Modeling and Derivation of Scenarios for a Mobile Telephony System in LOTOS

Randall Tuok

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

Master of Computer Science

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

University of Ottawa
Ottawa, Ontario, Canada
January 30, 1996

Acknowledgments

I would like to express my deep appreciation to my supervisor, Professor Luigi Logrippo, for his valuable time, patience and advice throughout my graduate studies. His accuracy in reviewing the drafts of this thesis has greatly improved its contents and presentation. I am also much indebted to the LOTOS Research Group for providing a pleasant academic environment to carry out my work. A special thanks goes to Amine Rachdi for the many fruitful discussions we had. The help and technical support of Jacques Sincennes and Daniel Amyot are greatly appreciated. A special thank-you is extended to Craig Leger of Nortel for his useful comments. I would also like to express my appreciation to my family for their support and encouragement. Last but not least, I would like to express my heartfelt thank to Lynda McGahan for her love, friendship and her belief in me.

This work was made possible by the financial support of the Telecommunications Research Institute of Ontario, Nortel, and the University Research Incentive Fund of Ontario.

Abstract

Use cases are widely recognized as useful tools for describing user requirements. Because they are represented informally, use cases permit user participation in and are conducive to the creativity needed for requirements elicitation and analysis. However, the representation of a system by a collection of use cases has disadvantages. For instance, because system description is spread over many small models, changing one use case requires to visit all use cases to determine whether the change affects them. This problem is exacerbated by the difficulty of consistency check between use cases. Furthermore, such a model does not show the starting and terminating contexts of the use cases, nor is it easy denote the relationships between them.

LOTOS is a mathematically based specification language developed within the framework of ISO standardization for the specification of OSI services and protocols. This thesis presents a method for transforming system descriptions in the form of *scenarios* (defined as collections of causally-ordered internal and external events) into 'unified' LOTOS specifications. This transformation eliminates some of the undesirable properties associated with describing a system with scenarios, and bridges the gap between early informal requirements and formal LOTOS-based development processes. Moreover, it describes a system's structure in the same model as its behaviors. The crux of this method is the integration of scenarios. Our proposed method performs integration at the semi-formal level and uses LOTOS operators for the integration.

The method was derived by a posteriori rationalization of the process followed to specify the European standard for the Global System for Mobile Communications (GSM). Thus, GSM itself and its specification in LOTOS are discussed. The thesis also reports results for automatic regeneration of scenarios from LOTOS specifications as well as a technique for determining whether or not a specification contains a given scenario.

Table of Contents

List of Figures

Glossary

BSC	Base station controller
BSS	Base station subsystem
BTS	Base transceiver station
CM	Communications Management
HLR	Home location register
ISDN	Integrated service digital network
IMSI	International mobile subscriber identity
ISUP	ISDN User Part
MAP	Mobile Applicaiton Part
ME	Mobile Equipment
MM	Mobility Management
MS	Mobile station or mobile subscriber
MSC	Mobile switcingcenter
MSISDN	MS ISDN number
NSS	Network switching subsystem
OSS	Operation Sub-System
PCH	Paging channel
PLMN	Public land mobile network
PSTN	Public switched telephone network
RACH	Random access channel
RIL3-CC	Radio Interface Layer 3 - Call Control
RIL3-MM	Radio Interface Layer 3 - Mobility Management
RIL3-RR	Radio Interface Layer 3 - Radio Resource Management
RR	Radio Resource Management
SIM	Subscriber identity modul
SS7	(CCITT's) System signalling No. 7
TMSI	Temporary mobile subscriber identity
TUP	Telephone User Part
VLR	Visitor location register

to Lynda

CHAPTER 1 Introduction and Problem Definition

1.1 Background and Motivation

As software systems continue to increase in size and complexity, Formal Description Techniques (FDTs) and their related methodologies are becoming recognized as tools that could aid in the construction of quality software. FDTs are a class of mathematically based notations (language and/or graphical) that enable to describe systems in consistent and unambiguous manners as well as to reason about them in formal ways. Some of the more popular languages in this class are: VDM, SDL, Estelle, Z and LOTOS.

The FDT LOTOS, *Language Of Temporal Ordering Specifications* [ISO89], was developed within the framework of ISO standardization for the specification of OSI services and protocols. The language is based on process algebras and abstract data types and enjoys well defined syntax and semantic. The basic idea of specification in this FDT is to describe systems by defining the temporal relations among the events that constitute their externally observable behaviors. Chapter 2 discusses LOTOS principles in detail.

LOTOS does not enjoy as widespread a use in industry as older FDTs such as SDL. The lack of penetration of LOTOS in the software development arena can be attributed to at least two main reasons [PKT92]: 1) lack of sufficiently powerful support tools, and 2) lack of the rich variety of techniques and methods that other, more widely-used although perhaps less formal, notations have. We hypothesize that an increase in the availability of robust support tools, simple techniques for specifying in this language, and powerful manipulation techniques are some of the ways to gain increased use of this FDT.

The utilization of LOTOS is probably most beneficial at the early stages of software development when descriptions of a system usually exist in fragments and are often represented in informal notations such as natural language. At this stage, the user requirements that define the behaviors and properties of the system to be build are elicited in bits and pieces, often one functionality at a time. These requirements are represented in informal notation because they are conducive to creativity and because they are generated with the participation of users, who are often unfamiliar with formal notation. An accepted technique of generating requirements is to imagine the interactions between the system and its users as they use the system and capture the externally observable behaviors as *use cases* [Jac92]. The problem with modeling a system by a collection of use cases is that they are prone to inconsistencies, ambiguities and incompleteness. We believe that LOTOS can be used at this stage to create *unified models* from use cases as well as to validate the requirements. By ‘unified models’, we mean representations in which relationships between different behaviors are shown, i.e., it is possible to tell how a system progresses from one behavior to another. A more rigorous definition of *model* is given in Section 3.3.

LOTOS can also be used at later stages of the software life cycle. According to [Vig91], sequences of internal and external system events are often generated manually during design as part of design walk-through to quickly verify the design’s capability to provide the required service. (Note that these sequences are similar to use cases except that they contain internal events.) There exists a rich body of theories and techniques for validating system designs by validating their LOTOS specifications. For instance, there are techniques and tools that check LOTOS specifications for properties such as the absence of deadlocks, and those that verify equivalence (according to some equivalence relation) between two specifications [QPF88] [Hus88] [Ash92] [Eer94]. Section 3.3.3 contains a brief overview of testing in LOTOS.

These important applications make obvious the need for methods for creating and validating LOTOS specifications from use cases and from system designs, which is the problem addressed by this thesis. We will deal only with high-level designs, i.e., those that are derivable from black-box systems by applying a small number of decompositions. It is possible to specify low-level designs in LOTOS but such specifications are likely to be large and difficult to create and too complex for existing tools to handle. More specifically, we choose a view that focuses on the topmost system components and their interactions. For example, if the system under consideration is a communication network, the components we deal with are the network nodes such as switches, gateways, databases, etc., and the system's behaviors are the interactions between these components; if the system is an operating system, its components are the subsystems such as file manager, scheduler, etc. Systems viewed at this abstraction level are called "reactive systems" in [Vig91] and "interaction systems" in [VSA89]. They correspond to a class of system more commonly known as *distributed systems*.

At this abstraction level, three kinds of system properties are relevant: 1) functional properties, 2) architecture (or structure), and 3) non-functional properties (sometimes called constraints). Non-functional properties pertains to aspects of performance (throughput, speed, reliability, etc.) and appearance (colour, size, etc.). Since LOTOS was not designed to denote non-functional system characteristics in an elegant way, this thesis does not deal with this kind of system properties.

1.2 Behavior and Architecture Concepts for Distributed Systems

Unlike the specification of black-box systems which requires consideration of only external properties, the specification of distributed systems must also take into account their internal (top-level) behaviors and architectures. The architecture of a system is its structural properties, and its behavior is its operational characteristics. An architecture identifies subsystem units, called *components* (Section 1.2.1), from which the system is built, and defines how these components are connected. The connections between components determine their static relationships. The ways the components interact with each other form their dynamic relationships, and constitute behavior.

In this thesis, internal system behaviors are described by *use case* or *scenarios* (Section 1.2.2), and system components and system users are called *objects*. The term objects, as used here, does not refer to objects as they are defined in OO programming but to object-like entities.

1.2.1 System Components

This thesis defines components as *kind*s of subsystem entities (as opposed to specific entities) that possess specific functionalities and clear interfaces. Specific instances of components are encapsulated architectural units that interact with one another to provide system services. They operate independently of and concurrently with other instances, except possibly at specific points of synchronization. At the abstraction level considered in this thesis, components are the subsystems that result from the first few decompositions of the system. They may be functional or physical units that result from partitions of the system along functional or physical lines.

In our view, two kinds of relations exist that can be used to relate components to one another: static and dynamic relations. The static relations include the relations *contains*, *connects-to*, and *connects*. The *contains* relation is used to relate embedded or nested components to the ones that contain them. The other two relations are used to relate components that communicate with one another. If the communication link between two object is unidirectional, then the *connects-to* relation is used; an object that is connected to another object can send message to that object, but not vice versa. If the communication link is bidirectional, then the *connects* relation is applicable. Two objects which are related by this relation can communicate with one another.

