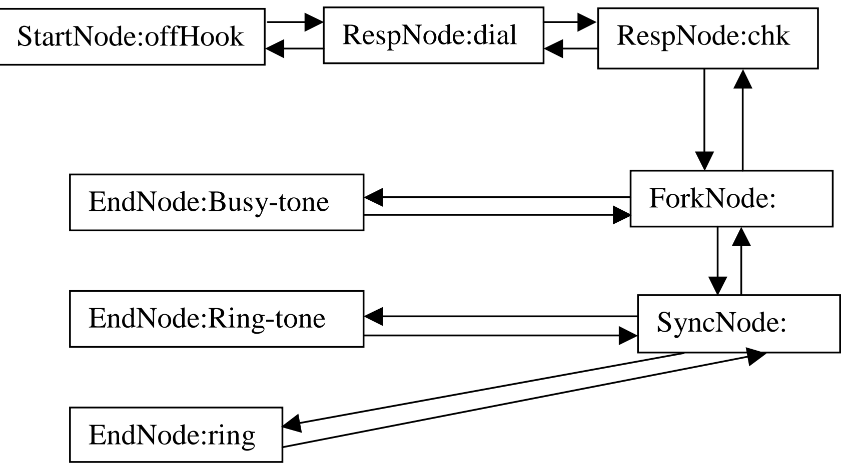


Figure 40 internal representation of the UCM

StartNode:offHook -> RespNode:dial -> RespNode:chk -> ForkNode -> SyncNode -> EndNode:Ring-tone;

StartNode:offHook -> RespNode:dial -> RespNode:chk -> ForkNode ->SyncNode -> EndNode:ring.

The *previous* link is used when paths are visited in the reversed direction.

5.7.3 Outline of the Algorithm

For each connected UCM,

1. The UCM is traversed and each *path segment* is numbered according to its start point. Figure 41 shows a UCM where the *path segments* are numbered in the curly brackets.
2. The UCM is traversed from an end point to the start point(s). Starting from the end point with the greatest number of paths, a root node of a tree is created as the current node. Visiting the path in the reverse direction,
    1) When reaching OR-Join, AND-Join or Waiting-Place, a child node, with the OR-Join/AND-Join/Waiting-Place as its attribute, is created for the current node and it becomes the current node. Visiting each of its reverse branches, repeat 1) or 2).

2) When reaching Start Point, a child node, with ID of the Start Point as its attribute, is created for the current node. This path has been visited.

After visiting the whole UCM, a tree called MTree will have been produced. In this tree, the leaves are start points and the internal nodes are OR-Join/AND-Join/Waiting-Place. For example, Figure 42 shows the MTree for the UCM of Figure 41.

If UCM has disconnected paths (UCM is not connected), one MTree is produced for each independent path in the UCM. The MTree is used to assemble the individual behavior for each start point in order to construct LOTOS behaviors for the given UCM. For example, let the LOTOS process led by the start point S_i be PS_i. From the MTree in Figure 42, we can construct a LOTOS behavior for this tree: PS_1 *OR-Join* ((PS_2 *OR-Join* PS_3) *AND-Join* PS_4).

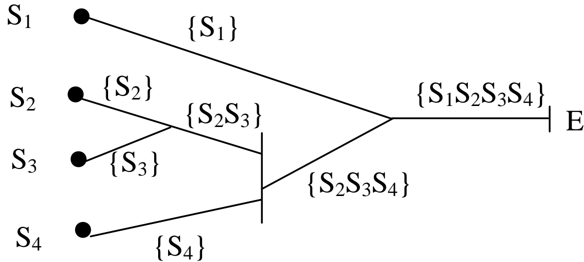


Figure 41 A UCM Example

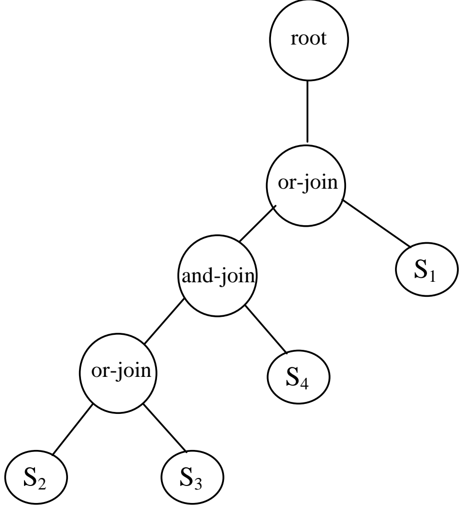


Figure 42 A MTree for the UCM of Figure 41

3. For each start point of each MTree, create one binary tree for each component, trace the UCM from the start point in the MTree,
    1) Any UCM notation element is translated into LOTOS behaviors based on the mapping rules introduced in the above sections. The LOTOS behaviors are stored in the binary tree corresponding to the component.
    2) When reaching an OR-Join, AND-Join or Stub, a sub-process is defined and a new binary tree for the sub-process is created to store the behaviors after these operators.
4. For each MTree, LOTOS processes are constructed from binary trees generated for each start point (step 3). The LOTOS processes for the same component are assembled in an upper-level LOTOS process based on the MTree (step 2).
5. For the UCM, one LOTOS process for each component is defined as a collection of interleaving LOTOS processes for the same component generated in step 4. For example, there are three components (C1, C2, C3) in the root map (m0) of Figure 33, after step 1-5, there will be three LOTOS processes for each component: C1_m0, C2_m0 and C3_m0.

For a UCM model,

1. After constructing LOTOS processes for each UCM, the control part of the LOTOS specification for the UCM model is made up of a set of LOTOS processes, each of which consists of the interleaving LOTOS processes for one component in each UCM and all of which are synchronized on their shared gates. For example in Figure 33, the LOTOS behavior is

```
((C1 [Start,End,Resp,C1_to_C2,C2_to_C1,C1_to_C3, C3_to_C1, to_handler, from_handler]
|[C1_to_C2, C2_to_C1]|
C2 [Start,End,Resp,C1_to_C2,C2_to_C1,C2_to_C3, C3_to_C2, to_handler, from_handler])
|[C1_to_C3, C3_to_C1, C2_to_C3, C3_to_C2]|
C3 [Start,End,Resp,C1_to_C3,C3_to_C1,C2_to_C3,C3_to_C2, to_handler, from_handler])
|[to_handler, from_handler]|
plugin_handler[to_handler, from_handler]
(*Where C1 is C1_m0 ||| C1_m1, C2 is C2_m0 ||| C2_m1 and C3 is C3_m0 ||| C3_m1.*)
```

2. The data part of the LOTOS specification is constructed simply.

Four data types are defined in the LOTOS specification Data Part for responsibilities, start points, end points and components.

Each new defined data type consists of defining names of data carriers (called *sorts*) and operations. All the operations in these four data types are nullary operations, which are called *constants* and represent each start point, end point, responsibility and component in the UCM model. Therefore, the specification of the data type for start points, end points, responsibilities and components are:

Type startType **is**
Sorts start
opns
    S1, S2, …, Sn:->start
Endtype
(*Where S1, S2, … and Sn are the name of the start points in the UCMs representing the triggering events. *)

Type endType **is**
Sorts epoint
opns
    E1, E2, …, En:->epoint
Endtype
(*Where E1, E2, … and En are the name of the end points in the UCMs representing the resulting events. *)

Type actionType **is**
Sorts responsibility
opns
    R1, R2, …, Rn:->responsibility
Endtype
(* Where R1, R2, … and Rn are the name of the responsibility in the UCMs representing actions, tasks.*)

```
Type compType is
Sorts comp
opns
        C1, C2, …, Cn:->comp
Endtype
(*Where C1, C2, … and Cn are the name of the Components in the UCMs.*)
```

Also we extend the Boolean type specification for preconditions.

```
Type preConditions is Boolean
opns
        Con1, Con2, …,Conn:->comp
eqns
ofsort Bool
        Con1 = true ;
        Con2 = true ;
        …
        Conn = true ;
Endtype
(* Where Con1, Con2, … , Conn are the pre-conditions for selections predicates.*)
```

In the automatic generation, all the selection predicates are considered to be true in our specification.

For the UCMs presented in Figure 33, following the algorithm described above, the control part of the LOTOS specification can be generated as follows:

```
Behavior
((C1 [Start,End,Resp,C1_to_C2,C2_to_C1,C1_to_C3, C3_to_C1, to_handler, from_handler]
|[C1_to_C2, C2_to_C1]|
C2 [Start,End,Resp,C1_to_C2,C2_to_C1,C2_to_C3, C3_to_C2, to_handler, from_handler])
|[C1_to_C3, C3_to_C1, C2_to_C3, C3_to_C2]|
C3 [Start,End,Resp,C1_to_C3,C3_to_C1,C2_to_C3,C3_to_C2, to_handler, from_handler])
|[to_handler, from_handler]|
plugin_handler[to_handler, from_handler]
Where
```

```
Process Plugin_handler[to_handler, from_handler]: noexit:=
        to_handler ?From: datatype ?To:datatype ?para:datatype1;
        from_handler !From !To !para;
        plugin_handler[to_handler,from_handler]
Endproc

Process C1 [Start, End, Resp, C1_to_C2, C2_to_C1, C1_to_C3, C3_to_C1,
to_handler, from_handler]: noexit:=  (*for Component C1*)
        C1_m0 [Start,End,Resp, C1_to_C2, C2_to_C1, C1_to_C3,C3_to_C1,
    to_handler, from_handler]
        |||
        C1_m1 [Start,End,Resp, C1_to_C2, C2_to_C1, C1_to_C3, C3_to_C1,
    to_handler, from_handler]
Where
        Process C1_m0 [Start, End, Resp, C1_to_C2, C2_to_C1, C1_to_C3,
    C3_to_C1, to_handler, from_handler]: exit:= (*for root map in C1*)
                Start !C1 !s;
                (       Resp !C1 !r1;
                        C1_to_C2 !C1 !C2 !m1;stop
                        |||
                        C1_m0 [Start, End, Resp, C1_to_C2, C2_to_C1, C1_to_C3,
                C3_to_C1, to_handler, from_handler])
        Endproc
        Process C1_m1[Start, End, Resp, C1_to_C2, C2_to_C1, C1_to_C3,
    C3_to_C1, to_handler, from_handler]: noexit:= (*for plug-in map in C1*)
                C2_to_C1 !C2 !C1 !m3 ?from:datatype;
                (       Resp !C1 !r4;
                        C1_to_C3 !C1 !C3 !m4 !from;stop
                        |||
                        C1_m1 [Start, End, Resp, C1_to_C2, C2_to_C1, C1_to_C3,
        C3_to_C1, to_handler, from_handler])
        Endproc
Endproc
```

```
Process C2 [Start, End, Resp, C1_to_C2, C2_to_C1, C2_to_C3, C3_to_C2,
to_handler, from_handler]: noexit:= (*for Component C2*)
        C2_m0 [Start, End, Resp, C1_to_C2, C2_to_C1, C2_to_C3, C3_to_C2,
    to_handler, from_handler]
        |||
        C2_m1 [Start, End, Resp, C1_to_C2, C2_to_C1, C2_to_C3, C3_to_C2,
    to_handler, from_handler]
    Where
        Process C2_m0 [Start, End, Resp, C1_to_C2, C2_to_C1, C2_to_C3,
    C3_to_C2, to_handler, from_handler]: exit:=  (*for root map in C2*)
            C1_to_C2 !C1 !C2 !m1;
            (       Sub_m0_0_C2 [Start, End, Resp, C1_to_C2, C2_to_C1,
            C2_to_C3, C3_to_C2, to_handler, from_handler]
                    |||
                    Sub_m0_1_C2 [Start, End, Resp, C1_to_C2, C2_to_C1,
            C2_to_C3, C3_to_C2, to_handler, from_handler]
                    |||
                    C2_m0 [Start, End, Resp, C1_to_C2, C2_to_C1, C2_to_C3,
            C3_to_C2, to_handler, from_handler]
            )
        Where
            Process Sub_m0_0_C2 [Start, End, Resp, C1_to_C2, C2_to_C1,
    C2_to_C3, C3_to_C2, to_handler, from_handler]: noexit:=
                (*preStub in C2 in root map*)
                to_handler !C2_m0 !C2_m1 !entry1_plugin;
                stop
            Endproc
            Process Sub_m0_1_C2 [Start, End, Resp, C1_to_C2, C2_to_C1,
    C2_to_C3, C3_to_C2, to_handler, from_handler]: noexit:=
                (*postStub in C2 in root map*)
                from_handler !C3_m1 !C2_m0 !exit1_plugin;
                C2_to_C3 !C2 !C3 !m2;
                stop
            Endproc
        Endproc
```

```
		Process C2_m1[Start, End, Resp, C1_to_C2, C2_to_C1, C2_to_C3,
	C3_to_C2, to_handler, from_handler]: noexit:= (*for plug-in map in C2*)
			from_handler ?from:datatype !C2_m1 !entry1_plugin;
			(	Start !C2 !ss;
				Resp !C2 !r3;
				C2_to_C1 !C2 !C1 !m3 !from;stop
				|||
				C2_m1[Start, End, Resp, C1_to_C2, C2_to_C1, C2_to_C3,
			C3_to_C2, to_handler, from_handler]
			)
		Endproc
	Endproc
	Process C3 [Start, End, Resp, C1_to_C3, C3_to_C1, C2_to_C3, C3_to_C2,
to_handler, from_handler]: noexit:= (*for Component C3*)
		C3_m0 [Start, End, Resp, C1_to_C3, C3_to_C1, C2_to_C3, C3_to_C2,
	to_handler, from_handler]
		|||
		C3_m1 [Start, End, Resp, C1_to_C3, C3_to_C1, C2_to_C3, C3_to_C2,
	to_handler, from_handler]
	Where
		Process C3_m0 [Start, End, Resp, C1_to_C3, C3_to_C1, C2_to_C3,
	C3_to_C2, to_handler, from_handler]: noexit:=  (*for root map in C3*)
			C2_to_C3 !C2 !C3 !m2;
			(	Resp !C3 !r2;
				End !C3 !E;stop
				|||
				C3_m0 [Start, End, Resp, C1_to_C3, C3_to_C1, C2_to_C3,
			C3_to_C2, to_handler, from_handler]
			)
		Endproc
		Process C3_m1[Start, End, Resp, C1_to_C3, C3_to_C1, C2_to_C3,
	C3_to_C2, to_handler, from_handler]: noexit:= (*for plug-in map in C3*)
			C1_to_C3 !C1 !C3 !m4 ?to:datatype;
```

```
                    (       Resp !C3 !r5;
                            End !C3 !ee;
                            to_handler !C3_m1 !to !exit1_plugin;stop
                            |||
                            C3_m1[Start, End, Resp, C1_to_C3, C3_to_C1, C2_to_C3,
                C3_to_C2, to_handler, from_handler]
                    )
                Endproc
            Endproc
```

5.8 Degree of Automation

In Section 5.2 of [AM01], Amyot presented 8 general construction guidelines for the generation of LOTOS prototypes from UCMs. Table 5 provides a degree of automation for each guideline, from complete automatic (3) to manual (0).

5.9 Conclusion

In this chapter, we have presented the design of the tool *Ucm2LotosSpec* that is used to automatically generate LOTOS specifications from UCMs. First, the problems to be overcome for the translation were discussed and the subset of UCM notation that our tool supports was defined. Then the translation of the basic UCM notation was introduced. Next, the translation from unbound and bound UCMs to LOTOS specifications was presented. Finally, the data structure and outline of the algorithm are discussed. An example of application is given in Chapter 7.

Construction Guideline	Automation	Degree
1. Interpreting Interaction Points and Responisbilities	Start Points, End Points, Waiting Places and responsibilities can be translated.	3
2. Causal Paths	Linear causal paths, OR-Fork, AND-Fork, OR-Join, AND-Join can be translated.	3
3. Stubs and Plug-ins	Skeletons can be generated for stubs and plug-ins. Parameter passing has not been implemented.	2
4. Other Path elements	Timers, Aborts, Failures Points, Dynamic responsibilities are not translated.	0
5. Structures	Components can be translated into different processes. Components inside components also can be solved.	3
6. Unrelated path segments	Unrelated path segments have been handled	3
7. Inter-component causality	Communication channels can be generated. Use of empty point relays messages.	3
8. Data	Simple data types are defined for start points, end points and responsibilities, components, preconditions.	1

Table 5 The Degree of Automation from UCM to LOTOS

Chapter 6 Ucm2LotosScenario: Automatic LOTOS Scenario Generation from Use Case Maps

6.1 Overview

Scenarios are the basis for the design of tests, which are used to validate a system against its requirements. Also, the illustration of scenarios is the purpose of Message Sequence Charts, which are widely used in the telecommunication area.

UCMs capture the functional requirements of a system by describing a set of scenarios of the system, including scenario interactions. In UCMs, one scenario is a route traced from start point(s) to end point(s). It illustrates a functionality of the system it describes. A LOTOS scenario is a representation in LOTOS of one scenario in the UCM. Our method (see section 4.2 and Figure 7) consists in constructing LOTOS scenarios and executing them with the LOTOS specification using the tool LOLA. To make it possible, LOTOS scenarios must satisfy certain constraints:

1. They should not contain processes.
2. They should contain a termination event that indicates the end of the scenario (these will be called *scenario*, *scen1*, *scen2*, …). This appears just before a stop.