A set of components and their static relationships constitute an architecture. The connection topology of the components identifies communication channels that exist between them. A connection may be dynamic. This represents situations such as when new nodes are added to a network or when some nodes fail and operationally cease to be part of the network, or on a shorter time scale, when some of the nodes are mobile.

1.2.2 Behavioral Concepts

Use cases have been widely accepted as useful tools for describing system requirements because they provide manageability, are conducive to creativity, and facilitate human understanding [Jac92] [RKW95][Car95]. Some authors use the term *scenario* for this same concept [HSG94][Car95]. Buhr provides a visual notation for use cases, called *Use Case Maps* [BuC95], that is also a means of transforming them into high-level designs.

The basic definition of use case/scenario, provided by Jacobson, is that it is a sequence of events that illustrates the behavior of a system when a service is invoked. However, there exist in the literature many variants of this basic definition. The definitions differ on:

- the number of users that may be involved,
- whether or not a use case can contain branches,
- whether a use case should describe a behavior pattern that is applicable to classes of objects or a specific behavior of particular instances of objects,
- the criteria upon which the events are ordered (i.e., by causal or temporal criteria) and
- the types of events (i.e., internal or external) that could be in a use case.

It is generally accepted that a use case may involve any number users. For example, Jacobson suggests that a use case should normally describe the interaction between a system and only one of its users, but it is acceptable to involve more than one user if more complete and extensive use cases are desired [Jac92]. On the other hand, Regnell restricts the number of users involved in a use case to exactly one [RKW95].

[Jac92] states that a use case may contain branches that denote variants of the basic course or error-handling procedures. None of [HSG94], [RKW95] or [Car95] say anything about the 'shape' of use cases.

According to Jacobson, use cases should describe behaviors of classes of similarly behaved objects rather than of specific objects. Descriptions of the first kind are said to be *abstract* and of the latter to be *concrete*. To use OO terminology, abstract use cases are class concepts and concrete use cases are instance concepts. Instances of use cases (or concrete use cases) are derived from abstract ones by giving specific identities to the objects involved in the interactions and supplying specific values to any data involved the events in the use cases. Another way to think about abstract use cases is to consider them as programs [Jac92]. When a use case is initiated, a use case instance executes and the control of execution flows over the objects involved in the use case according to a path defined by the use case. After execution has reached the last event, the use case instance disappears. A use case instance exists as long as it is being executed. The specific values that must be supplied to a use case to instantiate it can be thought of as arguments that are given to a program to bind its parameters to real values.

This thesis uses the term use case/scenario for abstract use case/scenario and concrete use case/ scenario will sometimes be called use case/scenario instances.

The criteria for ordering events in a use case is very important in determining which sequences of events are use cases and which are not. Many definitions of use case, for instance those of [Jac92], [HSG94] and [Car95], do not specify explicitly the ordering criteria. There are at least two ways to order sequences of events. In a *causally-ordered* use case, every event (except the first one) is caused by the preceding events in the sequence [BuC95]. A use case that satisfies this definition contains only events that occur when a user invokes a system function. *Temporally-ordered* use cases may contain events from unrelated parallel behaviors. The latter is a more relaxed notion, and use cases by this definition may not tell very much of how a user interacts with the system.

The kind of events that should be included in a use case is an important factor in the definition of this concept. Use cases that contain only external events express requirements and those that include also internal events express design solutions. Because the term scenario is generally used interchangeably with use case, “requirements scenario” and “design scenario” are also valid and they have the same meaning as their use case counterparts. However, in this thesis we shall use the term use case to refer to requirements use case and the term scenario to refer to design use case.

Use Cases

Although the terms use case and scenario are used interchangeably in the literature, this thesis will distinguish between the two concepts. Henceforth we shall take use cases to mean “requirements use cases” and scenarios to mean “design use cases” or “solution use cases”. We note that these definitions are conventions used only in this thesis. In the literature, use case and scenario are used synonymously. The different between use case and scenario is illustrated by Figure 1.

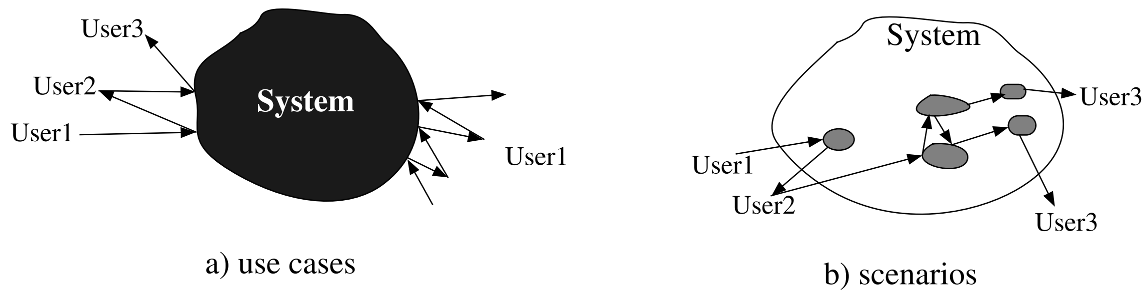


Figure 1 The difference between use case and scenarios.

We define a use case as follows. A use case is a (possibly branching) flow of externally observable events that occur when *one* system service is invoked. It describes the behaviors of a system and a group of similarly-behaved users. An example of a use case for a cellular telephone system is shown in Figure 2. Note that it does not refer to any particular user, telephone or telephone number.

```
user performs OFFHOOK
-> handset gives dial-tone
-> user dials number
-> user pushes SEND button
-> if the dialed number is busy,     -> if dialed number is not busy,
   give busy tone                       connect the two users
-> user performs ONHOOK
```

Figure 2 Use case for making a telephone call on a cellular telephone.

Scenarios

We define scenario to be a course of externally observable events that could occur between a system and its users *and,* possibly, internal events that could take place between system components. In this sense scenario is a more general concept than use case; a use case is a scenario but the reverse might not be true. A use case describes only the observable events that occur when a user invokes a system service [Jac92], but a scenario might also describe the interactions between system components in providing that service. In other words, the scenario concept is applicable at the requirements level as well as the design level where it can be used to illustrate the internal mechanisms that realize system services. It does not specify components independently of each other but rather it specifies behaviors involving a set of objects. Figure 1 illustrates the difference between use cases and scenarios. In the figure, *User1*, *User2* and *User3* may refer to the same user. Note that the main distinction between the two notions is in their levels of abstraction.

Definition 1.1

An *event* is an atomic unit of behavior. It is an element of an object's capabilities. The following information is associated with an event:

- a name,
- an interface on which it may occur,
- a set of objects that participate in the event,
- data that are exchanged (if any) during the occurrence of the event and,
- a set of conditions (possibly empty) that determine whether or not the event can occur.

On a more conceptual level, an event is an abstraction that conceals the detailed protocol of the interactions inside an object to produce an observable manifestation (which is that event). For example, an event in GSM may be "sending a message", although to actually send a message it may have to perform many internal events such as composing the message, calculating the recipient's address, and transmitting the message. An event description does not contain reference to any specific object, interface or data value but to objects, kinds of interfaces and data types *viz*. event is a class concept, and its occurrence is an instant concept. An event occurs (or is instantiated) when specific object identities, interface identities and data values are supplied. The occurrence of an event is atomic and instantaneous. The term *action* is synonymous with event.

Generally, an interface may be described at several levels of details. For example, the communication between an MS and a BTS may be described as occurring on the radio interface, on an interface between the specific pair of MS and BTS or, at an even more detailed level, on a specific radio channel. In our specification of GSM, gate splitting is used to split the LOTOS gate representing the first kind of interface into representations for the latter two kinds (see Section “Event Structures” on page 94).

Definition 1.2

A *scenario* is a causally-ordered finite collection of events. It has the following information:

- a name,
- a collection of events, and
- the ordering of these events.

The concept of scenario, as defined above, is similar to *use case paths* [BuC95] and *timethreads* [Amy94]. Its definition as “causally-ordered” follows Buhr’s definition of use case paths. In [BuC95], causally-ordered means that all events preceding a particular event in a scenario, including the stimuli that triggered the very first event of that scenario, contribute to the cause that triggers that event. The stimuli that trigger the first event of a scenario may come from outside of the system or may be “spontaneously generated” by a system component [BuC95]. The last event in a scenario is the ultimate effect of the scenario.

In this thesis, the concept of causality is further refined to take into account definition of scenarios as class concepts. The cause-effect relationship at one point in a scenario (say, between events X and Y) is the property of the *particular* flow leading up to that point. That is, two consecutive events in a scenario do not always occur together (i.e., the occurrence of X is not always followed by the occurrence of Y), and different instances of a scenario may follow different paths. (This implies that scenarios have branches.) To use OO terminology, the cause-effect relation is an instance relationship and not a class relation. This definition makes sense of scenarios that have alternative sub-paths; different sub-paths may be exercised in different instances of the scenario. To see this, consider the make-telephone-call scenario in Figure 2. Suppose that two instances of this scenario differ in the telephone number that is dialed (the third event). In one instance, the call is made to a number that is busy and the busy tone is sounded. And in the other instance, the call is made to an emergency number (where a busy signal is never given) and the caller is connected to the emergency response unit. This example shows that: 1) the occurrence of an event does not depend entirely on the occurrence of its *immediate* predecessor, and 2) different instances of a path may produce different effects.

Note that scenarios are not defined as *sequences* of events because they may contain alternative paths of events; the term sequence implies linearity. This definition allows scenarios to contain more complete behavior descriptions than if they were defined as causally-ordered *sequences* of events. Normally, relationships between scenarios are determined after the scenarios have been identified and described. However, in some cases it is possible to identify them earlier. In such cases, it is desirable to keep these relationships and propagate them in the analysis and specification process rather than remove them and try to rediscover them later. Furthermore, it is not always obvious what functionality should be described separately by different scenarios. The given definition avoids having to separate closely related behaviors. For instance, variants of the basic course and error-handling procedures give alternative sub-paths of the basic course of a scenario; and describing these variant behaviors independent of the basic course might be an impediment to understanding them.

On the other hand, a scenario that is too 'big' can also be difficult to understand and unwieldy to handle. In general, a scenario should describe only one basic course of behavior, i.e., it should contain only the events that occur when *one* system service is invoked.

Figure 3 illustrates a (abstract) call scenario for the use case in Figure 2. Note that the description involves internal events and mechanisms, and that it refers to another scenario. Note also that this informal description is ambiguous (e.g., in the second event, "`MS requests channels from BSC with random number`", it is unclear whether the random number is associated with the request, the channel, or the BSC). The degree of abstraction is not uniform throughout the length of the scenario, for some events the actual messages that are sent are specified, for others only general descriptions are given, e.g., "`MSC establishes a connection to the called terminal and alerts it`". This example shows the need to formalize scenarios before attempting to integrate them, or before attempting to do anything with them.

```
user performs OFFHOOK, dials the number to be called, presses SEND button
-> MS requests channels from BSC with random number
-> BSC assigns a signalling channel and a data channel
-> MS sends CM Service Request to MSC
-> MSC sends CM Service Accept to MS
-> MS requests Setup
-> MSC checks on status of called number
   -> If called number is free,          -> If the called number is busy,
      MSC establishes a connection to       see Call-Busy scenario
      to the called terminal and
      alerts it
   -> MSC sends Alert to the MS
```

Figure 3 The mobile-originated call scenario; an example.

1.2.3 Representing System Requirements with Scenarios

Scenarios are useful thinking tools as they partition a system's functionality into small manageable pieces that could be handled separately. "It is well known that scenarios are spontaneously and pervasively used in system development as currently practiced" [Car95].

In addition, if scenarios are represented informally, such as in natural language [Jac92] or graphically [BuC95][HSG94] [RKW95], they facilitate communication with users, who often do not understand formal notations. This combination of concepts and notation is especially useful in the requirements phase of the software life-cycle where usually the behaviors of a system have to be defined with user inputs. According to [Car95], scenarios have been shown to "provide an effective means for users and designers to communicate about system requirements and design options."

In general, scenarios have the following characteristics:

- Context: A scenario has starting and terminating contexts which are the system states in which it may be initiated or terminated. The contexts of different scenarios are likely to be different. Thus, using a collection of scenarios to model a system does not give a 'unified' picture of the system's behavior.
- Intersection: Scenarios that describe the same system may overlap fully or partially.

- Granularity: When scenarios are derived, the length and abstraction level of each scenario is a matter of arbitrary choice. This may lead to underspecification or overspecification of the system they describe.
- Interaction: Scenarios may interact in many ways. For example, the occurrence of a particular scenario may exclude the possibility of another scenario from occurring, or the occurrence of one may guarantee the occurrence of another, etc.
- Concurrency: Different scenarios may be able to occur concurrently or only sequentially.

These properties imply that system models that consist of collections of scenarios suffer the following problems:

- Since the scenarios are likely to have different starting and terminating contexts, it is difficult to perceive a unified picture of the system's behavior.
- Since scenarios may overlap, a particular behavior may be described multiple times in different scenarios.
- Such a model is difficult to change. When a change is required, many scenarios in the collection would need to be checked to determine if the change affect them. Furthermore, no criteria exists to determine those that have to be checked. Over time or with frequent changes, scenarios may become mutually inconsistent.
- Because of the fact the use cases are separate models, it is more difficult to perceive the completeness of the model.
- The same behavior may be described by different terminologies, at different levels of formality, etc. in different scenarios. This problem can be especially serious if the model were derived by different people.

These problems are exacerbated when dealing with large and complex systems because their descriptions often consist of large numbers of scenarios. Some of these problems may be alleviated by consolidating and integrating the scenarios into unified models. By merging scenarios it is possible to show relationships such as context and intersection that exist between them as well as facilitating validation.

1.3 Objective

The objective of this thesis is to investigate how collections of scenarios can be integrated to produce unified models represented by LOTOS specifications. The investigation focuses on systems at the high design level. Since modeling systems at these abstraction levels requires consideration of internal properties such as system components and their interaction, precise definitions are given for these concepts.

A technique for transforming informal models into LOTOS specifications will be presented. The technique attempts to bring some formality to the specification process by breaking it down into activities and formalizing them where possible. It takes as inputs informal descriptions and produces as output resource-oriented LOTOS specifications. The technique represents a contribution towards a methodology for the specification and validation of system designs. The novelty of this technique is the semi-formal approach for integrating scenarios to produce a unified model that also captures the structure of a system. The process is based on Jacobson's Object-Oriented Software Engineering methodology (OOSE) and on experience the author has gained from specifying small digital circuits, an operating system, and the GSM mobile telecommunication system. The technique will be illustrated with an application on an example of real size, the GSM system.

The thesis will also report results of validating LOTOS specifications by automatically generating scenarios from them and by using the specification to 'recognize' scenarios. These two applications also provide the means for improved understanding of the specification behaviors.

1.4 Approach and Related Work

The crux of the problem of deriving a unified model from a collection of scenarios is the integration of the scenarios. This integration may be carried out at the informal level when the scenarios are still informally represented, at the formal level after the scenarios have been translated to LOTOS specifications, or at a point between these two extremes. A semi-formal technique that merges scenarios is reviewed in Subsection 1.4.2. Only limited success has been reported for integration of LOTOS specifications, it is reviewed in Subsection 1.4.3. This thesis proposes integration at the semi-formal level.

The front-end process that generates scenarios from informal information sources for input into our process is the same one used in OOSE. It is reviewed in Subsection 1.4.1.

1.4.1 Scenario Driven Analysis

Jacobson's Object-Oriented Software Engineering (OOSE) methodology starts system development with the gathering and analysis of scenarios. Note that what Jacobson called use case we call scenario in this thesis. We use the term Scenario-Driven Analysis (SDA) to refer to that part of OOSE that is applicable at the requirements stage to distinguish it from the whole OOSE. SDA takes as input informal requirement descriptions and produces a number of models, each capturing different aspects of the system. The SDA process and the models it produces are illustrated by Figure 4.

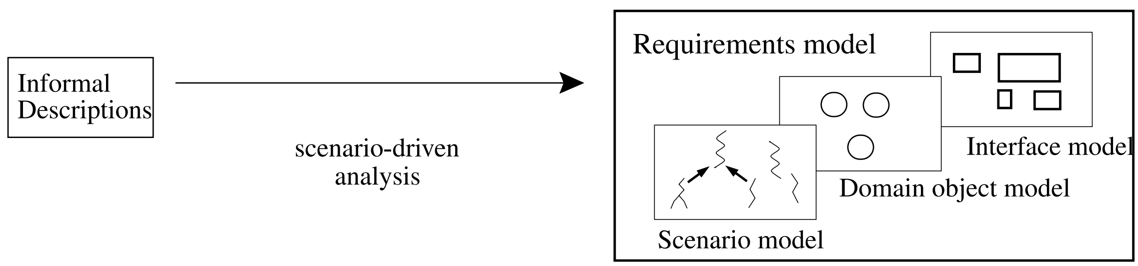


Figure 4 Scenario-Driven Analysis (SDA).

SDA produces a Requirements Model that consists of three separate small models. The Scenario Model (SM) consists of a collection of scenarios, it describes the functional aspects of the system. Each scenario captures the dynamics of the interactions between the system and one or more of its users. The Domain Object Model identifies the users that interact with the system. It indirectly defines the system's boundary and environments. The Interface Model captures the look and feel of the system's interfaces. Non-functional properties are not included in the Requirements Model. All the models are represented informally, either in natural language or with diagrams or combinations of these, and are developed in close association with end users. In Figure 4, squiggly lines are used to represent scenarios because they provide better visualization. In practice, scenarios are described in structured English.

Small arrows in the figure denotes the *extend* and *use* relationships (defined below) between scenarios.

In SDA, potential users of a system are identified by examining why the system is needed and by looking at the environments in which it will operate. Each type of user is analyzed to determine roles that it can play. The specific role played by a user type is called an *actor*, which represents a category of users that exhibit similar behaviors when using the system. A type of user may use the system in a number of ways (or play a number of roles). For example, a telephone user can play the role of a caller or of a called party. The identification of users/ actors effectively reveals the system's boundaries and environment.

Actors are used to find scenarios. The specific ways in which an actor interacts with the system are scenarios; a scenario is generated for every service the system provides. A scenario describes the basic course of actions that transpired between an actor and the system when the actor invokes a service and possibly its variances or error conditions (see the extend relation below). Conceptually, the collection of scenarios in the SM describes the system's behaviors or operational characteristics.

Once a stable picture of the system is obtained, the scenarios are analyzed and interrelated by two relations: *extends* and *uses*.