In our method, a set of LOTOS scenarios is run against the LOTOS specification to validate it, hence indirectly to validate the UCM design. The valid LOTOS specification with scenarios produces LOTOS traces that are used to generate Message Sequence Charts. In order to provide complete and valid LOTOS scenarios, it is necessary to have a tool to generate them from UCMs automatically.

In this chapter, the design of a tool called *Ucm2LotosScenario*, which automatically generates LOTOS scenarios from UCMs, is presented. The basic algorithm for *Ucm2LotosScenario* is also given.

6.2 Design of Ucm2LotosScenario

In this section, the translation of basic path elements will be introduced first. Then the translation of unbound UCM and bound UCM will be presented. The translation tool that does these tasks is called *Ucm2LotosScenario*.

The same predefined gates are used: Start, End and Resp to represent start point, end point and responsibility at the very early stages of system design. When the design of a system evolves and responsibilities are assigned to components, specific gates between components are added. At this time, unbound UCMs become bound UCMs.

The difference between the translation from UCMs to LOTOS specification and the translation from UCM scenarios to LOTOS scenarios is that LOTOS scenarios do not contain sub-processes, nor the choice operator. When UCMs are translated to LOTOS scenarios, UCMs stubs have to be flattened and OR-Fork and OR-Join have to be decomposed. Also, concurrency is flattened.

6.2.1 Translation of Basic Path Element

For the reasons presented in section 5.3, *Ucm2LotosScenario* doesn't support all the UCM features. It supports the same subset of UCM Notation supported by *Ucm2LotosSpec*.

6.2.1.1 Start Point, End Point and Responsibility

UCM start point, end point and responsibilities are translated by using the same mapping as the one used to translate UCMs into LOTOS specifications (see section 5.4.2).

A Use Case Map that only contains start point, responsibility and end point is translated to LOTOS scenario as follows:

Start! (Triggering Event| name of start point);
Resp! the label of the responsibility;
End! (Resulting Event| name of end point);
scenario; stop

6.2.1.2 OR-Fork

A Use Case Map including OR-Forks describes alternative scenarios and it can be translated into several LOTOS scenarios. Figure 43 will be described in two LOTOS scenarios as follows:

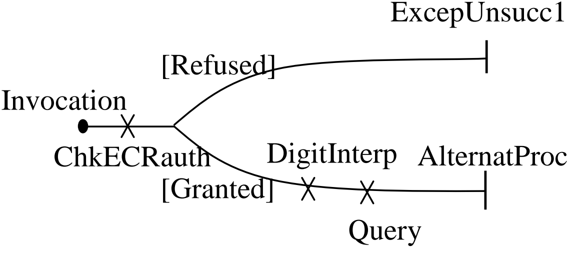


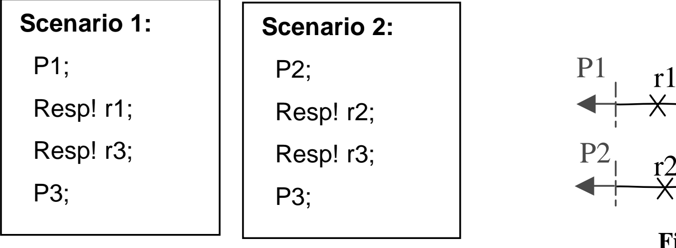
Figure 43

```
Process   scenario1[Start,End,Resp,scen1]:
noexit: =
  Start! Invocation;
  Resp! ChkECRauth;
  (* If Refused*)
  End! ExcepUnsucc1;
  scen1;
  stop
Endproc
```

```
Process  scenario2  [Start,End,Resp,scen2]:
noexit: =
  Start! Invocation;
  Resp! ChkECRauth;
  (* If Granted*)
  Resp! DigitInterp;
  Resp! QueryLoc;
  End! AlternatProc;
  scen2; stop
Endproc
```

6.2.1.3 OR-Join

An OR-Join in a Use Case Map contains a sequence of responsibilities that are after the OR-Join and are shared by two or more path segments before the OR-Join. There are several LOTOS scenarios for the OR-Join. The LOTOS scenarios for Figure 44 are (P1, P2, .., Pn in this section represents a set of LOTOS behaviors) :



```
Scenario 1:
  P1;
  Resp! r1;
  Resp! r3;
  P3;
```

```
Scenario 2:
  P2;
  Resp! r2;
  Resp! r3;
  P3;
```

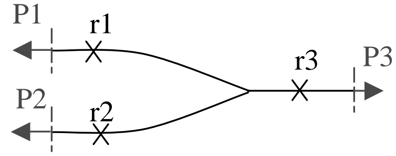


Figure 44

6.2.1.4 AND-Fork and AND-Join (Synchronization)

If there is an AND-Join in a UCM, all path segments before the AND-Join must be executed before carrying on the rest of path after the AND-Join in one LOTOS scenario. The concurrency is flattened. In Figure 45, the number of LOTOS scenarios is the permutation of r1, r2, r3, r4. Two of the LOTOS scenarios for Figure 45 are as follows:

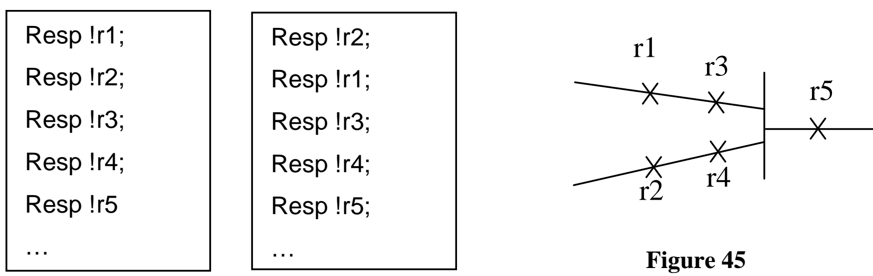


Figure 45

If there is AND-Fork in a UCM, all path segments after the AND-Fork are executed in parallel in one LOTOS scenario. The LOTOS scenario of Figure 46 is:

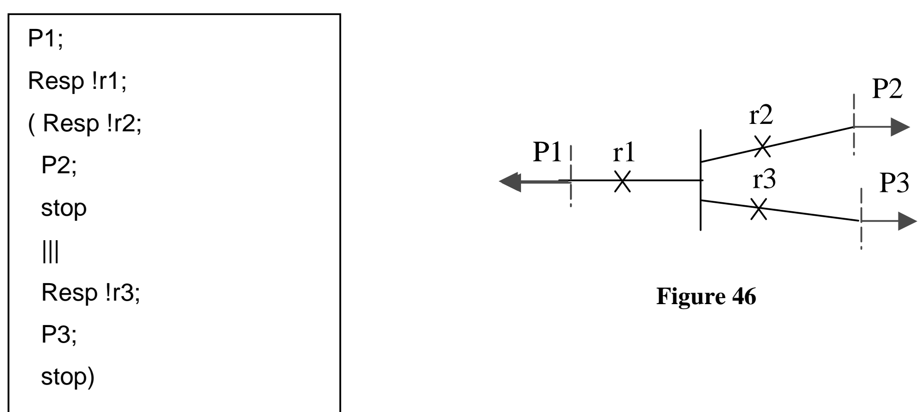
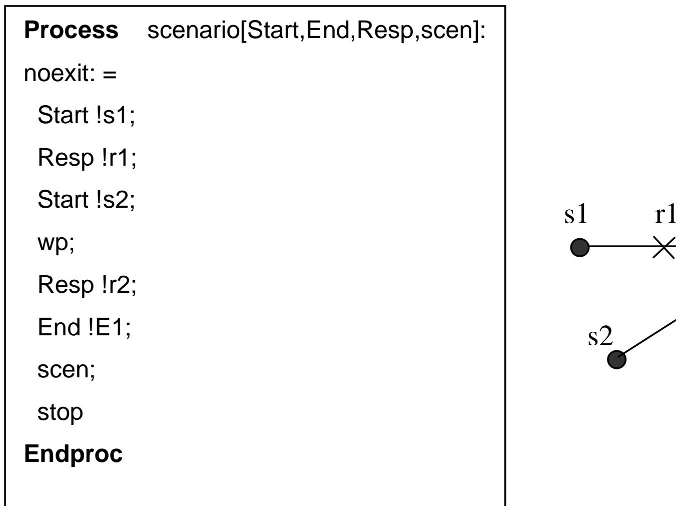


Figure 46

6.2.1.5 Waiting Place

A waiting place triggers a suspending path. The LOTOS scenario for the UCM in Figure 47 is as follows:



```
Process    scenario[Start,End,Resp,scen]:
noexit: =
  Start !s1;
  Resp !r1;
  Start !s2;
  wp;
  Resp !r2;
  End !E1;
  scen;
  stop
Endproc
```

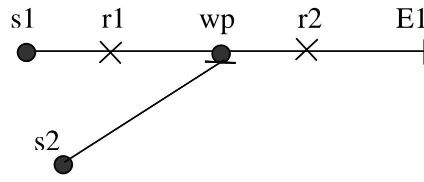


Figure 47

6.2.1.6 Stub

As explained in section 6.1, LOTOS scenarios do not contain any sub-process. Therefore, when UCMs are translated into LOTOS scenarios, all stubs inside UCMs must be flattened. A UCM with static stubs can be flattened into a UCM directly. For example, the UCM on the left in Figure 48 is flattened to the one on the right before translating it into a LOTOS scenario.

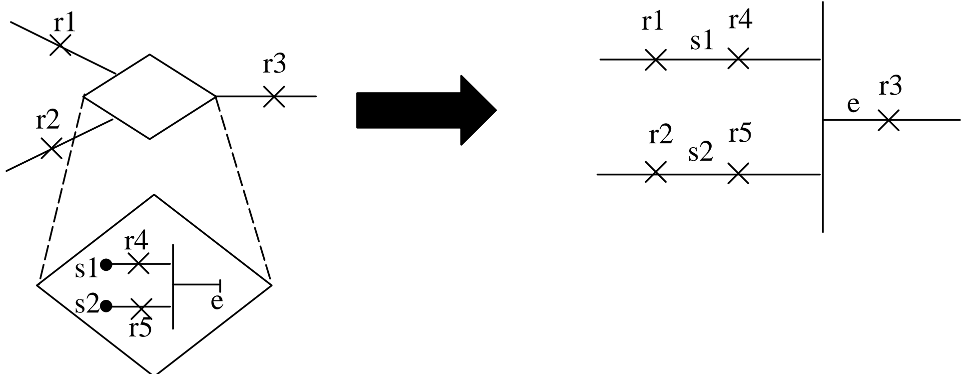


Figure 48

If a UCM contains dynamic stubs, one plug-in map is needed for each LOTOS scenario according to the plug-in selection policy. For example in Figure 49, translating the UCM on the left into LOTOS scenarios is equivalent to translating the two UCMs on the right.

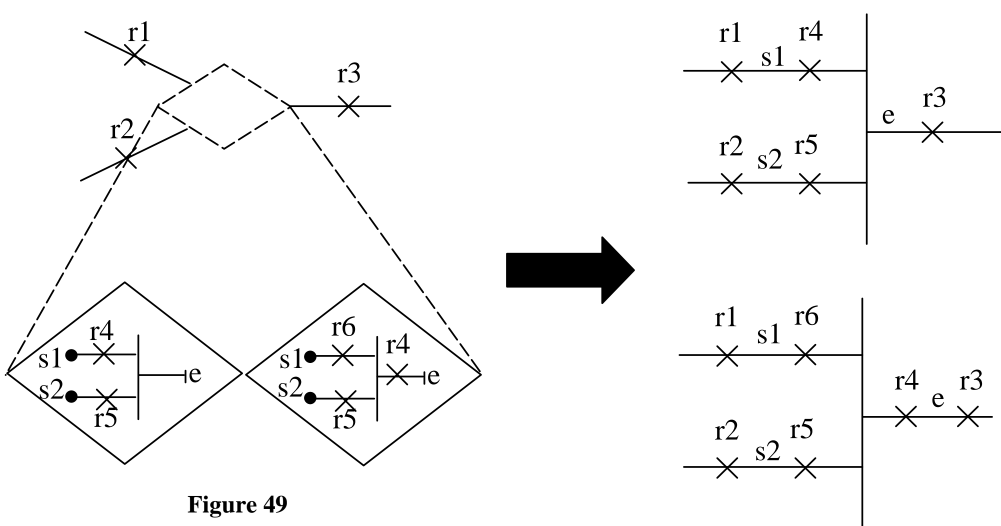


Figure 49

6.2.1.7 Loop

Although this tool does not support the UCM loop notation, a loop can be represented as an OR-Join preceding an OR-Fork, where at least one of the paths from the OR-Fork goes back to the OR-Join. The tool can translate a loop described in this way into LOTOS scenarios.

A construct, which consists of one or more mutually ***embedded loops***, is called a ***loop construct***. For example in Figure 50, there is one *loop construct* with two *embedded loops*. The path segment including responsibility r3 is one *embedded loop*; another path segment including responsibility r2 is another *embedded loop*. In Figure 51, there are two *loop constructs*, one with one *embedded loop* and the other with *two embedded loops*.

In principle, there is an infinite number of scenarios for a UCM that has a loop (loops). The *Ucm2LotosScenario* tool generates enough LOTOS scenarios from a UCM with *a loop construct* to include each *embedded loop* by itself and all permutations of that *embedded loop* with all combinations of the other *embedded loops*. In our tool, each *embedded loop* in a *loop construct* is only traversed once for each scenario. If a UCM has more than one *loop construct*, the generated LOTOS scenarios are the multiplication of the number of LOTOS scenarios for each *loop-construct*. Therefore, there are five possible scenarios as shown for Figure 50. There are ten possible scenarios for the UCM in Figure 51.

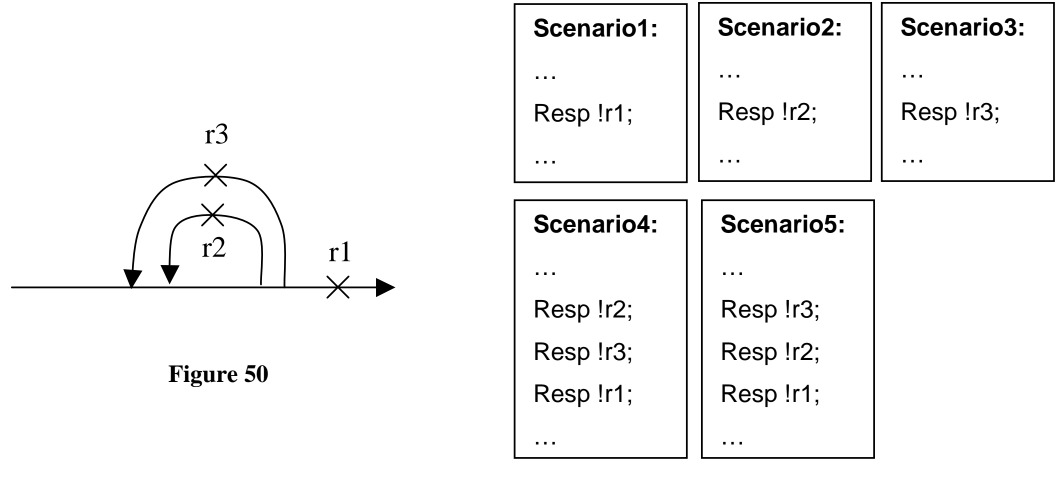


Figure 50

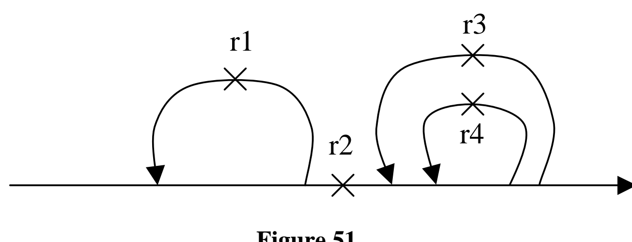


Figure 51

In general, let L be the number of *loop constructs* and B_n be the number of *embedded-loops* in the n_{th} *loop construct.* We can conclude that in a UCM that contains L loop constructs with B_L embedded-loops, there are $\prod_{n=1}^{L}$ (B_n* B_n! + 1) scenarios.

Two issues must be noted here:

1. According to different testing methods, there will be different ways to produce scenarios from UCMs with loops. Since this thesis is not focused on testing, only our method is discussed. Other methods could be devised, of course. Also, we do not discuss the advantages and disadvantages of our method compared to others. For a more complete discussion of these issues, please refer to the thesis of Charfi [Ch01].
2. In a loop consisting of an OR-Join and OR-Fork, before the OR-Fork, usually there is a checking point to check which *embedded-loop* should be traversed. However, our tool generates all possible scenarios without considering the selection condition. This may lead to producing some unfeasible scenarios. This problem and its possible solutions will be discussed in section 6.5.

6.2.2 Scenarios for Unbound UCMs

It has been already shown that at the early stages of system design, the system is described by unbound UCM. For translating unbound UCMs to LOTOS scenarios, three rules have to be followed:

1. The three predefined gates: Start, End and Resp are used to represent start point, end point and responsibility.
2. When translating unbound UCMs without loops, all possible scenarios are produced using the rules of translation of basic path elements.
3. When translating unbound UCMs with loops, enough scenarios are generated according to the principle discussed in section 6.2.1.7.

In Chapter 7, the tool was applied in the WIN project to obtain LOTOS scenarios for unbound UCMs for Fleet and Asset Management and Enhanced Call Routing.

6.2.3 Scenarios for Bound UCMs

As the description of the system evolves, responsibilities are assigned to different entities and unbound Use Case Maps become bound Use Case Maps. To translate bound UCMs

to LOTOS scenarios, the following rules are applied in addition to all the rules for translation of unbound UCMs:

1. Besides the three predefined gates, there are two gates between any two components. Detailed information can be found in section 5.6.
2. When a path crosses from one UCM component to another UCM component, it is assumed that a message is sent from the first UCM component to the second one (see section 5.6). This same assumption is made when bound UCMs are translated into LOTOS specifications. The same message names can be generated for the LOTOS specification and LOTOS scenarios because the same hyperedge Id is used as the name of the message.

All LOTOS scenarios of unbound UCMs for ECR service are shown in Appendix A. Some LOTOS scenarios of bound UCMs for FAM service are shown as examples in Appendix B.

6.3 Basic Algorithm for *Ucm2LotosScenario*

Ucm2LotosScenario uses the same Internal Representation of UCMs and data structures as *Ucm2LotosSpec* (see section 5.7.2.2). Given a UCM, the tool *Ucm2LotosScenario* traces all UCM scenarios while translating these scenarios into LOTOS format.

The basic algorithm is as follows:

1. Trace the innermost plugin-map first, then its parent map. In other words, the maps are translated from inside to outside.
2. Trace all start points in order to obtain all UCM scenarios. If the start point has been traced before, return. Otherwise, generate a new scenario, translate to LOTOS operation, store the necessary information and trace its next node with this incomplete scenario. If the next node is
3. A) Responsibility Node: translate to a LOTOS operation and trace its next node with this incomplete scenario (back to step 3).

   B) OR-Join Node: trace its next node with this incomplete scenario (back to step 3).

C) OR-Fork Node: For each branch, generate one scenario that contains all the actions before the OR-Fork, then trace the next node in this branch with this incomplete scenario (back to step 3).

D) Synchronization: If all the incoming paths have not been traced once, return. Otherwise, for each outgoing branch, trace its next node (back to step 3). After receiving return scenarios from its next nodes, flatten concurrency of the incoming paths, interleaving the outgoing paths and construct all scenarios.

E) Stub: for each plugin map, choose the appropriate scenario(s) and trace the next node with the incomplete scenario (back to step 3)

F) Waiting Place: visit the path that triggers this waiting place and then trace the next node with the incomplete scenario (back to step 3).

G) End: trace the next node with the incomplete scenario if it has one (back to step 3). Otherwise, translate to LOTOS action and complete the scenario and return the scenario.

H) Other Node: trace the next node directly (back to step 3).

6.4 Comparison with *UCMNav*

One of the purposes for generating LOTOS Scenarios from UCMs is to produce the Message Sequence Charts from these scenarios. *UCM Navigator* [UCM], a prototype tool for UCM, has the functionality to derive Message Sequence Charts from UCM scenario specification [URN-Z152]. When comparing the *UCM Navigator* with our tools including *Ucm2LotosSpec* and *Ucm2LotosScenario*, it should be taken into consideration that the main purpose of our tools is to produce a formal specification of the behaviors specified in the Use Case Maps. This specification can be used in a variety of ways, for example, formal verification, model checking, and automatic extraction of test cases. If we limit consideration to the generation of scenarios, then the following differences can be noted:

1. The Path Traversal Mechanism for abstracting scenarios in *UCMNav* covers all UCM path elements [URN-Z152], while *Ucm2LotosScenario* only supports a subset of UCM path elements (see section 5.3).
2. The scenarios generated by *Ucm2LotosScenario* are compatible with the LOTOS specification constructed by *Ucm2LotosSpec*. Any changes in any of UCMs, LOTOS specification, LOTOS scenarios or the MSCs produced from these scenarios are easily reflected on the others. This is useful to allow designers to trace changes in design.
3. *UCMNav* can generate basic MSCs and HMSCs while MSCs generated via *Ucm2LotosScenario* and *Lotos2Msc* are basic MSCs without MSC actions.

In terms of completeness of generated MSCs, *UCMNav* is more powerful than *Ucm2LotosScenario*. However, *Ucm2LotosScenario* is more suitable in a methodology that uses the formal specification as well as the scenarios.

6.5 Eliminating Unfeasible Scenarios

When generating UCMs with OR-Fork(s) or dynamic stub(s), the tool generates all possible scenarios without considering selection conditions (all the selection predicts are considered to be true in our specification). This may lead to producing unfeasible scenarios. For example in Figure 52, there are two possible scenarios. But using this tool, two unfeasible scenarios are also generated besides the two valid scenarios.

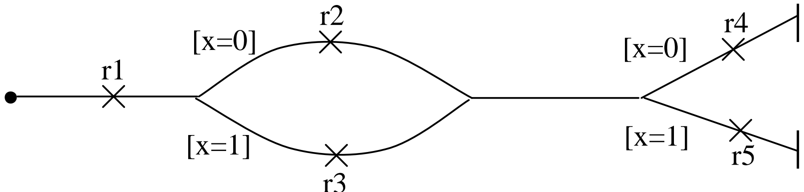


Figure 52

Valid Scenarios:	
Scenario 1:	**Scenario 2:**
Start !s;	Start !s;
Resp !r1;	Resp !r1;
Resp !r2;	Resp !r3;
Resp !r4;	Resp !r5;
End !e1;	End !e2;
scen1;	scen2;
stop	stop

Unfeasible scenarios generated by the tool:	
Scenario3:	**Scenario4:**
Start !s;	Start !s;
Resp !r1;	Resp !r1;
Resp !r2;	Resp !r3;
Resp !r5;	Resp !r4;
End !e2;	End !e1;
scen3;	scen4;
stop	stop

There are the following possible solutions for this problem:

1. For the case of UCM [Mi01], the new concepts of *Scenario Variables* and *Scenario Definition* were introduced. *Scenario Variables*, which are global boolean variables, must be defined for selection conditions at choice points. One *scenario definition* is given for each scenario. The *scenario definition* ensures that only the proper path is taken.
   This method could be automated. Instead of defining a *scenario definition* for each scenario, valid scenarios can be generated by using all combinations of true/false for all *scenario variables*.
2. In the case of LOTOS, the problem can be solved by including in the specification additional data types to be used in tests. This could be obtained either by editing manually the specification obtained by our tool, or by augmenting the tool to add the capability to generate automatically tests and the necessary data types. In order to obtain this second goal, however, the UCM notation would have to be augmented in order to contain precise semantics for data and related boolean operations.

6.6 Conclusion

In this chapter, we have presented the design of the tool *Ucm2LotosScenario* to automatically generate LOTOS scenarios. Also, the solutions for the problem of eliminating unfeasible scenarios were discussed. This tool can be used in practice to

generate Message Sequence Charts, thus enabling some automation in the design process. An example of application is given in Chapter 7.

Chapter 7 Case Study: Wireless Intelligent Network Location Based Service System

In this chapter, our approach is applied to produce correct scenarios, in the form of Message Sequence Charts, for the new Wireless Intelligent Network services – Location Based Services (LBS) in Stage 1 and the Stage 2.

7.1 Initial Requirements of WIN LBSS and UCM Presentations

Standards Requirements Document for WIN Phase III [WINTIA99] and Review 1 [WINTIA00] describe user requirements for the Wireless Intelligent Network enhancements to support Location Based Services. Based on these two documents [WINTIA99] and [WINTIA00], we give the informal requirements of Location Based Services and describe these services by means of UCM.

7.1.1 Location Based Services: General Description

There are four services in Location Based Services System which will be developed by network operators and which will be supported by WIN. They are Location-Based Charging (LBC), Fleet and Asset Management (FAM), Location Based Information Service (LBIS), Enhanced Call Routing (ECR). These four illustrative services have a common characteristic: they all require that the service providers have access to the location information of the subscriber and this information is fundamental to the delivery of the service. This requires the existence in the wireless network of a location determining capability. PDE (Position Determining Entity) and MPC (Mobile Positioning Center) in the WIN Reference Model provide this capability. In this section, FAM and ECR are chosen as our examples and discuss them in details.

7.1.2 Fleet and Asset Management (FAM)

The English description of the requirements of FAM service is as follows [WINTIA00]:

Fleet and Asset Management (FAM) service allows for Fleet and Asset Tracking and Management using call associated and non-call associated Location Based Services. Examples may include a supervisor of a delivery service who needs to know the location and status of his/her employees, parents who wish to know where their children are, or even inanimate objects such as vending machines which a company may wish to track. FAM service incorporates a supervisor function and a subordinate function. The FAM supervisor is typically leader of the fleet or owner of the asset or parent of the child. The FAM subordinate is the member of the fleet or the asset or the child. There are typically multiple subordinates for every supervisor.

The FAM supervisor shall be capable of tracking (e.g., knowing the location of) all active subordinates at all times. All FAM subordinates are equipped with appropriate mobile stations.

The FAM supervisor should have access to real-time location information of the FAM subordinates. If this information is not available, the "best available" location information may be provided and a "cause code" [specifying the reason why real-time location information is not available] may be provided to the supervisor. If the location information is not current, a "timestamp" [specifying the time at which the location data was procured] may be provided to the supervisor.

Optionally, the FAM subordinate may have access to his or her own location information upon request.

Normal Procedures with Successful Outcome

FAM is invoked when a FAM "event"" occurs. Examples of FAM events are:

- *Supervisor Request (supervisor member requests location status of a subordinate member), or*
- *Location Change (when changes in location status are detected, e.g., when the subordinate crosses a zone boundary; the smallest location change that can cause a Location Change event to occur is 10 meters, or the minimum change that can be detected by the location determining equipment, which ever is greater), or*
- *Mobile Status Change (when change in mobile status is detected, e.g., when the subordinate mobile station powers on, or when it powers off, or when the mobile becomes*

- *Scheduled (periodicity of scheduled FAM events is generally determined by FAM service logic; the minimum period is 1 minute; the maximum period is 24 hours), or.*
- *Call Associated (when a subordinate originates or terminates a call), or*
- *Subordinate Request (when a subordinate manually initiates sending of location status information to the supervisor).*

…

Exception Procedures with Successful Outcome

If a FAM event occurs, and the network determines that the location information for the specified Subordinate is not available, the network may send a Cause Code to the supervisor indicating the reason that the information is not available (e.g., mobile station not found, mobile station out of service area, resource shortage, etc.).

If a FAM event occurs, and the network determines that the location information for the specified Subordinate is not current, the network may send a Time Stamp to the supervisor indicating the time at which the location information was procured.

…

Location Information Flows

The WIN location architecture shall support multiple modes of operation in terms of the "flow" of Location Information and commercial services provided based on location.

- *Asynchronous Event Location Information Push*

The WIN location service architecture shall support the capability to have the Mobile Station (MS) initiate the flow of location information to the location-based service logic program based on the occurrence of an asynchronous event detected by the mobile entity.

- *Asynchronous Event Location Information Pull*

The WIN location service architecture shall support the capability to have the Service Control Point (SCP) initiate the flow of location information to the location-based service logic program based on some SCP based asynchronous event.

- *Synchronous Event Location Information Poll*

The WIN location service architecture shall support the capability to have the SCP poll multiple mobile entities to initiate the flow of location information to the location-based service logic program based on some SCP based synchronous clock or via a periodic, cyclic request for those mobile entities.

- *Synchronous Event Location Information Pull*

The WIN location service architecture should support the capability to have the location-based service logic program synchronously pull location information from a subscribed mobile entity.

According to the documents, the requirements are described in unbound UCMs since the network entities are unclear at this time. It should be noted that our UCMs include considerable interpretation of the English text of the standard. These were the result of discussion with experts and they are necessary in order to obtain meaningful UCMs.

Three unbound UCMs in Figure 53, Figure 54 and Figure 55 show how FAM handles the following FAM events:

1. Supervisor Request: as shown in Figure 53, a supervisor member requests location status of a subordinate member. If the supervisor doesn't have the authorization or it hasn't been activated, the network sends *Error Code* to the supervisor. If the network determines that the location information for the specified Subordinate is not available, the network may send a *Cause Code* to the supervisor indicating the reason why the information is not available. If the network determines that the location information for the specified subordinate is not current, the network may send a Time Stamp to the supervisor indicating the time where the location information was procured. If the current location of the specified subordinate is obtained, the network delivers the location information to the supervisor. In last three cases, the system should record call detail information (*RecInvocDet*).
2. Scheduled: as shown in Figure 54, the periodicity of scheduled FAM events is generally determined by FAM service logic. When the scheduled FAM event occurs, the new location may be obtained, updated and sent to the supervisor.
3. Update Location: as shown in Figure 54, when a subordinate manually initiates sending of location status information to the supervisor, the current location will be obtained, updated and sent to the supervisor.

4. Location Change: as shown in Figure 54, when changes in location status of an active FAM subordinate are detected, the new location may be obtained and updated. The new location may be sent to the supervisor.
5. Call Associated: as shown in Figure 54, when a subordinate originates or terminates a call, its location may be obtained, updated and sent to the supervisor.
6. Mobile Status Change: as shown in Figure 55, when change in subordinate mobile status is detected, the status is stored. If the WIN location architecture is in push mode (see requirement text), the location will be monitored and stored. The status information will also be sent to the supervisor. If the WIN location architecture is in pull mode, the status information will be sent to the supervisor.

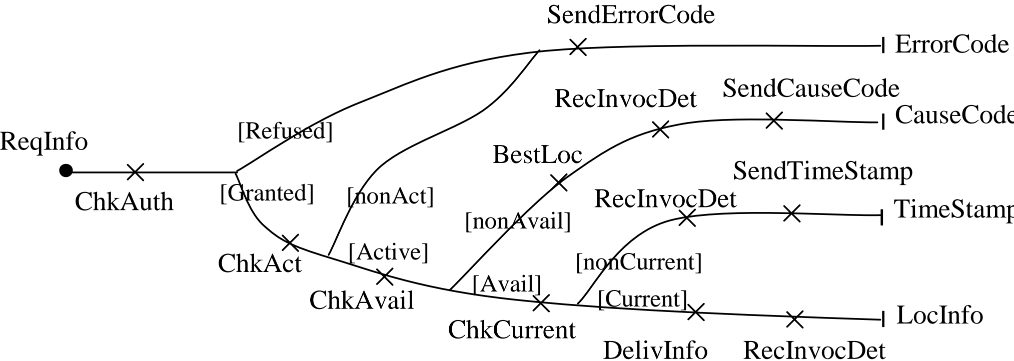


Figure 53 FAM Request of Location/Status

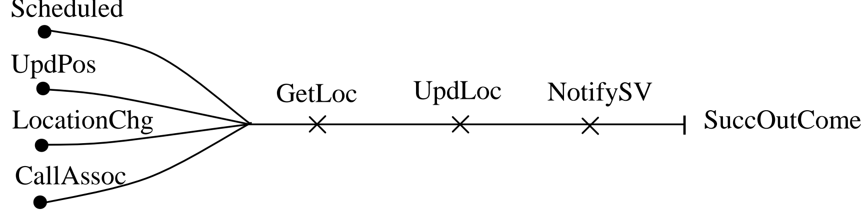


Figure 54 FAM Location Update

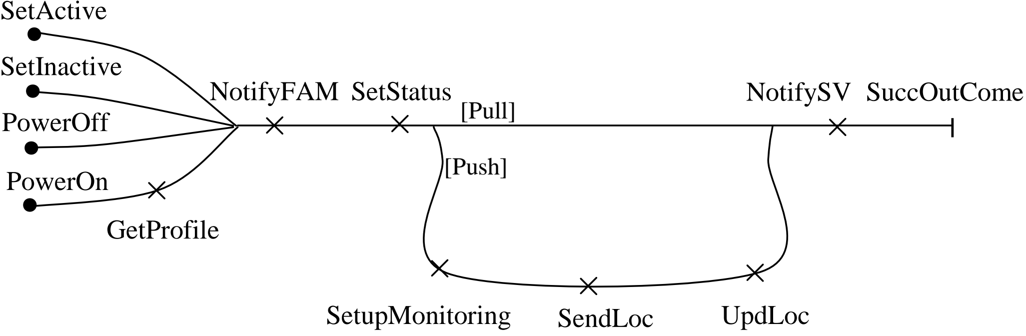


Figure 55 FAM Status Change

7.1.3 Enhanced Call Routing (ECR)

The requirements of FAM service is as follows [WINTIA00]:

Enhanced Call Routing (ECR) allows calls to be routed to the appropriate geographic destination based on the location/position of the mobile. For example, if the user dials # 427 (i.e., # GAS) the call will be routed to the nearest gas station. Another form of ECR causes the call to complete via the least expensive route, sends the call to a toll free 800 number or routes the call to the nearest private network node for a large customer, all based on customer location.

As an option, the call may be air time free to the caller and the number code holder may be charged for the air time.

Normal Procedures with Successful Outcome

This section describes a normal sequence of procedures for ECR operation.