- Extend: This relation determines how a scenario may be inserted into, and therefore extends, another scenario. The relation allows extensions of scenarios to be described in a simple way and facilitates changes and additions to the system functionality. However, the scenario in which the new functionality is to be inserted, is required to be a complete course so that it can be described independently of the inserted course. In this way, scenarios that describe basic functionalities can be described independently of any extended functionality, and any extensions can be described without changing the original descriptions.
- Use: This relation is used to identify common parts of scenarios, so that they could be extracted and described only once. In this way, any changes to the common part will automatically affect all scenarios that share this part. The original scenarios from which a common segment is extracted is said to *use* the extracted segment. This relation helps to

eliminate inconsistencies among scenarios, but it requires decomposition of scenarios which may produce incomplete courses that do not describe meaningful behaviors alone.

Our specification method recasts these concepts in terms of LOTOS operators, see Section 4.2.4.

1.4.2 Behavior Integration at the Scenario Level

In [RKW95], SDA is extended to include a *synthesis phase* that takes SM as input and produces a Synthesized Usage Model (SUM). This new process is called Usage Oriented Requirement Engineering (UORE). It assumes, unrealistically in our opinion, that a scenario is a flow of events between exactly *one* actor and the system. The synthesis phase transforms each use case into an *abstract usage scenarios* (AUS) by differentiating events in the scenario as either user actions or system actions. Each AUS is represented by a kind of FSM that possesses two kinds of states, one representing user actions and the other system actions. Transitions in the AUS are labeled with the messages that pass between the actor and the system. A *usage view*, which captures the behavior of one actor and the system, is then created for each actor by finding similar parts of abstract usage scenarios for that actor and merging them. The collection of usage views, one per actor, makes up the so called synthesized usage model (SUM), which describes the usage of the system.

We believe that the assumption made by UORE that a scenario should describe the interactions between *one* actor and the system is overly restrictive and unrealistic. The process can be easily adapted to deal with scenarios that involve more than one actors by distributing the events in the scenarios according to actors. The products of the distribution would be sequences of actions, one per actor, that retain the relative order of the original scenarios. This is the approach taken in our technique of scenario integration (Section 4.2.4).

1.4.3 Behavior Integration at the LOTOS Level

Several approaches for integrating partial system descriptions by integrating their LOTOS representations are documented in the literature. In [Rud92], the author attempts to compose LOTOS behaviors in the context of behavior inheritance. And in [IYK90], a LOTOS specifications combinator, called *merge,* for combining two specifications is proposed. This *merge* operator takes two operands (which are behavior expressions) and produces a specification that contains the combined behaviors. The definition of this binary operation is given in the context of incremental design, so that it is known that the specifications to be combined could be merged. In addition, the steps for deriving the combined specification are not given, only the relations that the resulting specification should satisfy with respect to the two original specifications are discussed. In other words, [IYK90] only tells what behavior the combined specification should have but does not tell which two specifications can be merged or how they are merged. The derived specification is required to satisfy the LOTOS extension relation [BSC87] with respects to both input specifications. The proposed combinator is not applicable to all LOTOS behaviors, in particular, it cannot handle non-determinism.

We employ a more pragmatic approach, integration at a less formal level. We formalize the scenarios to be integrated as much as possible so as to bring them to a uniform state, and then we do the integration informally. The main disadvantage of this approach is its informality, even though LOTOS concepts are used to aid the integration.

1.5 Organization of the thesis

This thesis is organized as follows. In Chapter 2 we give an informal description of the GSM system. Chapter 3 contains a brief discussion of LOTOS as a system specification language. In Chapter 4, a scenario-oriented specification process involving a semi-formal approach for relating and integrating them is described.

Chapter 5 presents a LOTOS specification of GSM. In this chapter, we attempt to illustrate the application of the specification method given in Chapter 4, but an illustration of the whole process cannot be given because this is long and difficult to express.

In Chapter 6, we investigate techniques for generating scenarios from specifications and for using LOTOS specifications to “recognize” scenarios that may be at slightly different abstraction levels. The investigation focuses on the *step-by-step* and *symbolic execution* modes of the ELUDO toolkit. We present the conclusion of the thesis and possible areas for future research in Chapter 7.

CHAPTER 2

The GSM Mobile Telecommunication system

2.1 Introduction

This chapter describes the standardized radiocommunications system GSM. The term GSM refers to both the standard and the system that is standardized. GSM, the system, is a digital public telecommunications system designed to provide mobile land-based voice and data services. A GSM network is also called a Public Land Mobile Network or PLMN. GSM, the standard, was drafted by the European Telecommunication Standard Institute (ETSI) in response to a need for a digital networks that could provide compatibility of service for mobile users. The rationale behind the standardization was to provide seamless and transparent mobile telecommunication services across European national boundaries (a PLMN cannot straddle national boundaries.)

The descriptions contained within this chapter are based on the standardization documents [ETSI92], formally known as the Phase 1 GSM Recommendations, [MoP92], [ReC90], and [MoJ94]. Only system properties that could be specified by LOTOS will be described. Specifically, non-functional properties shall not be discussed.

Section 2.2 contains a brief discussion of GSM services. Section 2.3 and Section 2.4 contain descriptions from the physical and distributed functional viewpoints, respectively. Section 2.5

contains a short discussion on the protocols of GSM, and Section 2.6 discusses the future of mobile communications.

2.2 Services of GSM

The main feature of GSM is *national roaming*, a service feature which enables subscribers to access GSM services in any geographical area serviced by a PLMN (provided that PLMN has a service agreement with their home PLMNs.) A home PLMN is the PLMN in which a user subscribed for service, and from which she will receive the bill. National roaming relies on the following two key service elements:

- for a call towards a mobile subscriber, the location of the mobile subscriber must be known. Tracing the movement of the mobile subscriber within a network is done through *location management*.
- during a connection, the link between a mobile subscriber and the infrastructure must be maintained automatically and transparently wherever the subscriber may wander. This process is called *handover*.

These two procedures constitute the functionalities that differentiate a cellular system operation from fixed-line system operation.

A related concept to national roaming is *sim roaming*, which allows for a SIM (a smart card containing subscriber identity and information that must be inserted into the mobile station in order to access services) to be used in any mobile station, see Section 2.3.1.

2.2.1 Basic Services

GSM telecommunication services include both speech and data services, similar to those delivered by fixed networks. Here, speech and data services have the usual meanings as in communications; in speech service, the information is voice, whereas “data services” refers to everything else, such as text, images, facsimile, etc. The best known service provided by GSM is basic telephony, which is delivered to small personal portable terminal equipment called mobile stations (MS).

2.2.2 Other Services

Beside telephony and data services, GSM is also capable of providing two other classes of services: Supplementary Services (SS), and Short Message Services (SMS).

Supplementary services are also known as service features, and are not specific to GSM. They modify and enrich the basic services by providing a means for the user to choose how calls toward or from herself are treated by the network, i.e., block all incoming calls for a certain duration. In the Phase 1 Recommendations, only a few features are fully specified, but it is planned that later versions of the standard will define fully all supplementary services provided by fixed networks.

SMS enable short alphanumeric messages of several tens of characters to be sent and received by a user in a point-to-point manner. They also allow broadcasts of short messages of a general nature (e.g., road traffic information) at regular intervals to all subscribers in a given geographical area.

2.3 GSM from the Physical Viewpoint

The basic architecture of a GSM network is depicted in Figure 5. The configuration shown presents all the possible signalling interfaces that could be found in any PLMN. Three subsystems are identified:

- the mobile station (MS)
- the base station sub-system (BSS)
- the network and switching sub-system (NSS)

2.3.1 The Mobile Station (MS)

The mobile station (MS) is the terminal equipment which the subscriber uses to access and obtain services from the network. It is usually the only equipment a GSM subscriber ever sees. The MS is composed of two components: the mobile equipment (ME), and the subscriber identity module (SIM).

The mobile equipment (ME) performs generic radio functions, and provides an interface to the human user or other terminal equipment such as a personal computer or a facsimile machine.

One of the main goals of the standard is to allow MEs produced by different manufacturers to be compatible with any PLMN.

Subscriber specific information is stored in a subscriber identity module (SIM) which is given to the user upon registration with a network operator. The smart card-size SIM must be inserted into an MS in order for the MS to be functional. SIM is removable and can be used with different MEs, i.e., one that comes with a rental car. In this sense, the SIM *is* the GSM subscriber.

A number of identifications are associated with a SIM (or MS). The International Mobile Subscriber Identity (IMSI) is used for internal purposes such as subscription identification and, to protect against fraud. It is rarely sent over the radio interface. The temporary Mobile Subscriber Identity (TMSI) is used instead to identify subscriber over the radio interface. The mobile station ISDN number (MSISDN) identifies uniquely an MS as an ISDN terminal and is the public identifier of a subscriber, i.e., it is the directory number that is dialed when a subscriber is called.

At any time, a functional MS (with a SIM plugged in) is in one of two modes. It is in *dedicated mode* if it is involved in a connection with the network; otherwise, it is in *idle mode*.