1. Customer dials the abbreviated dialing sequence such as # + XXX or # + DN.

2. The network routes the call to the appropriate geographic destination based on the location or position of the mobile and the dialed digits. For example, if the customer dials # 427 (i.e., # GAS), the call will be routed to the nearest gas station. Or if the customer dials # + DN, the call will be routed to the nearest network node for the customer.

Exception Procedures with Successful Outcome

A typical problem during ECR invocation is that the network may not have the location or position of the mobile at the time of call setup. In this case, exception procedures shall be defined locally.

According to the original requirement, an unbound UCM in Figure 56 represent ECR service. A subscriber dials the abbreviated dialing sequence such as # + XXX or # + DN. If the customer is not registered for the service, an exception procedure is invoked.

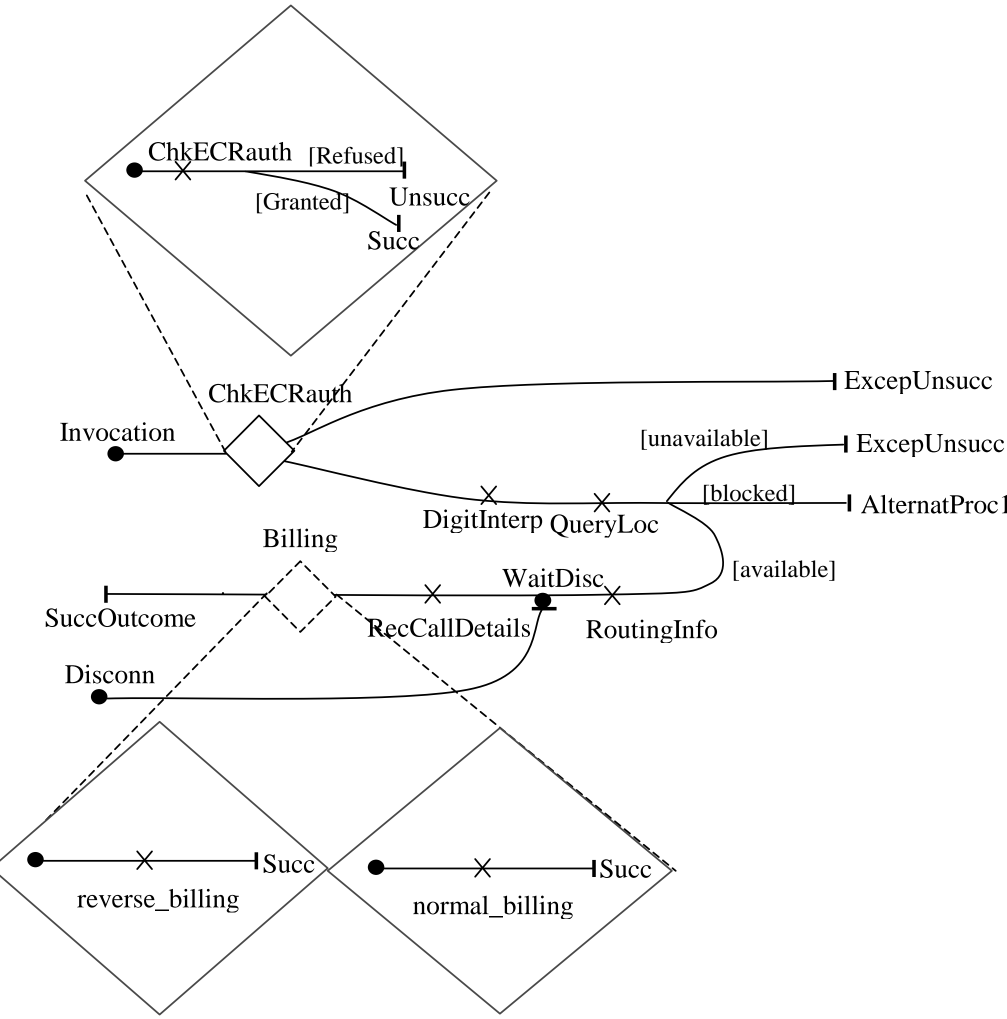


Figure 56 Enhanced Call Routing

Otherwise, the call is routed to the appropriate geographic destination based on the location/position of the mobile and the dialed digits. If the location/position of the mobile is not available at the time of call setup, exception procedures shall be defined locally. If the ECR subscriber has "blocked" its location information, the mobile station location information should not be provided to any service logic application. In such cases, the

location information would not be available to the ECR service logic, and alternative procedures shall be applied. The system should record call detail information and ECR calls may be billed to the subscriber or reverse billed to the ECR number code holder.
This example shows the use of stubs and the waiting place.

7.2 LOTOS Specification for Unbound UCM

WIN LBSS initial requirements have been described as unbound UCMs. At this stage, the system description does not need to be complete. The architecture of the system is still not clear. When analysis needs to be performed, the unbound UCMs are automatically translated into LOTOS specification by using *Ucm2LotosSpec* (Chapter 5). Then, LOTOS scenarios can be extracted from the unbound UCMs automatically by using *Ucm2LotosScenario* and used to validate the LOTOS specification. The following LOTOS process is generated from the UCM for FAM Location Update. The full LOTOS specification for unbound FAM and ECR are shown as examples in Appendix A.

```
Process FAM_Status_Change[start,resp,end] :noexit:=
 (
  (
   start !PowerOn;
   resp !GetProfile;
   sub_FAM_Status_Change_[start,resp,end]
   [ ]
   start !SetInactive;
   sub_FAM_Status_Change_[start,resp,end]
  )
  [ ]
  start !PowerOff;
  sub_FAM_Status_Change_[start,resp,end]
 )
 [ ]
 start !SetActive;
 sub_FAM_Status_Change_[start,resp,end]
```

```
Where
        Process sub_FAM_Status_Change_[start,resp,end] :noexit:=
          resp !NotifyFAM;
          resp !SetStatus;
          (
            sub_FAM_Status_Change_1[start,resp,end]
            [ ]
            resp !SetupMonitoring;
            resp !SendLoc;
            resp !UpdLoc;
            sub_FAM_Status_Change_1[start,resp,end]
          )
        Endproc
        Process sub_FAM_Status_Change_1[start,resp,end] :noexit:=
          resp !NotifySV;
          end !SuccOutCome; stop
        Endproc
Endproc
```

7.3 Bound UCM for WIN LBSS

As the description of the system evolves, the architecture and entities of the system are defined. More detailed information about actions performed and message exchanges between components is added to the UCM. This is an iterative process. In this section, FAM and ECR, as examples, are analyzed. Based on the information in this stage, bound UCMs for them are given. In these bound UCMs, an empty path crossing a component means that the component relays the message from its previous component to its next one along the path. When a path crosses one component to another component, it is assumed that there is a message sent from the first component to the second one.

7.3.1 Bound UCM for Fleet and Asset Management

According to the WIN Reference Model and the functionality of network entities, the responsibilities are assigned to different network entities (components in UCMs) and actions are added in the bound UCM for the FAM service. For example, the action

"GetLoc" in Figure 54 FAM Location Update is refined to four actions "QueryLoc" of SCP, "ReqLoc" of MPC, "CalcLoc" of PDE and "SendLoc" of MPC in Figure 58 Bound UCM for FAM Location Update. Figure 57, Figure 58 and Figure 59 present the bound UCMs for FAM service.

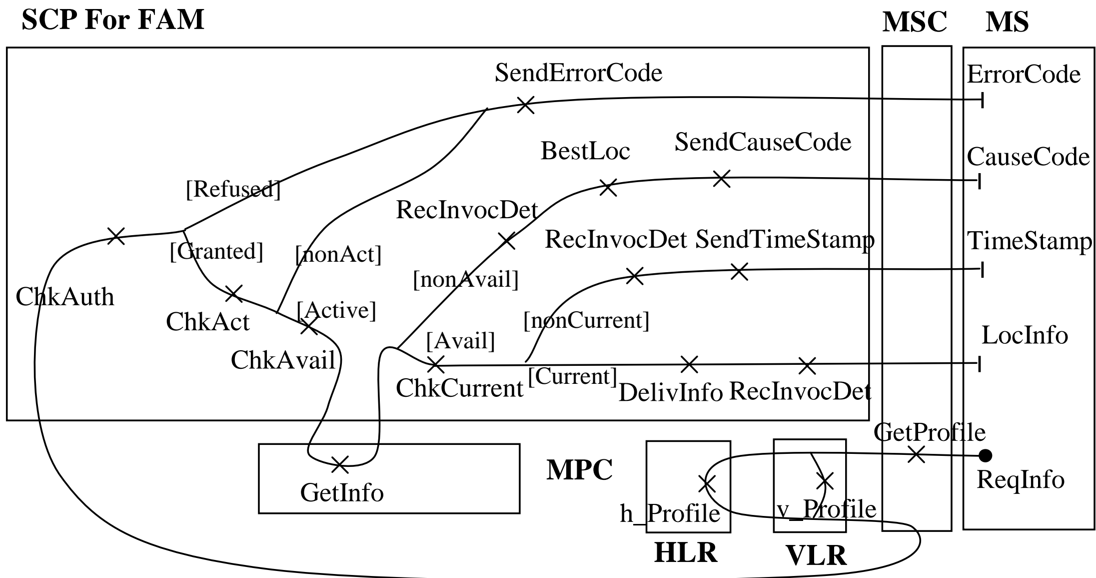


Figure 57 Bound UCM for FAM Request Info

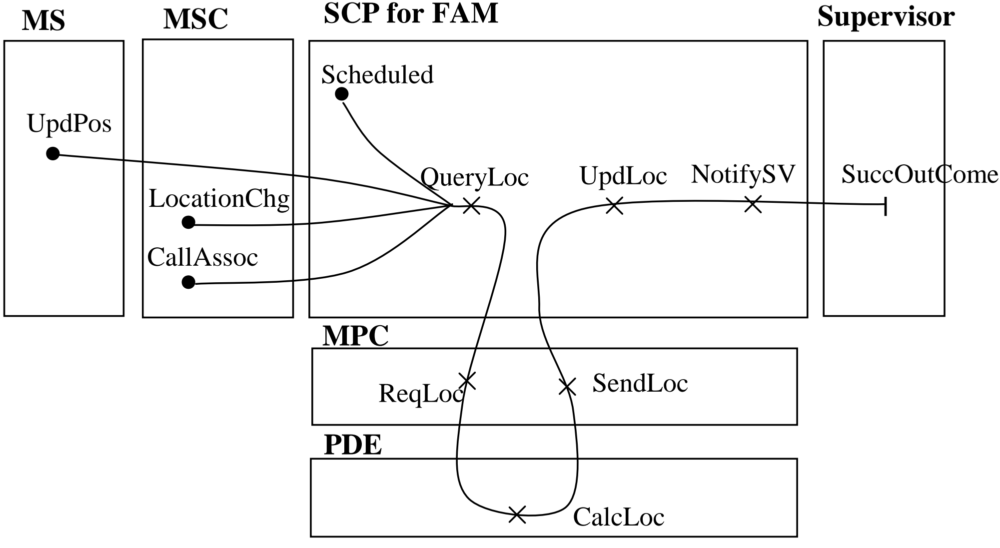


Figure 58 Bound UCM for FAM Location Update

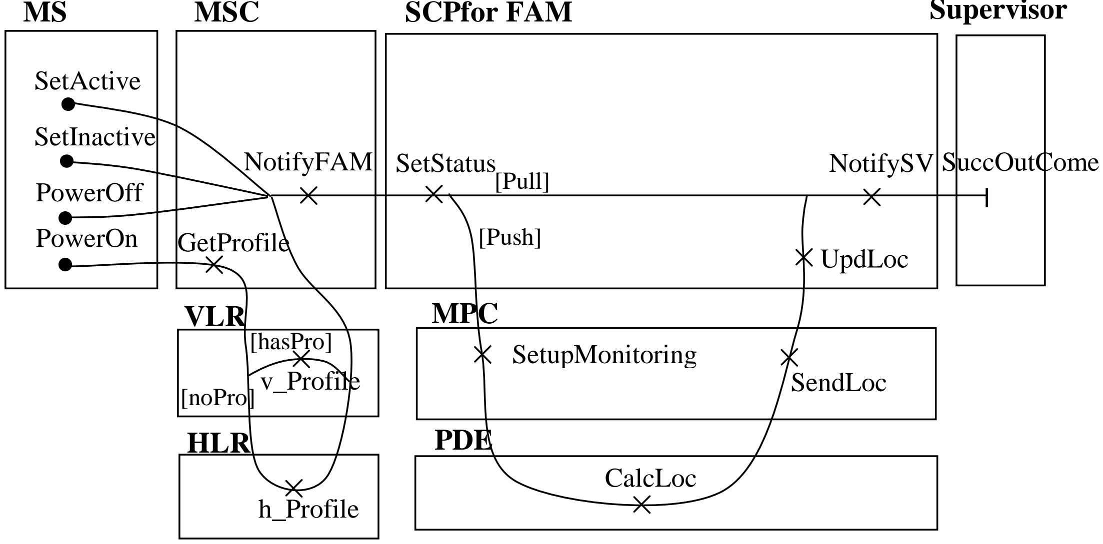


Figure 59 Bound UCM for FAM Status Change

7.3.2 Bound UCM for Enhanced Call Routing

According to the WIN Network Reference Model, all responsibilities are bound to different network entities (components). For instance, Mobile Station (MS) performs the action "Invocation" and "Disconn"; Home Location Register takes care of user profiles; Position Determining Entity is in charge of calculating the precise location of Mobile Station. At this time, the unbound UCMs for ECR become the bound UCMs in Figure 60, Figure 61 and Figure 62.

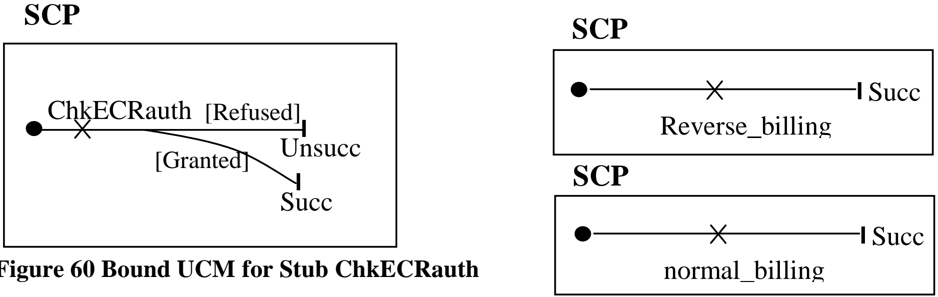


Figure 60 Bound UCM for Stub ChkECRauth

Figure 61 Bound UCM for Stub Billing

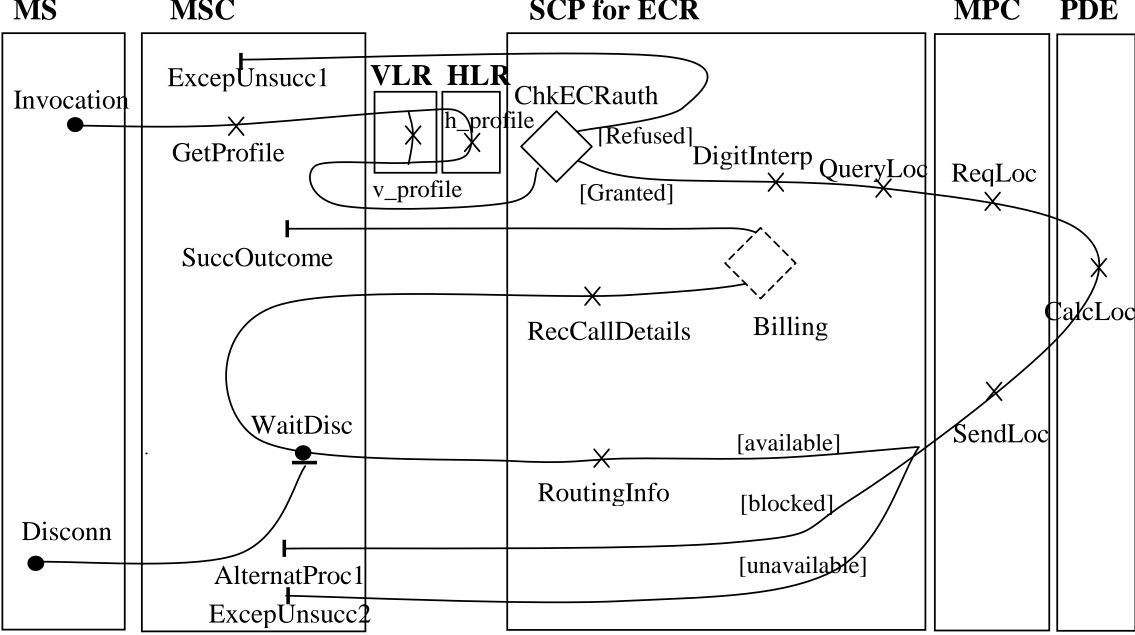


Figure 62 Bound UCM for Enhanced Call Routing

7.4 LOTOS specification for Bound UCMs

Till now, the system structure and scenarios of the WIN LBSS have been described as bound UCMs. In this step, the scenarios described in the bound UCMs are combined automatically to synthesize a LOTOS specification by using *Ucm2LotosSpec*. Then, LOTOS scenarios can be extracted from the bound UCMs automatically by using *Ucm2LotosScenario* and used to validate the LOTOS specification. The full LOTOS specification and some LOTOS scenarios for FAM service are in Appendix B.