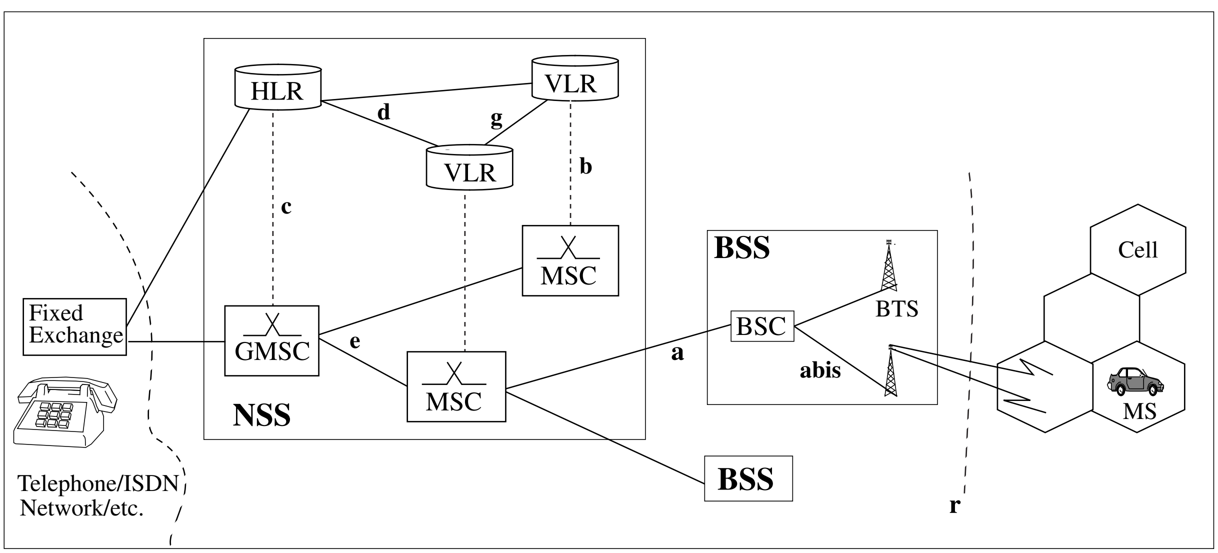


Figure 5 Architecture of GSM. The interfaces have been standardized.

2.3.2 Base Station Subsystem (BSS)

The BSS is a collection of infrastructure machines responsible for providing stable connections over the unstable radio path between MSs within a certain area and NSS entities. A BSS consists of a base station controller (BSC) and a number of base transceiver stations (BTS).

Base System Controller (BSC)

A BSC manages the radio interface through the remote control, at the Abis interface, of one or more BTSs. It allocates and releases radio channels and ensures that connection with an MS is not lost when the MS moves from one cell to another (by performing the handover procedure described in Section 2.4.2).

Base Transceiver Station (BTS)

A BTS is the physical machinery that performs radio transmission and reception. Each BTS provides radio coverage for a geographical area called a *cell*. The BTS provides communication links to the MS over the radio interface via logical channels.

2.3.3 The Network Switching Subsystem (NSS)

Unlike the BSSs which exist only to provide stable connections, NSS is the hub of GSM. It consists of entities which perform switching, as well as those that store subscriber information. Although the standard describes some of these entities as separate machines, an implementation may group two or more of them in one physical machine, e.g., VLRs are usually realized in the same machine as MSCs, forming MSC/VLR pairs. However, more often NSS entities are implemented in distant machines with no direct links. Distant machines communicate using the CCITT Signalling System No. 7 (SS7) network as the underlying transport mechanism.

In order to provide communication services to and from subscribers of other networks, NSS entities must interface with external networks, e.g., PSTN[1], ISDN. SS7 is also used as the underlying transport mechanism between NSSs and external networks. To avoid dealing with too much detail, we abstract from SS7 and assume direct links between NSS entities and between NSS and external networks.

1. PSTN = Public Switched Telephone Network. A PSTN is a fixed-line telephone network.

Mobile Switching Center (MSC)

The MSC performs switching function for all MSs located within a collection of cells known as *MSC area*. To obtain connections with MSs in its area, an MSC interfaces by dedicated lines with several BSSs at the *a* interface.

There is one or more MSC in a PLMN that additionally serves as a gateway point in/out of the PLMN. Such an MSC is called a Gateway MSC or GMSC.

Home Location Register (HLR)

The HLR stores subscription information such as service profiles and identifications (IMSIs and MSISDNs) of MSs registered in that PLMN, as well as the identities of the MSC areas in which the MSs are currently located. Each subscriber is registered in only one HLR which performs its charging and billing functions.

Visitor Location Register (VLR)

A VLR may be associated with a single MSC or be shared among several MSCs. It may be a separate entity or integrated with an MSC. In the latter case, the *b* interface becomes internalized and the combined entity is known as MSC/VLR. A VLR is used by an MSC to temporarily store subscription data and location information (at a more precise level than in the HLR) for those subscribers currently roaming in its MSC area.

2.4 GSM: Functional Viewpoint

The distributed nature of GSM implies that some of its procedures involve distant machines. The procedures can be grouped into a number of layers [MoP92] based on closeness in their purposes, as shown in Figure 6. The vertical ordering of the layers is intended to imply that (in some cases) higher layer procedures depend on or use lower layer procedures. For example, in order to perform call setup (a CM procedure), a stable radio connection must first be established (by an RR procedure called Network Access) between the mobile station and the PLMN. In this scenario, the CM layer ‘uses’ the services of the RR layer directly, and the MM layer is completely bypassed. Thus, in a sense, the higher layers depend on the services of the lower layers. This is the reason why a part of the CM layer is shown directly over the RR layer in Figure 6. It should be noted that this notion of layering is not identical to that of the OSI model. In the OSI model, an N-layer entity can only use the services of the (N-1)-layer, i.e., messages sent from or received by an N-layer entity must pass through the N-1 layer. The layer

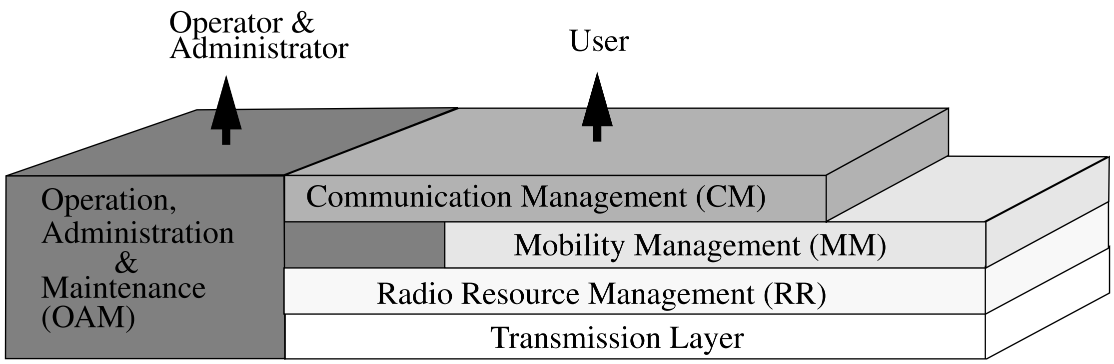


Figure 6 Functional planes of GSM

model of GSM should be thought of as a grouping of functions according to closeness in purpose rather than a description of GSM in the OSI model.

2.4.1 Transmission Layer

The transmission plane includes aspects not traditionally called transmission. This plane contains also functions of the OSI link layer and network layer which transport user data and signalling messages between communicating entities. It includes low level protocols to format data, to ensure proper sequencing, to correct errors through repetition and to route information throughout networks.

Physically, all information transmitted within the GSM system is in digital form, even at the radio interface. The MS must be capable of encoding human speech into a digital form and converting digital information to the analogue form (speech) understandable to human users. Since GSM provides its users with the capability to call users of other networks, a PLMN must be capable of interfacing with analogue networks (e.g., PSTN), digital ones (e.g., ISDN), and other networks.

At the radio interface, different frequency ranges are used in adjacent cells and distant cells may reuse frequency ranges. In each cell, the access mode is a combination of time division multiple access (TDMA) and frequency division multiple access (FDMA). So the packet of signal/data transmitted is a burst of radio energy transmitted in an assigned frequency band. The technique of frequency hopping, where the frequency used by a channel is regularly changed, is employed to help ensure privacy and combat jamming.

2.4.2 The Radio Resource Management Layer (RR)

This layer deals exclusively with the radio interface, and is most relevant at the BSS/MS interface. The RR entities contain elementary procedures for radio management, e.g., establishing, maintaining, and releasing radio links. These functions are mainly performed by the MS on the mobile side and the BSC on the infrastructure side, although the MSC may be involved in some cases of handover. The main procedures of this layer are: accessing the network, and handover. But first, the initial activation of the MS is described.

Initial Activation of the MS

Initially when a terminal equipment equipped with a SIM is put into service it scans the radio channels to locate the Frequency Correction CHannel (FCCH) and become synchronized. It then performs the network access procedure (described below) to request channels to do location update (Section 2.4.3) to register its location. After successful location update, the MS is in its normal operational state (idle mode), and can make or receive calls. During idle mode, the MS listens to broadcast channels and monitors signal strength from the serving BTS and the nearby BTSs. If it is found that stronger signals come from a nearby BTS, the MS will attempts to do a location update there (actually, the algorithm for using cells is much more complex.)

An MS listens to broadcast channels to stay synchronized with the network and to await instructions. For example, when there is an incoming call towards the MS, it is instructed on the Paging CHannel to access the network. Whenever an MS needs to access the network, and for whatever reason, it must first perform the “access” procedure to request radio channels.

Network Access Procedure

The access procedure consists of sending a short message (only 8 bits!) over the random access channel (RACH) requesting a channel. Three bits of this request specify the purpose (paging response, handover, CM service, etc.) and the other five bits represent a random number. The purpose for a request will tell the serving BSC how to handle the request, i.e., should channels be granted, which type of channels and how many should be given, etc.

The network has no method of knowing when mobile stations will transmit requests, and therefore random access attempts cannot be scheduled to avoid simultaneous independent transmissions (a situation known as collision). A given MS must be able to correlate a channel assignment from the network with its own request. This is achieved by having the network send back the random number with the channel assignment. An MS will know that a channel assignment is not for itself if the accompanying random number does not match the one it included in the request.

In normal load situations, after receiving an indication of a channel request, the BSC chooses a free pair of signalling and data channels belonging to the cell from which the request originated, and activate it in the BTS. After having received an acknowledgment for this activation from the BTS, the BSC then builds an initial assignment message to be sent on the PCH. The activation process requires the BTS to prepare for the access of the MS on the newly allocated channels.

In congested situations, when no channel is available for allocation, the BSC may choose not to answer, to grant only a signalling channel, or to send back a rejection indication. If no response is made to a request, an MS may repeat its attempt. In the case where only a signalling channel is initially granted, a data channel (if one is needed) will be granted either after it is known that the call can go through (i.e., the called party is not busy), or even later when the called party has answered. Note that in location updates, there are no user data and data channels are not needed for successful execution.

Basic Handover Procedure

The handover procedure ensures a quality connection over the air interface during a call. Information on the received signal quality from an MS is collected and processed by the serving BSC. When the BSC detects that the signal quality is below a certain minimum threshold, it performs a handover of the MS to a new cell or to a new channel in the same cell.

In the discussion to follow, the suffix “old” shall be used to refer to machines along the communication path before the handover, and “new” shall refer to the path after the handover.

There are three cases of handover (see Figure 7) depending on the positions in the infrastructure hierarchy of the BSC-old and BSC-new:

Case 1: Intra-BSC handover: the MS is to be served by a new cell (BTS) which is under the control of the same BSC as the currently serving cell, that is BSC-old and BSC-new are one and the same.

Case 2: Inter-BSS Intra-MSC handover: the new cell is controlled by a BSC-new which is different from BSC-old, but the two BSCs are controlled by the same MSC.

Case 3: Inter-MSC handover: this case involves either two or three MSCs. An anchor-MSC is an MSC which was originally involved with the Access procedure. If the anchor-MSC is also MSC-old, then only two MSCs (anchor-MSC or MSC-old, and MSC-new) are involved, otherwise three MSCs (anchor-MSC, MSC-old, and MSC-new) are involved. In the latter case, an inter-MSC handover (from anchor-MSC to the now MSC-old) must have occurred.

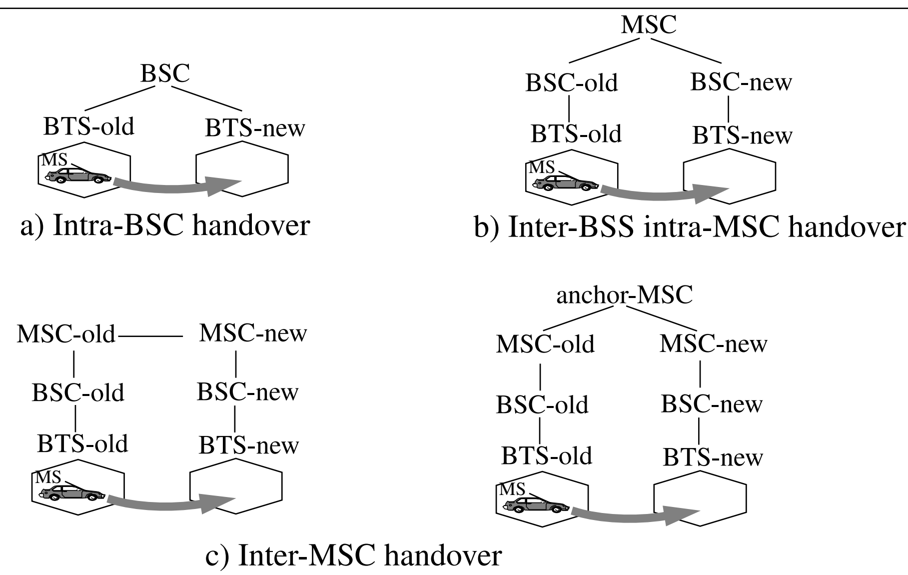


Figure 7 Three cases of handover

The anchor MSC is the MSC on which the call as originally established. In no case does this MSC relinquish its communication control to a new MSC. This design choice reduces the complexity of the charging problem, it is much simpler when one MSC follows a call from its beginning to its end.

The procedure for all cases of handover consists of two phases:

Phase 1: BSC-old asks BSC-new to establish a new communication path. Then BSC-old sends a handover command to the MS to access the network on the new channel.

Phase 2: the MS accesses the new channel, which triggers the switching of paths in the infrastructure and the release of the old path.

2.4.3 The Mobility Management Layer (MM)

The mobility management layer handles issues peculiar to cellular networks: mobility and security managements. The functions involved are:

- location update: a process by which the databases are updated when the MS moves across certain cell boundaries.
- security management: this includes subscriber authentication to ensure that he is an authorized user, and ciphering of radio transmissions to ensure privacy. Strictly speaking, ciphering is an RR function.

The network entities that are most involved in location update and security management are the MSC, VLR, and HLR. The functioning of the MSC and the VLR are closely coupled in these functions. Therefore, they will be referred to as one entity, the MSC/VLR, in this section.

Location Update Procedure

The GSM infrastructure must keep track of the location of MSs roaming in its service area. This information which is held in the HLR and the VLR (and also stored on the SIM) is crucial for mobile-terminated calls.

The normal reason for location update is that the MS decides that the location area best fit to serve the subscriber must be changed. The MS starts the location update procedure by initiating the Access procedure (Section 2.4.2), as for any dialogue between the MS and the infrastructure. Then it sends the request toward the MSC/VLR controlling the new cell. This MSC/VLR may be the same as before, if it controls both the previous and the new location areas, or a new MSC/VLR.

The MSC/VLR may answer the MS autonomously or may have to first consult the HLR. The MSC/VLR answers autonomously if the subscriber is already registered in the VLR, that is the new and old location area are under the control of the same MSC/VLR. In this case, the location update request is usually accepted after some actions such as authentication and setting cipher key. The successful conclusion of a location update involves the assignment of a new temporary mobile subscribe identity (TMSI) to the MS.

The MSC/VLR consults the HLR when an MS asks for registration under a new MSC/VLR. The new MSC/VLR knows which HLR to contact from the MS's TMSI, or in rare instances when the TMSI is not available the IMSI is used. The HLR determines whether or not to accept to register the MS in the new MSC/VLR by considering the subscription limitations of the user. On a positive result, the HLR will send the new MSC/VLR information of the user, and will inform the old MSC/VLR to remove the subscriber's record from its database. The subscription information sent to the new MSC/VLR allows it to perform more efficient call setups (Section 2.4.4).

If the MSC/VLR receives a negative result from the HLR, it will erase all information about the subscriber and sends a location update reject indication to the MS.

Security Management

The open nature of radio transmission makes it more prone to eavesdropping and fraud than fix wire transmission. The security functions of GSM have two aims: to guard the privacy of users, and to protect the network from unauthorized access.

Preserving the privacy of the users is achieved by a number of means. Ciphering of transmission over the radio path and frequency hopping prevent eavesdropping. In addition, user identity protection is achieved with the use of a temporary identity alias, the TMSI, which is used in lieu of the subscriber identity IMSI over the radio interface. This alias is allocated by the MSC/VLR at every successful location update so that it cannot be associated permanently with any specific subscriber.

Prevention of unauthorized access is achieved by an authentication procedure. In this procedure, the MS applies an algorithm on a *RAND*om number chosen by the infrastructure and on a user-specific secret key *Ki* to produce a *S*igned *RES*ult (or *SRES*) which is verified by the network. The secrecy of *Ki* and the algorithm, called *A3*, is the cornerstone of this security mechanism.

In fact, the parameters *Ki* and *RAND* are used in another computation by an algorithm known as *A8*. The result of this computation is the key *Kc*, used by the encryption algorithm to cipher and de-cipher data at the radio interface.

2.4.4 Communication Management Layer (CM)

The communication management layer consists of conventional call control procedures, i.e., setting up calls between users, and maintaining and releasing them. This layer also includes the means for users to have some control over the management of calls through supplementary services (SS) such as call forwarding, and to send or broadcast short alphanumeric messages. The latter services are known as short message services or SMS, see Section 2.2.2.

Mobile-Terminated (MT) Call Procedure

For MT calls, the external network uses the GSM directory number (called MSISDN) of the called MS to locate a record in the MS' home HLR. This record contains the identity of the MSC where the subscriber is currently visiting. If for some reason the external network cannot interrogate the HLR, then the call is routed to the GMSC. The GMSC then interrogates the HLR and routes the call to the appropriate MSC, which then queries its VLR for the exact location (i.e., cell identity) in which the MS is currently roaming. The MSC then instructs the BSC to page the mobile station over its paging channel (PCH). The mobile subscriber responds by performing the Access procedure, and upon contact with the MSC it is instructed by the MSC to alert the user.