7.5 Message Sequence Charts for WIN LBSS

Once all LOTOS scenarios run against the LOTOS specification successfully, LOTOS traces can be generated. With these traces, Message Sequence Charts are produced by *Lotos2Msc* [SteLo]. One scenario in FAM service is chosen as an example to show how

the Message Sequence Chart is generated. The scenario, called PowerOn, describes the FAM invocation when MS is powered on.

All the LOTOS scenarios produced by the tool successfully run against the LOTOS specification. As an example, we show below the LOTOS trace for Scenario PowerOn is generated as follows.

```
LOTOS Trace for Scenario PowerOn:
start !ms !poweron;
ms_to_msc  ! ms ! msc ! m1;
resp !msc !getprofile;
msc_to_vlr  ! msc ! vlr ! m2;
vlr_to_hlr  ! vlr ! hlr ! m2;
resp !hlr !h_profile;
hlr_to_vlr  ! hlr ! vlr ! m3;
vlr_to_msc  ! vlr ! msc ! m3;
resp !msc !notifyfam;
msc_to_scp  ! msc ! scp ! m4;
resp  !scp !setstatus;
scp_to_mpc  ! scp ! mpc ! m5;
resp !mpc !setupmonitoring;
```

```
mpc_to_pde ! mpc ! pde ! m6;
resp  !pde !calcloc;
pde_to_mpc  ! pde ! mpc ! m7;
resp  !mpc !sendloc;
mpc_to_scp ! mpc ! scp ! m8;
resp !scp !updloc;
resp !scp !notifysv;
scp_to_sv  ! scp ! sv ! m9;
end !sv !succoutcome;
```

The Message Sequence Chart for Scenario PowerOn is produced from this LOTOS trace by *Lotos2MSC*. It is shown in Figure 63.

Following is a line-by-line explanation of the MSC:

a) When a MS is powered on, a message (m1) is sent to notify the serving Mobile Switching Center of its activity.
b) The MSC queries the user's profile from VLR (m2).
c) If the MS has not registered in the VLR, HLR is asked to provide the profile of the user (m2).
d) The user's profile is sent to the VLR (m3), VLR stores the profile.
e) The profile is forwarded to the MSC (m3).
f) The MSC does some authentication work and notifies the FAM application (SCP) of the activation of this MS (m4).

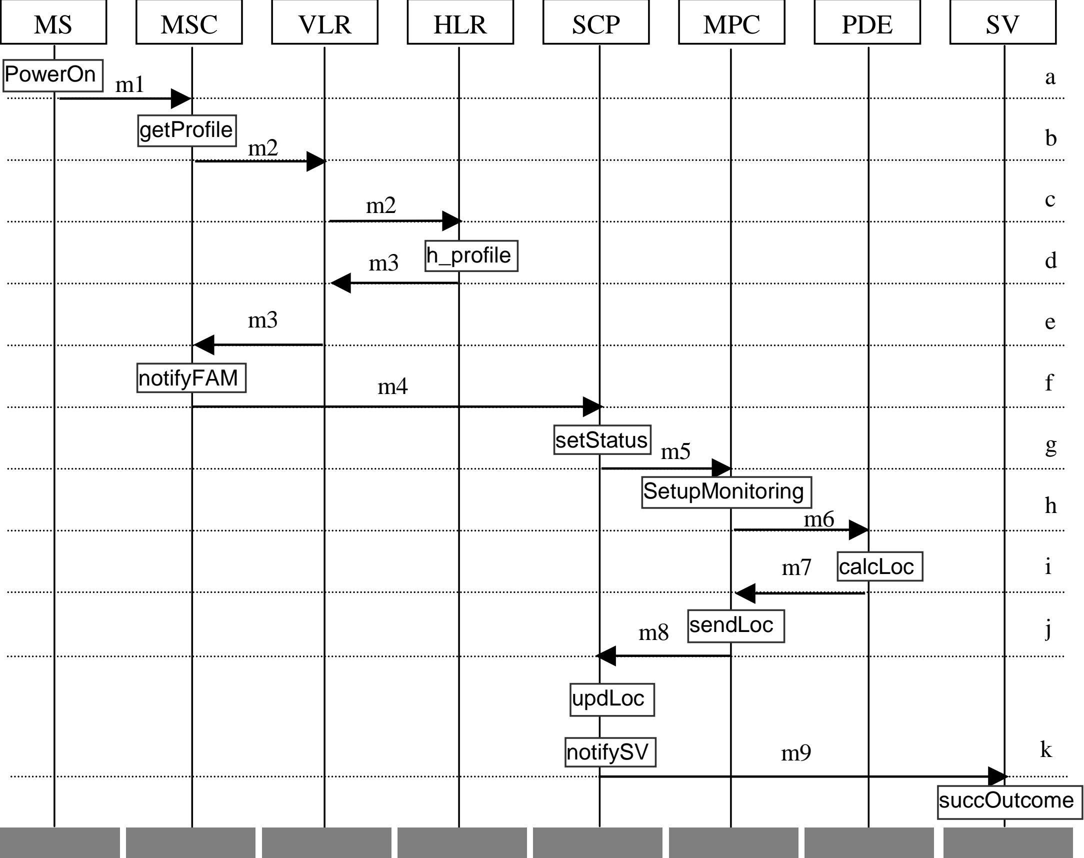


Figure 63 Message Sequence Chart for PowerOn Scenarios in FAM service

g) The status of the MS is set in the FAM application and the FAM application sends the requests for the geoposition of the MS to the MPC (m5).

h) The MPC sets up the monitoring for the position of MS. Also, the MPC selects the appropriate PDE and instructs that PDE to determine the geo position of the MS (m6).

i) The PDE calculates the position of the MS and sends it to the MPC (m7).

j) The MPC stores the position information and also returns it to the FAM application (m8).

k) The FAM application notifies the FAM supervisor of this MS's status and its location (m9).

All the messages produced by our tools are abstract messages (m1, m2, …) and the actions on the MSC instances are added manually in order to make the MSC clear. If concrete messages are needed, these have to be defined. The section 5.6.1 proposes a method to do this, by introducing message names in the UCMs (see Figure 30).

7.6 Conclusion

In the case study of this chapter, we have demonstrated that our approach is feasible on realistic examples, the LBSS service of WIN. We should add that our MSCs were discussed with our industrial collaborator and found by them to be of good quality.

Chapter 8 Conclusions and Future Work

This chapter reviews the contributions of the thesis and discusses the further work related to the topic.

8.1 Contributions

This thesis is focused on implementing and automating a crucial part of the *SPEC-VALUE* methodology and demonstrating its application to Location Based Services in the Wireless Intelligent Network. The essential work in this thesis includes:

1. Two automated translation-tools from UCMs to LOTOS specifications and to LOTOS scenarios. These two tools can be used to assist in the process of system design with the needed level of precision, quality and completeness.
2. Our case study, the WIN LBSS project, describing the requirements using the Use Case Map notation, generating a LOTOS specification using *Ucm2LotosSpec*, producing LOTOS scenarios using *Ucm2LotosScenario*, and finally generating Message Sequence Charts.

Therefore, the thesis provides three major contributions.

Contribution 1: Developing an Automatic LOTOS specification generation tool from UCMs

We establish a relationship between LOTOS and UCM. Then we develop a tool *Ucm2LotosSpec* to generate LOTOS specifications from UCMs (see Chapter 5). Although LOTOS is an ideal specification language for the formal prototyping communication systems, its complexity keeps many designers from using it. This tool can free designers from the complexity and difficulty of LOTOS. The tool is based on the internal representation of UCM in XML used by the tool *UCMNav*.

Contribution 2: Developing an Automatic LOTOS scenario generation tool from UCMs

One of the motivations of this thesis is the automatic generation of Message Sequence Charts from requirements expressed in UCMs. The concept of scenarios is at the foundation of Message Sequence Charts. We develop a tool that generates a set of LOTOS scenarios from UCMs. Once they run against the LOTOS specification successfully, these LOTOS scenarios generate LOTOS traces that can be used to generate Message Sequence Charts (see Chapter 6).

Contribution 3: Implementing and automating part of the *SPEC-VALUE* methodology in WIN LBSS project

The *SPEC-VALUE* methodology provides an approach to describing communicating systems using a semiformal description (UCM) and to produce formal specifications for them. In this thesis, we have presented the automation of some steps of this methodology based on our two other contributions listed above (see Chapter 4) and we have applied this enhanced approach to the design of a protocol for Wireless Intelligent Network (Chapter 7). For this purpose, we

1. Describe the informal requirements of LBSS in UCM for Stage 1.
2. Generate a LOTOS specification from the UCMs of LBSS automatically by using the *Ucm2LotosSpec* tool.
3. Generate LOTOS scenarios from the UCMs of LBSS automatically by using the *Ucm2LotosScenario* tool. The LOTOS scenarios run against the LOTOS specification and LOTOS traces are produced.
4. Produce Message Sequence Charts of LBSS for Stage 2 from the LOTOS traces by using *Lotos2Msc*.

8.2 Future Work

Several research topics can be pursued in the future in order to improve and enhance the implementation of the *SPEC-VALUE* approach:

Improvement of *Ucm2LotosSpec*

It has already been mentioned (see section 5.3) that the tool *Ucm2LotosSpec*, although helpful for fast system prototyping, could be improved in several ways.

1. *Ucm2LotosSpec* does not support the full UCM notation. For example, it doesn't support timers, one of the important notations in UCM. Also, some legal UCMs cannot be handled by the tool (see section 5.3). A future version of the tool should include improvements to support a larger part of the notation.
2. The tool should be enhanced to have the capability to report incompleteness problems and compliance problems in UCMs.
3. The LOTOS code should be optimized. For example, it is assumed that there are two gates between any two components. However it is possible that some of these gates are never used. The tool should remove the unused gates.
4. The tool should handle scenario definitions defined in UCM.

Improvement of Ucm2LotosScenario

When generating UCMs with OR-Fork(s) or dynamic stub(s), the tool generates all possible scenarios without considering the selection conditions. This may lead to explosion in the number of scenarios. An improvement of the tool would be to generate valid scenarios only (section 6.5).

Even with this improvement, it is not always useful to generate all possible scenarios. Some scenarios may be more important than others. Another improvement of the tool would be to allow users to generate only important scenarios.

Enhancement of Use Case Map Notation

The following points identify possible improvements of the Use Case Map Notation:

1. There is no well-defined semantics for the Use Case Map Notation. Some discussion and enhancements on this issue have been provided in [URN-Z152].
2. There is a need to create a new element of the UCM notation to represent channels and messages between components (see section 5.6.1).

3. Use Case Map has a very simple data model including only global boolean variables. It would be better to have a more complete data model in order to represent requirements involving data.

Automatic Generation of Use Case Maps from Message Sequence Charts

In the thesis, we have described how to generate Message Sequence Charts from system requirements expressed by UCM. However, if only Message Sequence Charts are available, Use Case Maps may need to be obtained from these Message Sequence Charts [Cr01] in order to perform formal specification and validation using *SPEC-VALUE*. Therefore, it would be useful to develop a tool to generate UCMs from MSCs automatically. This automatic generation would also be used to keep track of changes during the design.

REFERENCES:

[Am94] D. Amyot. *Formalization of Timethreads Using LOTOS.* Master thesis, Dept. of Computer Science, University of Ottawa, Ottawa, Canada. 1994 http://www.site.uottawa.ca/~damyot/phd/mscthese.pdf

[Am99A] D. Amyot, R. Andrade. *Description of Wireless Intelligent Network Sevices with Use Case Maps*, Proc. Brazilian Symposium on Computer Networks (SBRC'99), Salvador, Brazil, May 1999

[Am99B] D. Amyot. *Use Case Maps Quick Tutorial.* Version 1.0, 1999

[Am99C] D. Amyot, R. Andrade, L. Logrippo, J. Sincennes, Z. Yi. *Formal Methods for Mobility Standards.* IEEE 1999 Emerging Technology Symposium on Wireless Communications & Systems, Richardson, Texas, USA, April 1999.

[Am99D] D. Amyot and L. Logrippo. *Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System*, Computer Communications 23 (8), 1999

[Am01] D. Amyot. *Specification and validation of Telecommunication Systems with Use Case Maps and LOTOS.* Ph.D. thesis, Dept. of Computer Science, University of Ottawa, Ottawa, Canada, 2001 http://www.UseCaseMaps.org/pub/da_phd.pdf

[An00] R. Andrade. *Applying Use Case Maps and Formal Methods to the Development of Wireless Mobile ATM Networks.* Lfm2000, The Fifth NASA Langley Formal methods Workshop, Williamsburg, Virginia, USA, June 2000.
http://www.UseCaseMaps.org/pub/lfm2000.pdf

[Bo87] Tommaso Bolognesi. *Introduction to the ISO Specification Language LOTOS,* Protocol Specification, Testing and Validation VIII, 1988, North-Holland

[Bur96] R. J. A. Buhr and R. S. Casselman. *Use Case Maps for Object-Oriented systems.* Pretice-Hall, 1996

[Bur98] R. J. A. Buhr. *Use Case Maps as Architectural Entities for Complex Systems.* IEEE Transactions on software Engineering, Special Issues on Scenario Management. Vol. 24, No. 12, pages 1131-1155, 1998

[Ca93] J. Cameron, N. Griffeth, Y.J. Lin, M. Nilson, W. Shnure, and H. Velthuijsen. *Toward a Feature Interaction Benchmark for IN and Beyond*. IEEE Communications 31, 64-69, March, 1993

[Ch01] L. Charfi, *Formal Modeling and Test Generation Automation With UCMs and LOTOS.* Master thesis, Dept. of Computer Science, University of Ottawa, Ottawa, Canada. 2001

[Cr01] R. G. Crespo, *Semantic of Message Sequence Charts with Use Case Maps.* Submitted for publication.

[CTIA1] Cellular Telecommunications Industry Association Wireless Intelligent Network Sub-Task Group. *Standards Requirements Document: Wireless Intelligent Network Enhancement to Support Location-Based Services [WIN Phase III]*. December 13, 1999

[Ehr85] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specification - 1*, Springer-Verlag, Berlin, 1985.

[GaLo] S. Gallouzi, L.Logrippo, A. Obaid, *An Expressive Trace Theory For LOTOS,* Protocol Specification, testing and Verification XI, Proceedings of the IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Stockholm, Sweden, June 1991

[GaSi] H. Garavel, J. Sifakis. *Compilation and Verification of LOTOS Specifications*, Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification, pages 379-394, Ottawa Canada, June 1990.

[Ho85] C. A. R. Hoare, *Communicating Sequential Processe*s, Prentice-Hall International, 1985. ISBN: 0131532715.

[IN] The International Engineering Consortium, *International Intelligent Network* http://www.iec.org/online/tutorials/intern_in/topic01.html

[Intra] Intrado Inc. *ALI to MPC Interface Specification For TCP/IP Implementation of TIA/EAA/J-STD-036 E2*, Boulder, Colorado USA, December 2001.

[ITU-I130] International Telecommunications Union (ITU-T), *Recommendation I.130: Method for the characterization of telecommunication services supported by an ISDN and network capabilities of ISDN,* 1998

[ITU-Q65] International Telecommunications Union (ITU-T), *Recommendation Q.65: The unified functional methodology for the characterization of services and network capabilities including alternative object-oriented techniques,* 2000

[ITU-Z120] International Telecommunications Union (ITU-T), *Recommendation Z.120: Message Sequence Charts,* 2000.

[ITU-Z100] International Telecommunications Union (ITU-T), *Recommendation Z.100: Specification and Description Language,* 2000.

[Lo91] L. Logrippo, Mohammed Faci and Mazen Haj-Hussein. *An Introduction to LOTOS: Learning by Examples. Computer Network and ISDN System #23,* North-Holland, 1992

[Lo00] L. Logrippo. *Towards a High Quality Standards Development Process for Mobile Telephony.* Presentation at Wireless Forum IX, Ottawa Canada, 2000

[Lo02] L. Logrippo. *Network Architectures, Services, Protocols and Standards*. Lecture notes, University of Ottawa, 2002.

[LOLA] *LOTOS Laboratory User Manual*, Version 3R6.

[Mi89] R. Milner, *Communication and Concurrency,* Prentice-Hall, New York, 1989. ISBN 0131149849.

[Mi01] A. Miga, D. Amyot, F. Bordeleau, D. Cameron, M. Woodside, *Deriving Message Sequence Charts from Use Case maps Scenario Specifications.* SDL Forum, May, 2001.

[OSI89] OSI - IS 8807, *"Information Processing Systems - Open Systems Interconnection – LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", International Standard IS 8807 (E. Brinksma, ed.),* 1989.