Mobile-Originated (MO) Call Procedure

To a GSM subscriber, using an MS to call a fixed or mobile subscriber follows the same basic steps as for (old) fix-line telephone. Typically, a call starts by the user clicking the “Power on” button, an action which is answered by the MS (not the network infrastructure) with a dial tone. Once the last number is entered, the SEND button is pressed, after which there are a few moments of inactivity to permit the call to be cancelled. Then *all* the digits are sent to the network at once. This is different with respect to basic telephony where contact with the network is made as soon as offhook is performed and digits are sent to the network one by one as they are being keyed.

The MS makes contact with the network by executing the Access procedure, after which it requests, with the called number, the MSC to set up a connection. Upon receiving the setup message, the MSC checks with the VLR to determine whether the service can be provided for the requesting MS. This determination occurs locally because the subscriber's information were sent by the HLR in location update and stored in the VLR. Whether the service can be provided depends on the subscription characteristics and on the availability of network resources.

In case of failure, the MS is returned to idle mode. In case of success, the MSC starts connection establishment toward the called party by sending, for instance, an ISUP[2] initial address message (IAM) in the case of communication with ISDN.

Eventually the external network will report the failure or success of the call. The MSC reacts respectively by disconnecting the MS, or alerting it to ring. In the latter case, an end-to-end connection path may not be established until the called party has performed an offhook.

2.4.5 Operation, Administration and Maintenance Layer (OAM)

The Operation, Administration and Maintenance layer contains functions which enable an operator to control and configure the PLMN. It is also the mechanism which the GSM administrator employs to measure and record, for the purpose of billing, usage by individual MSs. This layer is not strictly above the other layers in terms of services, since it does not enhance directly the services offered by these layers.

2.5 Protocols of GSM

Figure 8 shows an overview of the signalling architecture in the GSM transmission chain. The horizontal axis corresponds to the distribution of the various physical machines and the vertical axis corresponds to the functional planes. The procedures described in this chapter and

2. ISUP = ISDN User Par. It is a call management protocol for ISDN networks.

specified in this thesis concerns mostly protocols of the RR, MM and CM layers shown in the figure.

2.5.1 RR Protocols

The RIL3-RR (Radio Interface Layer 3) protocol enables mobile stations and BSCs to cooperate for the management of radio resources. This protocol also appears on the *abis* interface where it works closely with the RSM (Radio Subsystem Management) protocol. The latter allows a BSC to control a set of BTSs.

A similar management function is supported on the *a* interface by the BSSMAP (BSS Management Application Part[3]) protocol, which enables an MSC to manage several BSCs. The BSSMAP is also partially responsible for handover management (in collaboration with the MAP/D protocol, see below).

2.5.2 MM Protocols

The main protocol for this layer is the RIL3-MM which bears many similarities to the protocol of the CM layer. For this reason, it is discussed in the next section with the RIL3-CM.

2.5.3 CM Protocols

The upper layer protocols RIL3-MM and RIL3-CC define rules for signalling exchanges between the MS and NSS entities. Although these protocols also appear at the *abis* and *a* interfaces, both the BTSs and BSCs are “transparent” to these signalling exchanges, i.e., they are relay points and are not aware of the semantics of RIL3-MM and RIL3-CC messages. Non call-related signalling specific to GSM corresponds to many protocols which are grouped together as MAP/*x* (Mobile Application Part) protocols where *x* is an interface, e.g., MAP/*d* is that part of MAP that is applicable at the *d* interface between the MSC/VLR and the HLR.

The protocols that govern the interactions between the MSCs and external networks are specific to each of these networks. The functionalities of GSM and other communications networks are more or less the same, but variation exists according to the type of network, and also to the specific implementation. The basic examples of protocols at the boundaries of GSM are TUP[4] and ISUP[5], for communications establishment with PSTN and ISDN networks

3. MAP protocols deal with non-call related function. BSSMAP allows MSCs to control BSSs.

respectively. TUP and ISUP are the standard circuit call management protocols used in SS7 environments [MoP92], and in the context of GSM, they interwork directly with RIL3-CC.

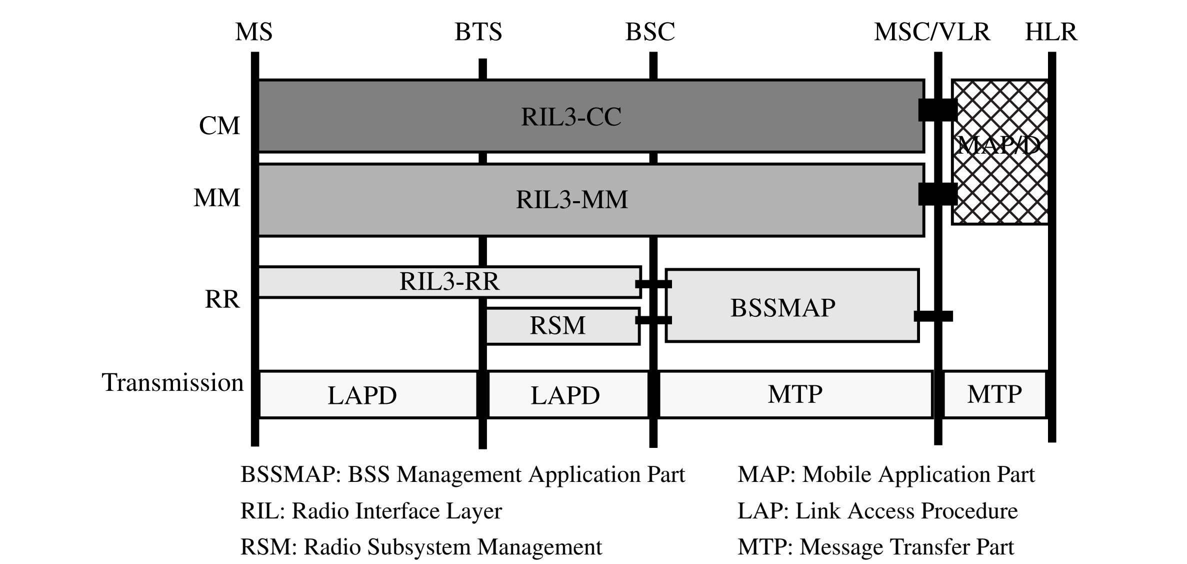


Figure 8 Protocols of GSM

2.6 Future of Mobile Communications

GSM is an evolving standard. At the time of writing, Phase 2 Recommendations have been released. Phase 1 and Phase 2 GSM are classified as second generation communication systems, and were intended mainly to provide transparent and seamless pan-European land-based mobile services. However, they were adopted by many other countries. The North-American counterpart of GSM is known as IS-54. (First generation systems use analogue signal over the radio interface and provide only voice services.) These second generation systems are capable of providing voice and data services.

Third generation mobile communications systems are under study and drafted standards are planned for 1998 [Rap95]. They are known as UMTSs or Universal Mobile Telecommunication Systems. UMTSs will truly be able to provide global services as they will include satellite technologies to provide connections with users over water and in the air. These

4. TUP = Telephone User Part. It is a call management protocol for PSTN.
5. ISUP = ISDN User Par. It is a call management protocol for ISDN networks.

third generation systems will provide sufficiently high bit rates (up to 155 Mb/s) to be capable of providing multimedia services [DaF95]. It is forecasted that by the years 1996/97, the number of new subscriptions world-wide to mobile services will exceed new subscriptions to fixed-line services at around 35 millions per year [Rap95].

CHAPTER 3

LOTOS Principles

3.1 Introduction

The FDT LOTOS [ISO89] was developed and standardized by ISO for the specification of, among others, distributed concurrent information systems. The language has not yet reached the level of maturity that some other, more widely used FDTs have attained. It is still undergoing extension and enhancement.

This chapter discusses general principles of LOTOS, a tutorial of the language is not given but the interested reader is referred to [LHF92][BoB87]. Section 3.2 contains brief discussion on the two components of the language and some of their features. Section 3.3 explores the notions of *specification* and *model*. Subsection 3.3.1 comments on the advantages and disadvantages of specifying a system in LOTOS, and Subsection 3.3.2 discusses the executability of LOTOS specifications. In Section 3.4, a number of specification styles are reviewed. Section 3.5 presents some of the LOTOS techniques used in our specification of GSM.

3.2 The LOTOS Language

LOTOS consists of two clearly separate component: 1) a *data* component, and 2) a *contro*l component. LOTOS specifications, thus, are also composed of two parts, one describing data types and the other describing behaviors.

3.2.1 The Data Component

The data component of LOTOS is the algebraic abstract data type (ADT) language ACT ONE [EM85], which is based on equational models. This component is most suitable for describing data structures and value expressions. The choice of *abstract* data types (as opposed to *concrete* data types) permits implementation independent specifications. With ADTs, it is possible to represent only the essential qualities of data and their associated operations without indicating how they are actually represented and manipulated in computer memory. The abstract nature of ADTs gives it a high level of expressiveness. Since they are defined in terms of equations without side effects, ADTs can be "built up" easily to arbitrary complexity. Indeed, they are capable of denoting the Turing machine which implies (in principle) that it can denote any systems.

Unfortunately the current version of LOTOS does not augment ACT ONE with any pre-defined data types, although a small library of some basic data type specifications is given in the standard. This deficiency forces the specifier to write a specification for almost every data types he needs. Commonly-used types such Integer, Record, etc., that can be taken for granted in concrete data type languages have to be specified from scratch. This often results in large specifications, which are difficult to understand and slow to compute. However, the following features of the data part of LOTOS offset some of the negative aspects:

- reference to a library of predefined types
- combination and extension of data types to form new types
- renaming of generic types
- actualization of parameterized types

These features support re-use of ADT specifications.