[PL91] S. Pavon and M. Llamas. *The Testing Functionalities of LOLA*. In Juan Quemada, Jose A. Manas, and Enrique Vazques, editors, *Formal Description*

Techniques, III, pages 559-562, Madrid (ES), 1991, IFIP, Elsevier Science B.V. (North-Holland). Proceedings FORTE'90, 5-8 November, 1990

[QFM87] J. Quemada, A. Fernandez, and J. A. Marias. *LOLA: Design and Verification of Protocols Using LOTOS.* In *IBERIAN Conference on Data Communications,* Lisbon, May, 1987.

[SteLo] B. Stépien, L. Logrippo, *Graphic visualization and Animation of LOTOS execution trace,* Submitted for publication

[TIA/EIA/IS-771] Telecommunication Industry Association, *Wireless Intelligent Network*, July 1999

[TIA/EIA/IS-826] Telecommunication Industry Association, *WIN Capabilities for Pre-paid Charging*, September 2000

[TIA/EIA/IS-848] Telecommunication Industry Association, *Enhanced Charging Service,* December 2000

[UCM] *Use Case Map Navigator User's Manual, version 1.10,* June 1999

[URN-Z152] *Draft Specification of the Use Case Map Notation Z.152.* Geneva, February 2002

[WIN98] Telecommunications Industry Association, *WIRELESS INTELLIGENT NETWORK TIA/EIA-664 MODIFICATION: WIN Phase I Ballot version PN3661*

[WIN00] The International Engineering Consortium, *Wireless Intelligent Network* http://www.webproforum.com/dsc/

[WINTIA99] Telecommunications Industry Association, *Standards Requirements Document -- Wireless Intelligent Network Enhancements to Support Location-Based Services [WIN Phase III],* December 1999

[WINTIA00] Telecommunications Industry Association, *Wireless Features Description: Location Based Services: TIA/EIA-PN4818,* September 2000

[Yi00] Z. Yi, *CNAP Specification and Validation: A Design Methodology Using LOTOS and UCM.* Master thesis, Dept. of Computer Science, University of Ottawa, Ottawa, Canada. 2000

[3GPP2] *Location-Based Services System (LBSS): Stage 1 Description.* S.R0019 v1.0.0, September 22, 2000

ACRONYMS

AIN	Advanced Intelligent Network
AMA	Automatic Message Accounting
ANSI	American National Standards Institute
BCSM	Basic Call State Model
CCF	Call Control Function
CDR	Call Detail Record
CS-n	Capability Set n
CTIA	Cellular Telecommunications Industry Association
DFP	Distributed Functional Plane
ECR	Enhanced Call Routing
FAM	Fleet and Asset Management
FE	Functional Entity
FM-CMM	Formal Methods-Capability Maturity Model
HLR	Home Location Register
IMSI	International Mobile Station Identity
IN	Intelligent Network
INAP	Intelligent Network Application Protocol
IP	Intelligent Peripheral
ISDN	Integrated Services Digital Network
ITU	International Telecommunications Union
LBC	Location-Based Charging
LBIS	Location-Based Information Service
MPC	Mobile Positioning Center
MS	Mobile Station
MSC	Mobile Switching Center
NE	Network Entity
NRM	Network Reference Model
PDE	Position Determining Entity
PN	Project Number

PSTN	Public Switched Telephone Network
SCE	Service Creation Environment
SCF	Service Control Function
SCP	Service Control Point
SDF	Service Data Function
SMS	Service Management System
SN	Service Node
SRD	Standards Requirements Document
SSF	Service Switching Function
SS7	Signaling System 7
SSP	Service Switching Point
TDP	Trigger Detection Point
TIA	Telecommunications Industry Association
WIN	Wireless Intelligent

APPENDIX A:

LOTOS Specification for FAM Unbound UCM

```
Specification fam[start,resp,end] :noexit
(**************************************Data Part***********************************************************)
Library Boolean
Endlib

Type actionType is
Sorts responsibility
opns
        GetLocation, UpdLoc, NotifySV, ChkAuth, SendErrorCode, ChkAct, ChkAvail,
        BestLoc, SendCauseCode, RecInvocDet, ChkCurrent, SendTimeStamp,
        DelivInfo, NotifyFAM, SetStatus, SetupMonitoring, SendLoc, GetProfile:-> responsibility
Endtype

Type startType is
Sorts start
opns
        Scheduled, UpdPos, ReqInfo, LocationChg, CallAssoc, PowerOn, PowerOff, SetActive,
        SetInactive:->start
Endtype

Type epointType is
Sorts epoint
opns
        SuccOutCome, ErrorCode, CauseCode, TimeStamp, LocInfo:->epoint
Endtype

Type compType is
Sorts comp
opns
        Env_m0, Env_m1, Env_m2, Env:->comp
Endtype

Type preConditions is Boolean
opns
        NonAvail, Avail, NonCurrent, Current, refused, Granted, NonAct, Active, notVisiting,
        Push, Pull:-> Bool
eqns
ofsort Bool
        NonAvail = true;
        Avail = true;
        NonCurrent = true;
        Current = true;
        refused = true;
        Granted = true;
        NonAct = true;
        Active = true;
        notVisiting = true;
        Push = true;
        Pull = true;
Endtype
```



```
(**************************************Control Part*********************************************************)
Behaviour
        Env[start,resp,end]
Where
        Process Env[start,resp,end]:noexit:=
          Env_m0[start,resp,end] (* FAM Location Update *)
          |||
          Env_m1[start,resp,end] (* FAM Request of Location/Status *)
          |||
          Env_m2[start,resp,end] (* FAM Status Change *)
        Where
                Process Env_m0[start,resp,end]:noexit:=
                  h0_Env_m0[start,resp,end]
                  |||
                  h4_Env_m0[start,resp,end]
                  |||
                  h10_Env_m0[start,resp,end]
                  |||
                  h13_Env_m0[start,resp,end]
                Where
                        Process h0_Env_m0[start,resp,end]:noexit:=
                        start !Scheduled;
                          ( sub_m0_0_Env[start,resp,end]
                            |||
                            h0_Env_m0[start,resp,end]
                          )
                        Endproc
                        Process h4_Env_m0[start,resp,end]:noexit:=
                        start !CallAssoc;
                          ( sub_m0_0_Env[start,resp,end]
                            |||
                            h4_Env_m0[start,resp,end]
                          )
                        Endproc
                        Process h10_Env_m0[start,resp,end]:noexit:=
                        start !LocationChg;
                          ( sub_m0_0_Env[start,resp,end]
                            |||
                            h10_Env_m0[start,resp,end]
                          )
                        endproc
                        Process h13_Env_m0[start,resp,end]:noexit:=
                        start !UpdPos;
                          ( sub_m0_0_Env[start,resp,end]
                            |||
                            h13_Env_m0[start,resp,end]
                          )
                        Endproc
                        Process sub_m0_0_Env[start,resp,end]:noexit:=
                          resp !GetLocation;
                          resp !UpdLoc;
                          resp !NotifySV;
                          end !SuccOutCome;
                          stop
                        Endproc
```



```
Endproc

Process Env_m1[start,resp,end]:noexit:=
  h22_Env_m1[start,resp,end]
Where
        Process h22_Env_m1[start,resp,end]:noexit:=
        start !ReqInfo;
        ( resp !ChkAuth;
          (
            [refused]->
            sub_m1_0_Env[start,resp,end]
            []
            [Granted]->
            resp !ChkAct;
            (
              [Active]->
              resp !ChkAvail;
              (
                [NonAvail]->
                resp !BestLoc;
                resp !SendCauseCode;
                resp !RecInvocDet;
                end !CauseCode;
                stop
                []
                [Avail]->
                resp !ChkCurrent;
                (
                  [NonCurrent]->
                  resp !SendTimeStamp;
                  resp !RecInvocDet;
                  end !TimeStamp;
                  stop
                  []
                  [Current]->
                  resp !DelivInfo;
                  resp !RecInvocDet;
                  end !LocInfo;
                  stop
                )
              )
              []
              [NonAct]->
              sub_m1_0_Env[start,resp,end]
            )
          )
          |||
          h22_Env_m1[start,resp,end]
        )
        Endproc
        Process sub_m1_0_Env[start,resp,end]:noexit:=
          resp !SendErrorCode;
          end !ErrorCode;
          stop
        Endproc
```



```
Endproc
Process Env_m2[start,resp,end]:noexit:=
  h95_Env_m2[start,resp,end]
  |||
  h81_Env_m2[start,resp,end]
  |||
  h123_Env_m2[start,resp,end]
  |||
  h99_Env_m2[start,resp,end]
Where
        Process h81_Env_m2[start,resp,end]:noexit:=
        start !PowerOn;
        ( resp !GetProfile;
          sub_m2_0_Env[start,resp,end]
          |||
          h81_Env_m2[start,resp,end]
        )
        Endproc
        Process h95_Env_m2[start,resp,end]:noexit:=
        start !SetInactive;
        ( sub_m2_0_Env[start,resp,end]
          |||
          h95_Env_m2[start,resp,end]
        )
        Endproc
        Process h99_Env_m2[start,resp,end]:noexit:=
        start !PowerOff;
        ( sub_m2_0_Env[start,resp,end]
          |||
          h99_Env_m2[start,resp,end]
        )
        Endproc
        Process h123_Env_m2[start,resp,end]:noexit:=
        start !SetActive;
        ( sub_m2_0_Env[start,resp,end]
          |||
          h123_Env_m2[start,resp,end]
        )
        Endproc
        Process sub_m2_0_Env[start,resp,end]:noexit:=
          resp !NotifyFAM;
          resp !SetStatus;
          (
            [Pull]->
            sub_m2_1_Env[start,resp,end]
            []
            [Push]->
            resp !SetupMonitoring;
            resp !SendLoc;
            resp !UpdLoc;
            sub_m2_1_Env[start,resp,end]
          )
        Endproc
        Process sub_m2_1_Env[start,resp,end]:noexit:=
          resp !NotifySV;
```

```
end !SuccOutCome;
stop
Endproc
Endproc
Endproc
(*************************************Scenario*************************************************************)
Process scenario1[start,resp,end,scenario1]: noexit:=
    start!Scheduled;
    resp!GetLocation;
    resp!UpdLoc;
    resp!NotifySV;
    end!SuccOutCome;
    scenario1;
    stop
Endproc

Process scenario5[start,resp,end,scenario5]: noexit:=
    start!ReqInfo;
    resp!ChkAuth;
    resp!SendErrorCode;
    end!ErrorCode;
    scenario5;
    stop
Endproc

Process scenario15[start,resp,end,scenario15]: noexit:=
    start!PowerOff;
    resp!NotifyFAM;
    resp!SetStatus;
    resp!SetupMonitoring;
    resp!SendLoc;
    resp!UpdLoc;
    resp!NotifySV;
    end!SuccOutCome;
    scenario15;
    stop
Endproc
Endspec
```


LOTOS Specification for ECR Unbound UCM

```
Specification ecr[start,resp,end,to_handler,from_handler,WaitDisc] :noexit
(*************************************Data Part*************************************************************)
Library Boolean
Endlib

Type actionType is
Sorts responsibility
opns
    ChkAuth, normal_billing, reverse_billing, h37, h40, h44, DigitInterp, QueryLoc, h54,
    RoutingInfo, RecCallDetails, h38, h49, h48, h53, h39, para:-> responsibility
Endtype

Type startType is
Sorts start
opns
    Invocation, Disconn, check, billing:->start
Endtype

Type epointType is
Sorts epoint
opns
    SuccOutcome, Unsucc, Succ, ExcepUnsucc1, ExcepUnsucc2, AlternatProc1,
    endDisconn:->epoint
Endtype

Type compType is
Sorts comp
opns
    Env_m0, Env_m1, Env_m2, Env_m3, Env, from, to:->comp
Endtype

Type preConditions is Boolean
opns
    available,unavailable,blocked, Refused, succPath, Granted:-> Bool
eqns
ofsort Bool
    available = true;
    unavailable = true;
    blocked = true;
    Refused = true;
    succPath = true;
    Granted = true;
Endtype

(*************************************Control Part*************************************************************)
Behaviour
    Env[start,resp,end,to_handler,from_handler,WaitDisc]
    |[to_handler,from_handler]|
    plugin_handler[to_handler,from_handler]
Where
    Process plugin_handler[to_handler,from_handler]:noexit:=
      to_handler ?from:comp ?to:comp ?para:responsibility;
```



```
  from_handler !from !to !para;
  plugin_handler[to_handler,from_handler]
Endproc

Process Env[start,resp,end,to_handler,from_handler,WaitDisc]:noexit:=
  Env_m0[start,resp,end,to_handler,from_handler,WaitDisc] (* root map *)
  |||
  Env_m1[start,resp,end,to_handler,from_handler] (* plugin ChkECRauth *)
  |||
  Env_m2[start,resp,end,to_handler,from_handler] (* plugin Normal-Billing *)
  |||
  Env_m3[start,resp,end,to_handler,from_handler] (* plugin Reverse_Billing*)
Where
    Process Env_m0[start,resp,end,to_handler,from_handler,WaitDisc]:noexit:=
      h14_Env_m0[start,resp,end,to_handler,from_handler,WaitDisc]
      |[WaitDisc]|
      h0_Env_m0[start,resp,end,to_handler,from_handler,WaitDisc]
    Where
        Process
    h14_Env_m0[start,resp,end,to_handler,from_handler,WaitDisc]:noexit:=
        start !Disconn;
        ( end !endDisconn;
          WaitDisc;stop
          |||
          h14_Env_m0[start,resp,end,to_handler,from_handler,WaitDisc]
        )
        Endproc
        Process
    h0_Env_m0[start,resp,end,to_handler,from_handler,WaitDisc]:noexit:=
        start !Invocation;
        (  to_handler !Env_m0 !Env_m1 !h39;
          sub_m0_0_Env[start,resp,end,to_handler,from_handler,WaitDisc]
          |||
          h0_Env_m0[start,resp,end,to_handler,from_handler,WaitDisc]
        )
        Endproc
        Process
    sub_m0_0_Env[start,resp,end,to_handler,from_handler,WaitDisc]:noexit:=
        (
          from_handler !Env_m1 !Env_m0 !h40;
          end !ExcepUnsucc1;
          stop
        )
        []
        (
          from_handler !Env_m1 !Env_m0 !h44;
          resp !DigitInterp;
          resp !QueryLoc;
          (
                [unavailable]->
                end !ExcepUnsucc2;
                stop
                []
                [blocked]->
                end !AlternatProc1;
```



```
                stop
                []
                [available]->
                resp !RoutingInfo;
                WaitDisc;
                resp !RecCallDetails;
                (
                  to_handler !Env_m0 !Env_m2 !h48;

        sub_m0_1_Env[start,resp,end,to_handler,from_handler,WaitDisc]
                  []
                  to_handler !Env_m0 !Env_m3 !h53;

        sub_m0_1_Env[start,resp,end,to_handler,from_handler,WaitDisc]
                )
          )
        )
        Endproc
        Process
    sub_m0_1_Env[start,resp,end,to_handler,from_handler,WaitDisc]:noexit:=
          from_handler !Env_m2 !Env_m0 !h49;
          end !SuccOutcome;
          stop
          []
          from_handler !Env_m3 !Env_m0 !h54;
          end !SuccOutcome;
          stop
        Endproc
    Endproc
    Process Env_m1[start,resp,end,to_handler,from_handler]:noexit:=
     h39_Env_m1[start,resp,end,to_handler,from_handler]
    Where

        Process h39_Env_m1[start,resp,end,to_handler,from_handler]:noexit:=
        from_handler ?from:comp !Env_m1 !h39;
        ( start !check;
          resp !ChkAuth;
          (
           [Refused]->
           end !Unsucc;
           to_handler !Env_m1 !from  !h40;
           stop
           []
           [Granted]->
           end !Succ;
           to_handler !Env_m1 !from  !h44;
           stop
          )
          |||
          h39_Env_m1[start,resp,end,to_handler,from_handler]
        )
        Endproc
    Endproc
    Process Env_m2[start,resp,end,to_handler,from_handler]:noexit:=
     h48_Env_m2[start,resp,end,to_handler,from_handler]
```

```
        Where
                Process h48_Env_m2[start,resp,end,to_handler,from_handler]:noexit:=
                from_handler ?from:comp !Env_m2 !h48;
                ( start !billing;
                  resp !normal_billing;
                  end !Succ;
                  to_handler !Env_m2 !from !h49;
                  stop
                  |||
                  h48_Env_m2[start,resp,end,to_handler,from_handler]
                )
                Endproc
        Endproc
        Process Env_m3[start,resp,end,to_handler,from_handler]:noexit:=
         h53_Env_m3[start,resp,end,to_handler,from_handler]
        Where
                Process h53_Env_m3[start,resp,end,to_handler,from_handler]:noexit:=
                from_handler ?from:comp !Env_m3 !h53;
                ( start !billing;
                  resp !reverse_billing;
                  end !Succ;
                  to_handler !Env_m3 !from  !h54;
                  stop
                  |||
                  h53_Env_m3[start,resp,end,to_handler,from_handler]
                )
                Endproc
        Endproc
    Endproc
(************************************Scenario************************************************************)
    Process scenario1[start,resp,end,to_handler,from_handler,scenario1]: noexit:=
        start !Invocation;
        to_handler !Env_m0 !Env_m1 !h39;
        from_handler !Env_m0 !Env_m1 !h39;
        start !check;
        (*Check if requesting unit has authorization to obtain the requested information.*)
        resp !ChkAuth;
        end !Unsucc;
        to_handler !Env_m1 !Env_m0 !h40;
        from_handler !Env_m1 !Env_m0 !h40;
        end !ExcepUnsucc1;
        scenario1;
        stop
    Endproc
    Process scenario2[start,resp,end,to_handler,from_handler,scenario2]: noexit:=
        start !Invocation;
        to_handler !Env_m0 !Env_m1 !h39;
        from_handler !Env_m0 !Env_m1 !h39;
        start !check;
        (*Check if requesting unit has authorization to obtain the requested information.*)
        resp !ChkAuth;
        end !Succ;
        to_handler !Env_m1 !Env_m0 !h44;
        from_handler !Env_m1 !Env_m0 !h44;
```



```
        (*The Serving MSC interprets the dialed digits, e.g. either as a directory number
        or as an abbreviated dialing string. For instance 427 might be interpreted as
        GAS.*)
        resp !DigitInterp;
        (*The current User Location is sought.*)
        resp !QueryLoc;
        end !ExcepUnsucc2;
        scenario2;
        stop
    Endproc
    Process scenario3[start,resp,end,to_handler,from_handler,scenario3]: noexit:=
        start !Invocation;
        to_handler !Env_m0 !Env_m1 !h39;
        from_handler !Env_m0 !Env_m1 !h39;
        start !check;
        (*Check if requesting unit has authorization to obtain the requested information.*)
        resp !ChkAuth;
        end !Succ;
        to_handler !Env_m1 !Env_m0 !h44;
        from_handler !Env_m1 !Env_m0 !h44;
        (*The Serving MSC interprets the dialed digits, e.g. either as a directory number
        or as an abbreviated dialing string. For instance 427 might be interpreted as
        GAS.*)
        resp !DigitInterp;
        (*The current User Location is sought.*)
        resp !QueryLoc;
        end !AlternatProc1;
        scenario3;
        stop
    Endproc
    Process scenario4[start,resp,end,to_handler,from_handler,WaitDisc,scenario4]: noexit:=
        start !Invocation;
        to_handler !Env_m0 !Env_m1 !h39;
        from_handler !Env_m0 !Env_m1 !h39;
        start!check;
        (*Check if requesting unit has authorization to obtain the requested information.*)
        resp !ChkAuth;
        end !Succ;
        to_handler !Env_m1 !Env_m0 !h44;
        from_handler !Env_m1 !Env_m0 !h44;
        (*The Serving MSC interprets the dialed digits, e.g. either as a directory number
        or as an abbreviated dialing string. For instance 427 might be interpreted as
        GAS.*)
        resp !DigitInterp;
        (*The current User Location is sought.*)
        resp !QueryLoc;
        (*The address or directory number of the called party is provided so that the call
        can be routed.*)
        resp !RoutingInfo;
        start !Disconn;
        end !endDisconn;
        WaitDisc;
        (*Record call details of the billing, optionnally billing the owner of the ECR
        number.*)
        resp !RecCallDetails;
```



```
        to_handler !Env_m0 !Env_m2 !h48;
        from_handler !Env_m0 !Env_m2 !h48;
        start !billing;
        resp !normal_billing;
        end !Succ;
        to_handler !Env_m2 !Env_m0 !h49;
        from_handler !Env_m2 !Env_m0 !h49;
        end !SuccOutcome;
        scenario4;
        stop
    Endproc
    Process scenario5[start,resp,end,to_handler,from_handler,WaitDisc,scenario5]: noexit:=
        start !Invocation;
        to_handler !Env_m0 !Env_m1 !h39;
        from_handler !Env_m0 !Env_m1 !h39;
        start !check;
        (*Check if requesting unit has authorization to obtain the requested information.*)
        resp !ChkAuth;
        end !Succ;
        to_handler !Env_m1 !Env_m0 !h44;
        from_handler !Env_m1 !Env_m0 !h44;
        (*The Serving MSC interprets the dialed digits, e.g. either as a directory number
        or as an abbreviated dialing string. For instance 427 might be interpreted as
        GAS.*)
        resp !DigitInterp;
        (*The current User Location is sought.*)
        resp !QueryLoc;
        (*The address or directory number of the called party is provided so that the call
        can be routed. *)
        resp !RoutingInfo;
        start !Disconn;
        end !endDisconn;
        WaitDisc;
        (*Record call details of the billing, optionnally billing the owner of the ECR
        number.*)
        resp !RecCallDetails;
        to_handler !Env_m0 !Env_m3 !h53;
        from_handler !Env_m0 !Env_m3 !h53;
        start !billing;
        resp !reverse_billing;
        end !Succ;
        to_handler !Env_m3 !Env_m0 !h54;
        from_handler !Env_m3 !Env_m0 !h54;
        end !SuccOutcome;
        scenario5;
        stop
    Endproc
Endspec
```

APPENDIX B:

LOTOS Specification for FAM Bound UCM

```
Specification ucm2lotos[start, resp, end, MSC_to_MS, MS_to_MSC, PDE_to_MPC,
MPC_to_PDE, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC, MSC_to_SCP, SV_to_SCP,
SCP_to_SV, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC, MSC_to_VLR] :noexit
(**********************************Data Part**********************************************************)
Library Boolean
Endlib

Type actionType is
Sorts responsibility
opns
        QueryLoc, ReqLoc, h20, h23, CalcLoc, h25, SendLoc, UpdLoc, h14, NotifySV, h17, h6,
        h10, h12, h13,GetProfile, h73, h89, h87, h_Profile, h91, h90, h70, ChkAuth,
        SendErrorCode, h59, h29, ChkAct, ChkAvail, GetInfo, h78, h77, BestLoc, RecInvocDet,
        SendCauseCode, h81, h41, ChkCurrent, SendTimeStamp, h84, h42, DelivInfo, h96, h97,
        v_Profile, h113, NotifyFAM, h114, SetStatus, h101, SetupMonitoring, h107, h145, h132,
        h125, h129, h130, h102, h135, h133, h140, h136:-> responsibility
Endtype

Type startType is
Sorts start
opns
        Scheduled, UpdPos, ReqInfo, LocationChg, CallAssoc, PowerOn, PowerOff, SetActive,
        SetInactive:->start
Endtype

Type epointType is
Sorts epoint
opns
        SuccOutCome, ErrorCode, CauseCode, TimeStamp, LocInfo:->epoint
Endtype

Type compType is
Sorts comp
opns
        HLR_m0, MPC_m0, MS_m0, MSC_m0,PDE_m0, SCP_m0, SV_m0, VLR_m0, Env_m0,
        HLR_m1, MPC_m1, MS_m1, MSC_m1, PDE_m1, SCP_m1, SV_m1, VLR_m1, Env_m1,
        HLR_m2, MPC_m2, MS_m2, MSC_m2, PDE_m2, SCP_m2, SV_m2, VLR_m2, Env_m2,
        HLR, MPC, MS, MSC, PDE, SCP, SV, VLR, Env:->comp
Endtype

Type preConditions is Boolean
opns
        NonAvail, Avail, NonCurrent, Current, refused, Granted, NonAct, Active, has_vlr_Profile,
        no_vlr_profile, notVisiting, Push, Pull, hasPro, noPro:-> Bool
eqns
ofsort Bool
        NonAvail = true;
        Avail = true;
        NonCurrent = true;
        Current = true;
```



```
        refused = true;
        Granted = true;
        NonAct = true;
        Active = true;
        has_vlr_Profile = true;
        no_vlr_profile = true;
        notVisiting = true;
        Push = true;
        Pull = true;
        hasPro = true;
        noPro = true;
Endtype
(**********************************Control Part*******************************************************)
Behaviour
        (((((((HLR[resp, HLR_to_VLR, VLR_to_HLR]
        |||
        MPC[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP, SCP_to_MPC])
        |||
        MS[start, end, MS_to_MSC, MSC_to_MS])
        |[MSC_to_MS,MS_to_MSC]|
        MSC[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP, SCP_to_MSC,
        MSC_to_VLR, VLR_to_MSC])
        |[PDE_to_MPC,MPC_to_PDE]|
        PDE[resp,PDE_to_MPC,MPC_to_PDE])
        |[SCP_to_MPC,MPC_to_SCP,SCP_to_MSC,MSC_to_SCP]|
        SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC, MSC_to_SCP,
        SCP_to_SV, SV_to_SCP])
        |[SV_to_SCP,SCP_to_SV]|
        SV[end, SV_to_SCP, SCP_to_SV])
        |[VLR_to_HLR,HLR_to_VLR,VLR_to_MSC,MSC_to_VLR]|
        VLR[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC, MSC_to_VLR])
Where
        Process HLR[resp, HLR_to_VLR, VLR_to_HLR]:noexit:=
         HLR_m1[resp, HLR_to_VLR, VLR_to_HLR]
         |||
         HLR_m2[resp, HLR_to_VLR, VLR_to_HLR]
        Where
                Process HLR_m1[resp, HLR_to_VLR, VLR_to_HLR]:noexit:=
                 h27_HLR_m1[resp, HLR_to_VLR, VLR_to_HLR]
                Where
                        Process h27_HLR_m1[resp, HLR_to_VLR, VLR_to_HLR]:noexit:=
                        VLR_to_HLR !VLR !HLR !h87;
                        ( resp !hlr !h_Profile;
                          HLR_to_VLR !HLR !VLR !h91;
                          stop
                          |||
                          h27_HLR_m1[resp, HLR_to_VLR, VLR_to_HLR]
                        )
                        Endproc
                Endproc
                Process HLR_m2[resp, HLR_to_VLR, VLR_to_HLR]:noexit:=
                 h100_HLR_m2[resp, HLR_to_VLR, VLR_to_HLR]
                Where
                        Process h100_HLR_m2[resp, HLR_to_VLR, VLR_to_HLR]: noexit:=
                        VLR_to_HLR !VLR !HLR !h140;
```



```
                        ( resp !hlr !h_Profile;
                          HLR_to_VLR !HLR !VLR !h136;
                          stop
                          |||
                          h100_HLR_m2[resp, HLR_to_VLR, VLR_to_HLR]
                        )
                        Endproc
        Endproc
Endproc

Process MPC[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP, SCP_to_MPC]:
noexit:=
  MPC_m0[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP, SCP_to_MPC]
  |||
  MPC_m1[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP, SCP_to_MPC]
  |||
  MPC_m2[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP, SCP_to_MPC]
Where
        Process MPC_m0[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
SCP_to_MPC]: noexit:=
         h0_MPC_m0[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
SCP_to_MPC]
        Where
                Process h0_MPC_m0[resp, MPC_to_PDE, PDE_to_MPC,
        MPC_to_SCP, SCP_to_MPC]: noexit:=
                SCP_to_MPC !SCP !MPC !h20;
                ( sub_m0_0_MPC[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
                SCP_to_MPC]
                  |||
                  h0_MPC_m0[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
        SCP_to_MPC]
                )
                Endproc
                Process sub_m0_0_MPC[resp, MPC_to_PDE, PDE_to_MPC,
        MPC_to_SCP, SCP_to_MPC]:noexit:=
                  resp !mpc !ReqLoc;
                  MPC_to_PDE !MPC !PDE !h23;
                  PDE_to_MPC !PDE !MPC !h25;
                  resp !mpc !SendLoc;
                  MPC_to_SCP !MPC !SCP !h14;
                  stop
                Endproc
        Endproc
        Process MPC_m1[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
SCP_to_MPC]:noexit:=
         h27_MPC_m1[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
        SCP_to_MPC]
        Where
                Process h27_MPC_m1[resp, MPC_to_PDE, PDE_to_MPC,
        MPC_to_SCP, SCP_to_MPC]:noexit:=
                SCP_to_MPC !SCP !MPC !h78;
                ( sub_m1_0_MPC[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
                SCP_to_MPC]
                  |||
```



```
                  h27_MPC_m1[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
        SCP_to_MPC]
                )
                Endproc
                Process sub_m1_0_MPC[resp, MPC_to_PDE, PDE_to_MPC,
        MPC_to_SCP, SCP_to_MPC]: noexit:=
                  resp !mpc !GetInfo;
                  MPC_to_SCP !MPC !SCP !h77;
                  stop
                Endproc
        Endproc


        Process MPC_m2[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
SCP_to_MPC]: noexit:=
        h111_MPC_m2[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
SCP_to_MPC]
        Where
                Process h111_MPC_m2[resp, MPC_to_PDE, PDE_to_MPC,
        MPC_to_SCP, SCP_to_MPC]: noexit:=
                SCP_to_MPC !SCP !MPC !h107;
                ( sub_m2_0_MPC[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
                SCP_to_MPC]
                  |||
                  h111_MPC_m2[resp, MPC_to_PDE, PDE_to_MPC, MPC_to_SCP,
                SCP_to_MPC]
                )
                Endproc
                Process sub_m2_0_MPC[resp, MPC_to_PDE, PDE_to_MPC,
        MPC_to_SCP, SCP_to_MPC]:noexit:=
                  resp !mpc !SetupMonitoring;
                  MPC_to_PDE !MPC !PDE !h145;
                  PDE_to_MPC !PDE !MPC !h132;
                  resp !mpc !SendLoc;
                  MPC_to_SCP !MPC !SCP !h125;
                  stop
                Endproc
        Endproc
Endproc

Process MS[start, end, MS_to_MSC, MSC_to_MS]:noexit:=
  MS_m0[start,end,MS_to_MSC,MSC_to_MS]
  |||
  MS_m1[start,end,MS_to_MSC,MSC_to_MS]
  |||
  MS_m2[start,end,MS_to_MSC,MSC_to_MS]
Where
        Process MS_m0[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
         h11_MS_m0[start,end,MS_to_MSC,MSC_to_MS]
        Where
                Process h11_MS_m0[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
                start !MS !UpdPos;
                (
                  MS_to_MSC !MS !MSC !h12;
                  stop
```

```
          |||
          h11_MS_m0[start,end,MS_to_MSC,MSC_to_MS]
         )
         Endproc
   Endproc
   Process MS_m1[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
    h27_MS_m1[start,end,MS_to_MSC,MSC_to_MS]
   Where
         Process h27_MS_m1[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
         start !MS !ReqInfo;
         ( MS_to_MSC !MS !MSC !h73;
           sub_m1_0_MS[start,end,MS_to_MSC,MSC_to_MS]
           |||
           h27_MS_m1[start,end,MS_to_MSC,MSC_to_MS]
         )
         Endproc
         Process sub_m1_1_MS[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
           MSC_to_MS !MSC !MS !h29;
           end !ms !ErrorCode;
           stop
         Endproc

         Process sub_m1_0_MS[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
           (
             sub_m1_1_MS[start,end,MS_to_MSC,MSC_to_MS]
             []
             (
               MSC_to_MS !MSC !MS !h41;
               end !ms !CauseCode;
               stop
               []
               (
                 MSC_to_MS !MSC !MS !h42;
                 end !ms !TimeStamp;
                 stop
                 []
                 MSC_to_MS !MSC !MS !h97;
                 end !ms !LocInfo;
                 stop
               )
             )
           )
         Endproc
   Endproc
   Process MS_m2[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
     h111_MS_m2[start,end,MS_to_MSC,MSC_to_MS]
     |||
     h127_MS_m2[start,end,MS_to_MSC,MSC_to_MS]
     |||
     h115_MS_m2[start,end,MS_to_MSC,MSC_to_MS]
     |||
     h100_MS_m2[start,end,MS_to_MSC,MSC_to_MS]
   Where
         Process h111_MS_m2[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
         start !MS !SetInactive;
```



```
         (  MS_to_MSC !MS !MSC !h113;
            stop
            |||
            h111_MS_m2[start,end,MS_to_MSC,MSC_to_MS]
         )
         Endproc
         Process h127_MS_m2[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
         start !MS !SetActive;
         (  MS_to_MSC !MS !MSC !h129;
            stop
            |||
            h127_MS_m2[start,end,MS_to_MSC,MSC_to_MS]
         )
         Endproc
         Process h115_MS_m2[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
         start !MS !PowerOff;
         (  MS_to_MSC !MS !MSC !h130;
            stop
            |||
            h115_MS_m2[start,end,MS_to_MSC,MSC_to_MS]
         )
         Endproc
         Process h100_MS_m2[start,end,MS_to_MSC,MSC_to_MS]:noexit:=
         start !MS !PowerOn;
         (  MS_to_MSC !MS !MSC !h102;
            stop
            |||
            h100_MS_m2[start,end,MS_to_MSC,MSC_to_MS]
         )
         Endproc
   Endproc
Endproc

Process MSC[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]:noexit:=
  MSC_m0[start,resp,end,MSC_to_MS,MS_to_MSC,MSC_to_SCP,SCP_to_MSC]
  |||
  MSC_m1[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP, SCP_to_MSC,
MSC_to_VLR, VLR_to_MSC]
  |||
  MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP, SCP_to_MSC,
MSC_to_VLR, VLR_to_MSC]
Where
   Process MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC]:noexit:=
     h4_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
   SCP_to_MSC]
     |||
     h8_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
   SCP_to_MSC]
     |||
     h11_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
   SCP_to_MSC]
   Where
```



```
         Process h4_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC,
   MSC_to_SCP, SCP_to_MSC]:noexit:=
         start !MSC !CallAssoc;
         (  MSC_to_SCP !MSC !SCP !h6;
            stop
            |||
            h4_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC]
         )
         Endproc
         Process h8_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC,
   MSC_to_SCP, SCP_to_MSC]:noexit:=
         start !MSC !LocationChg;
         (  MSC_to_SCP !MSC !SCP !h10;
            stop
            |||
            h8_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC]
         )
         Endproc
         Process h11_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC,
   MSC_to_SCP, SCP_to_MSC]:noexit:=
         MS_to_MSC !MS !MSC !h12;
         (  MSC_to_SCP !MSC !SCP !h13;
            stop
            |||
            h11_MSC_m0[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC]
         )
         Endproc
   Endproc
   Process MSC_m1[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]:noexit:=
    h27_MSC_m1[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
   Where
         Process h27_MSC_m1[start, resp, end, MSC_to_MS, MS_to_MSC,
   MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]:noexit:=
         MS_to_MSC !MS !MSC !h73;
         ( resp !msc !GetProfile;
           MSC_to_VLR !MSC !VLR !h89;
           sub_m1_0_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
           |||
           h27_MSC_m1[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
         )
         Endproc
         Process sub_m1_1_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
   MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]:noexit:=
           SCP_to_MSC !SCP !MSC !h59;
           MSC_to_MS !MSC !MS !h29;
           stop
         Endproc
```



```
         Process sub_m1_0_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
   MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]:noexit:=
           VLR_to_MSC !VLR !MSC !h90;
           MSC_to_SCP !MSC !SCP !h70;
           (
             sub_m1_1_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
             []
             (
               SCP_to_MSC !SCP !MSC !h81;
               MSC_to_MS !MSC !MS !h41;
               stop
               []
               (
                 SCP_to_MSC !SCP !MSC !h84;
                 MSC_to_MS !MSC !MS !h42;
                 stop
                 []
                 SCP_to_MSC !SCP !MSC !h96;
                 MSC_to_MS !MSC !MS !h97;
                 stop
               )
             )
           )
         Endproc
   Endproc
   Process MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]: noexit:=
     h111_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
     |||
     h127_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
     |||
     h115_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
     |||
     h100_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC, MSC_to_SCP,
SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
   Where
         Process h111_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC,
   MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]: noexit:=
         MS_to_MSC !MS !MSC !h113;
         (  sub_m2_0_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
            |||
            h111_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
         )
         Endproc
         Process h127_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC,
   MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]: noexit:=
         MS_to_MSC !MS !MSC !h129;
         (  sub_m2_0_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
         MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
```

```
        |||
        h127_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC,
      MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
      )
      Endproc
      Process h115_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC,
  MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]: noexit:=
      MS_to_MSC !MS !MSC !h130;
      (   sub_m2_0_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
      MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
        |||
        h115_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC,
      MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
      )
      Endproc
      Process h100_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC,
  MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]: noexit:=
      MS_to_MSC !MS !MSC !h102;
      (   resp !msc !GetProfile;
        MSC_to_VLR !MSC !VLR !h135;
        VLR_to_MSC !VLR !MSC !h133;
        sub_m2_0_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
      MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
        |||
        h100_MSC_m2[start, resp, end, MSC_to_MS, MS_to_MSC,
      MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]
      )
      Endproc
      Process sub_m2_0_MSC[start, resp, end, MSC_to_MS, MS_to_MSC,
  MSC_to_SCP, SCP_to_MSC, MSC_to_VLR, VLR_to_MSC]:noexit:=
       resp !msc !NotifyFAM;
       MSC_to_SCP !MSC !SCP !h114;
       stop
      Endproc
  Endproc
Endproc

Process PDE[resp,PDE_to_MPC,MPC_to_PDE]:noexit:=
 PDE_m0[resp,PDE_to_MPC,MPC_to_PDE]
 |||
 PDE_m2[resp,PDE_to_MPC,MPC_to_PDE]
Where
  Process PDE_m0[resp,PDE_to_MPC,MPC_to_PDE]:noexit:=
   h0_PDE_m0[resp,PDE_to_MPC,MPC_to_PDE]
  Where
      Process h0_PDE_m0[resp,PDE_to_MPC,MPC_to_PDE]:noexit:=
      MPC_to_PDE !MPC !PDE !h23;
      ( sub_m0_0_PDE[resp,PDE_to_MPC,MPC_to_PDE]
        |||
        h0_PDE_m0[resp,PDE_to_MPC,MPC_to_PDE]
      )
      Endproc
      Process sub_m0_0_PDE[resp,PDE_to_MPC,MPC_to_PDE]:noexit:=
       resp !pde !CalcLoc;
       PDE_to_MPC !PDE !MPC !h25;
```



```
       stop
      Endproc
  Endproc
  Process PDE_m2[resp,PDE_to_MPC,MPC_to_PDE]:noexit:=
   h111_PDE_m2[resp,PDE_to_MPC,MPC_to_PDE]
  Where
      Process h111_PDE_m2[resp,PDE_to_MPC,MPC_to_PDE]: noexit:=
      MPC_to_PDE !MPC !PDE !h145;
      ( sub_m2_0_PDE[resp,PDE_to_MPC,MPC_to_PDE]
        |||
        h111_PDE_m2[resp,PDE_to_MPC,MPC_to_PDE]
      )
      Endproc
      Process sub_m2_0_PDE[resp,PDE_to_MPC,MPC_to_PDE]:noexit:=
       resp !pde !CalcLoc;
       PDE_to_MPC !PDE !MPC !h132;
       stop
      Endproc
  Endproc
Endproc

Process SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
  SCP_m0[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC, MSC_to_SCP,
 SCP_to_SV, SV_to_SCP]
  |||
  SCP_m1[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC, MSC_to_SCP,
 SCP_to_SV, SV_to_SCP]
  |||
  SCP_m2[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC, MSC_to_SCP,
 SCP_to_SV, SV_to_SCP]
Where
  Process SCP_m0[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
 MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
   h0_SCP_m0[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
  MSC_to_SCP, SCP_to_SV, SV_to_SCP]
   |||
   h4_SCP_m0[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
  MSC_to_SCP, SCP_to_SV, SV_to_SCP]
   |||
   h8_SCP_m0[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
 MSC_to_SCP, SCP_to_SV, SV_to_SCP]
   |||
   h11_SCP_m0[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
 MSC_to_SCP, SCP_to_SV, SV_to_SCP]
  Where
      Process h0_SCP_m0 [start, resp, end, SCP_to_MPC, MPC_to_SCP,
  SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
      start !SCP !Scheduled;
      (   sub_m0_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
        |||
        h0_SCP_m0 [start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
      )
```



```
      Endproc
      Process h4_SCP_m0 [start, resp, end, SCP_to_MPC, MPC_to_SCP,
  SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
      MSC_to_SCP !MSC !SCP !h6;
      (   sub_m0_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
        |||
        h4_SCP_m0 [start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
      )
      Endproc
      Process h8_SCP_m0 [start, resp, end, SCP_to_MPC, MPC_to_SCP,
  SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
      MSC_to_SCP !MSC !SCP !h10;
      (   sub_m0_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
        |||
        h8_SCP_m0 [start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
      )
      Endproc
      Process h11_SCP_m0 [start, resp, end, SCP_to_MPC, MPC_to_SCP,
  SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
      MSC_to_SCP !MSC !SCP !h13;
      (   sub_m0_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
        |||
        h11_SCP_m0[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
      )
      Endproc
      Process sub_m0_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
  SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]: noexit:=
       resp !scp !QueryLoc;
       SCP_to_MPC !SCP !MPC !h20;
       MPC_to_SCP !MPC !SCP !h14;
       resp !scp !UpdLoc;
       resp !scp !NotifySV;
       SCP_to_SV !SCP !SV !h17;
       stop
      Endproc
  Endproc
  Process SCP_m1[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
   h27_SCP_m1[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
MSC_to_SCP, SCP_to_SV, SV_to_SCP]
  Where
      Process h27_SCP_m1[start, resp, end, SCP_to_MPC, MPC_to_SCP,
  SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
      MSC_to_SCP !MSC !SCP !h70;
      ( sub_m1_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
       |||
       h27_SCP_m1[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
```



```
      )
      Endproc
      Process sub_m1_1_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
  SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]: noexit:=
       resp !scp !SendErrorCode;
       SCP_to_MSC !SCP !MSC !h59;
       stop
      Endproc
      Process sub_m1_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
  SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]: noexit:=
       resp !scp !ChkAuth;
       (
        [refused]->
        sub_m1_1_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
        []
        [Granted]->
        resp !scp !ChkAct;
        (
         [Active]->
         resp !scp !ChkAvail;
         SCP_to_MPC !SCP !MPC !h78;
         MPC_to_SCP !MPC !SCP !h77;
         (
          [NonAvail]->
          resp !scp !BestLoc;
          resp !scp !RecInvocDet;
          resp !scp !SendCauseCode;
          SCP_to_MSC !SCP !MSC !h81;
          stop
          []
          [Avail]->
          resp !scp !ChkCurrent;
          (
           [NonCurrent]->
           resp !scp !RecInvocDet;
           resp !scp !SendTimeStamp;
           SCP_to_MSC !SCP !MSC !h84;
           stop
           []
           [Current]->
           resp !scp !DelivInfo;
           resp !scp !RecInvocDet;
           SCP_to_MSC !SCP !MSC !h96;
           stop
          )
         )
         []
         [NonAct]->
         sub_m1_1_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
      SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
        )
       )
      Endproc
  Endproc
```

```
        Process SCP_m2[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
         h111_SCP_m2[start, resp, end, SCP_to_MPC, MPC_to_SCP, SCP_to_MSC,
MSC_to_SCP, SCP_to_SV, SV_to_SCP]
        Where
                Process h111_SCP_m2[start, resp, end, SCP_to_MPC, MPC_to_SCP,
         SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
                MSC_to_SCP !MSC !SCP !h114;
                ( sub_m2_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
                SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
                 |||
                 h111_SCP_m2[start, resp, end, SCP_to_MPC, MPC_to_SCP,
                SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
                )
                Endproc
                Process sub_m2_0_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
        SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
                 resp !scp !SetStatus;
                 (
                  [Pull]->
                  sub_m2_1_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
                SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
                  []
                  [Push]->
                  SCP_to_MPC !SCP !MPC !h107;
                  MPC_to_SCP !MPC !SCP !h125;
                  resp !scp !UpdLoc;
                  sub_m2_1_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
                SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]
                 )
                Endproc
                Process sub_m2_1_SCP[start, resp, end, SCP_to_MPC, MPC_to_SCP,
        SCP_to_MSC, MSC_to_SCP, SCP_to_SV, SV_to_SCP]:noexit:=
                 resp !scp !NotifySV;
                 SCP_to_SV !SCP !SV !h101;
                 stop
                Endproc
        Endproc
Endproc

Process SV[end,SV_to_SCP,SCP_to_SV]:noexit:=
 SV_m0[end,SV_to_SCP,SCP_to_SV]
 |||
 SV_m2[end,SV_to_SCP,SCP_to_SV]
Where
        Process SV_m0[end,SV_to_SCP,SCP_to_SV]:noexit:=
         h0_SV_m0[end,SV_to_SCP,SCP_to_SV]
        Where
                Process h0_SV_m0[end,SV_to_SCP,SCP_to_SV]:noexit:=
                SCP_to_SV !SCP !SV !h17;
                ( sub_m0_0_SV[end,SV_to_SCP,SCP_to_SV]
                 |||
                 h0_SV_m0[end,SV_to_SCP,SCP_to_SV]
                )
                Endproc
```



```
                Process sub_m0_0_SV[end,SV_to_SCP,SCP_to_SV]:noexit:=
                 end !sv !SuccOutCome;
                 stop
                Endproc
        Endproc
        Process SV_m2[end,SV_to_SCP,SCP_to_SV]:noexit:=
         h111_SV_m2[end,SV_to_SCP,SCP_to_SV]
        Where
                Process h111_SV_m2[end,SV_to_SCP,SCP_to_SV]:noexit:=
                SCP_to_SV !SCP !SV !h101;
                ( sub_m2_0_SV[end,SV_to_SCP,SCP_to_SV]
                 |||
                 h111_SV_m2[end,SV_to_SCP,SCP_to_SV]
                )
                Endproc
                Process sub_m2_0_SV[end,SV_to_SCP,SCP_to_SV]:noexit:=
                 sub_m2_1_SV[end,SV_to_SCP,SCP_to_SV]
                Endproc
                Process sub_m2_1_SV[end,SV_to_SCP,SCP_to_SV]:noexit:=
                 end !sv !SuccOutCome;
                 stop
                Endproc
        Endproc
Endproc

Process VLR[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC, MSC_to_VLR]:noexit:=
 VLR_m1[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC, MSC_to_VLR]
 |||
 VLR_m2[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC, MSC_to_VLR]
Where
        Process VLR_m1[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC,
MSC_to_VLR]: noexit:=
         h27_VLR_m1[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC, MSC_to_VLR]
        Where
                Process h27_VLR_m1[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC,
        MSC_to_VLR]: noexit:=
                MSC_to_VLR !MSC !VLR !h89;
                ( (
                  [no_vlr_profile]->
                  VLR_to_HLR !VLR !HLR !h87;
                  HLR_to_VLR !HLR !VLR !h91;
                  sub_m1_0_VLR[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC,
                MSC_to_VLR]
                  []
                  [has_vlr_Profile]->
                  resp !vlr !v_Profile;
                  sub_m1_0_VLR[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC,
                MSC_to_VLR]
                 )
                 |||
                 h27_VLR_m1[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC,
                MSC_to_VLR]
                )
                Endproc
```



```
                Process sub_m1_0_VLR[resp, VLR_to_HLR, HLR_to_VLR,
        VLR_to_MSC, MSC_to_VLR]:noexit:=
                 VLR_to_MSC !VLR !MSC !h90;
                 stop
                Endproc
        Endproc
        Process VLR_m2[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC,
MSC_to_VLR]:noexit:=
         h100_VLR_m2[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC,
        MSC_to_VLR]
        Where
                Process h100_VLR_m2[resp, VLR_to_HLR, HLR_to_VLR,
        VLR_to_MSC, MSC_to_VLR]:noexit:=
                MSC_to_VLR !MSC !VLR !h135;
                (  (
                   [hasPro]->
                   resp !vlr !v_Profile;
                   VLR_to_MSC !VLR !MSC !h133 ;
                   stop
                   []
                   [noPro]->
                   VLR_to_HLR !VLR !HLR !h140;
                   HLR_to_VLR !HLR !VLR !h136;
                   VLR_to_MSC !VLR !MSC !h133 ;
                   stop
                  )
                 |||
                 h100_VLR_m2[resp, VLR_to_HLR, HLR_to_VLR, VLR_to_MSC,
                MSC_to_VLR]
                )
                Endproc
        Endproc
(**************************************Scenarios********************************************************)
        Process scenario1[start, resp, end, plugin, ms_to_msc, msc_to_vlr, vlr_to_hlr,
                hlr_to_vlr, vlr_to_msc, msc_to_scp, mpc_to_scp, scp_to_mpc, mpc_to_pde,
                pde_to_mpc, scp_to_sv, scenario1]: noexit:=
                start !ms !poweron;
                ms_to_msc  ! ms ! msc ! h102;
                resp !msc !getprofile;
                msc_to_vlr  ! msc ! vlr ! h135;
                vlr_to_hlr  ! vlr ! hlr ! h140;
                resp ! hlr ! h_profile;
                hlr_to_vlr  ! hlr ! vlr ! h136;
                vlr_to_msc  ! vlr ! msc ! h133;
                resp ! msc !notifyfam;
                msc_to_scp  ! msc ! scp ! h114;
                resp  ! scp ! setstatus;
                scp_to_mpc  ! scp ! mpc ! h107;
                resp ! mpc !setupmonitoring;
                mpc_to_pde ! mpc ! pde ! h145;
                resp  ! pde ! calcloc;
                pde_to_mpc  ! pde ! mpc ! h132;
                resp  ! mpc !sendloc;
                mpc_to_scp ! mpc ! scp ! h125;
```



```
                resp !scp !updloc;
                resp !scp !notifysv;
                scp_to_sv  ! scp ! sv ! h101;
                end !sv !succoutcome;
                scenario1;
                stop
        Endproc

        Process scenario2[start, resp, end, plugin, ms_to_msc, msc_to_vlr, vlr_to_hlr, hlr_to_vlr,
vlr_to_msc, msc_to_scp, mpc_to_scp, scp_to_mpc, mpc_to_pde, pde_to_mpc, scp_to_sv,
scenario2]: noexit:=
                start !scp !Scheduled;
                resp !scp !QueryLoc;
                scp_to_mpc !scp !mpc !h20;
                resp !mpc !ReqLoc;
                mpc_to_pde !mpc !pde !h23;
                resp !pde !CalcLoc;
                pde_to_mpc !pde !mpc !h25;
                resp !mpc !SendLoc;
                mpc_to_scp !mpc !scp !h14;
                resp !scp !UpdLoc;
                resp !scp !NotifySV;
                scp_to_sv !scp !sv !h17;
                end !sv !SuccOutCome;
                scenario2;
                stop
        Endproc
Endspec
```