3.2.2 The Control Component

The control component of LOTOS is a process algebra and has well-defined semantics. It is based on Calculus of Communicating Systems (CCS) [Mil80] and Communicating Sequential Processes (CSP) [Hoa85]. In this part of the language, systems and their components are represented by *processes*, which are encapsulated units of data and behaviors. Processes display behaviors to their environments in terms of permitted sequences of observable events. That is, they appear as a black boxes to their environments, and the environments have no knowledge of their internal structures or mechanisms. Processes may also possess *internal* unobservable behaviors.

The encapsulation provided by the process concept makes this part of the language highly suitable for specifying objects. Indeed, LOTOS processes are often used to represent objects with object communications represented by process synchronization. In addition, the way processes are composed in a specification can be used to denote structure of a system.

A process interacts with its environment by means of synchronization at common points called *gates*. LOTOS gates may be used to model logical or physical interfaces between a system and its environment. Values, specified by ADTs, may be passed and received at these gates during synchronization. However, gates may not be dynamically created, nor passed in synchronizations.

The basic units of interactions are *events* (also called *actions*), which are atomic, instantaneous and synchronous behaviors. Each event is associated with a gate, namely the gate at which the event occurs. Event denotation is structured, consisting of a gate name and an optional list of data fields (called *experiments* in LOTOS terminology). The general structure of LOTOS events is

```
gatename !value_offered ?valued_received:Value_sort ...
```

An event occurs only if all processes that participate in it are ready to interact. When an event takes place, all the processes involved in that event synchronize and have a common view of the interaction. This synchronization is interpreted as communication, and permits values to be passed between processes.

Events model real-life occurrences in an abstract way. The degree of detail required in a specification determines the events that are considered. For example, a communication between two distant entities in a distributed system may be modeled by an (single) event that represents the transmission and propagation of the message through the network. Alternatively, the hops in the propagation of a message through the network can be modeled as distinct events, or in an even more extreme view, the transmission of individual bits can be represented as a sequence of distinct events. All these models represent the same communication at different levels of abstraction.

A major deficiency of the control part of LOTOS is that it does not possess inheritance, implying that existing process definitions cannot be re-used to produce new processes. [Rud92] contains a readable discussion on this shortcoming of LOTOS and proposes new operators to add inheritance to the language.

3.3 What exactly is a LOTOS Specification?

A LOTOS specification of a system is a representation of a model of that system. A model is an abstraction, a conception that exists in the mind. A model typically contains only the essential aspects of a system *viz.* the system properties of interest. A representation of a model such as a LOTOS specification is a rendering of the model, a visualization of the model. Although a model and its representation are distinct concepts, they are not independent and utilizing an appropriate notation to describe a model can greatly affect the ease of understanding and reasoning about the model by bearing out its pertinent aspects. This means that the comprehensibility and ease of reasoning of a LOTOS specification is aided in areas where LOTOS is strong and hindered in areas where it is weak. In fact, in cases of complex systems a representation may even be necessary just to make the modeling possible. Thus, a model and its representation may sometimes be thought of as equivalent.

For most practical purposes, the terms *model* and *specification* can be used interchangeably, and this thesis does not distinguish between the two when no misunderstanding is possible, i.e., the term *LOTOS model* will sometimes be used instead of *LOTOS specification*.

3.3.1 LOTOS Models: Pros and Cons

In general, the ability to represent a model is limited by the support provided by the notation used. The absence of the inheritance concept in LOTOS means that the language cannot *naturally* describe, and therefore aid reasoning on, object-oriented models. It can, however, represent object-based models, where inheritance is not essential. LOTOS also lacks the concept of time, implying that many non-functional properties cannot be specified elegantly. Some proposals for timed LOTOS have been proposed [BLT90] [VTZ94][QuF87]. Although these deficiencies are significant, LOTOS has many other properties (such as encapsulation, synchronization, events, concurrency, and others) that make it suitable for describing systems.

Some of the benefits of using LOTOS are related to the way descriptions may be organized, while others are due to the ability to execute specifications. LOTOS possesses concepts (i.e., operators such as ||, |[]|, etc.) that allow to relate scenarios to each other and form unified models. Because the language supports the representation of unified models, LOTOS specifications avoid some of the problems associated with separate and partial representations (see Section 1.2.3). Figure 9 illustrates the difference between representing a system by a collection of scenarios and by a LOTOS specification. In fact, LOTOS can handle most of the problems discussed in Section 1.2.3.

- Since a LOTOS specification is a unified model, it is possible to show how scenarios relate to each other, i.e., how they are connected. Amalgamated scenarios can represent the system states in which each scenario can be initiated and terminated.
- LOTOS can represent the intersection of scenarios by describing the common segments only once. For example, the following two scenarios

  ```
  a; b; c; d; exit
  a; b; e; f; g; exit
  ```

  may be specified as

  ```
  a; b; (c; d; exit [] e; f; g; exit)
  ```

  to emphasize the common segment (other kinds of integration are possible).
- LOTOS is a *wide-spectrum* language. It is capable of denoting systems at a range of abstraction levels. Through the use of embedded processes and the hide operator, sce-

narios of different granularity (or abstraction levels) can be specified together without having to abstract more detailed scenarios to the level of the most abstract one.

- LOTOS is capable of representing concurrent behaviors. Concurrency may be specified in LOTOS with the parallel operators ||, ||| or |[]|. Indeed, the ability to specify current behaviors is one of the strong point of LOTOS.

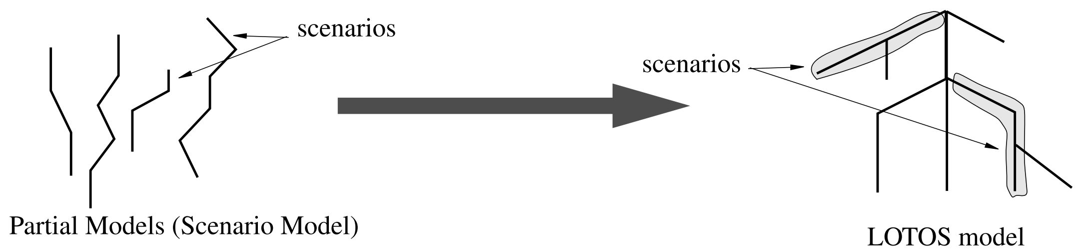


Figure 9 Comparison of the Scenario Model and the LOTOS Model.

LOTOS specifications have other advantages. One set of advantages are related to the formal basis of the language which permits consistent and unambiguous specifications to be produced. Another class of benefits involves the executability of LOTOS specifications, which could be carried out in a number of modes, e.g., step-by-step, symbolic expansion, goal-oriented. This FDT also possesses a large body of testing theory and other validation methods that could be used to validate specifications. Section 3.3.3 presents an overview of testing in LOTOS. A very important group of automatic manipulations is correctness-preserving design transformations [CPT92] which allows to formally refine designs to more concrete ones. A complete development methodology based on LOTOS has been proposed based on these transformations [LOT92]. However, automated manipulation is not always possible for every LOTOS expressions. The problems encountered with execution of LOTOS specifications are discussed in the next section.

3.3.2 Executability of LOTOS specifications

A LOTOS specification essentially denotes a Labelled Transition System (LTS) [Eer94], and execution of a specification amounts to computing its LTS. A state in the LTS corresponds to a behavior expression, and each transition label corresponds to an action that represents at least one action denotation in the specification. The set of transitions that are possible from a given state is defined by the set of inference rules [ISO89] that define the semantic of LOTOS. Inference rules define formally when a behavior expression *B* can perform an action *x* and then behave like *B'* (another behavior expression). This is commonly denoted by $B \xrightarrow{x} B'$ in LOTOS literature.

Complete execution of a LOTOS specification may be impossible due to a number of reasons. The data portions of specifications are equational algebra, and can be executed in practical way by interpreting them as sets of rewriting rules [Eer94]. However, it is possible that a set of rewriting rules correspond to valid specification is not executable because it may never terminate.

In the behavior portion of a specification, full execution may be impossible (practically) due to parallelism, which tends to produce state explosion as the amount of parallelism increases. In LOTOS, the semantic of parallelism is based on the concept of interleave. This concept allows multiple parallel scenarios (i.e., scenarios that could be executed in parallel) to be represented by as single scenarios with segments of the different parallel scenarios interleaved in any order, provided the relative order of segments from each scenario is preserved. For example, the interleaved behavior of the two parallel scenarios `a;b;stop` and `z;stop` could be any one of the scenarios `a;b;z;stop`, `a;z;b;stop`, or `z;a;b;stop`. Note that the number of possible action sequences quickly increases as the number or size of the parallel scenarios increases.

The ultimate state explosion problem is caused by behavior expressions of the type shown in Figure 10, which permits infinite numbers of actions at a given system state.

```
process MSInstaller[ri,oami](usedMSISDNs:MSISDNSet):noexit:=
   oami ?dn:MSISDN [(NotIn(dn,usedMSISDNs)) and (dn <> noMSISDN)];
   MStation[ri](IMSIof(dn),noTMSI,dn)
   |||
   MSInstaller[ri,oami](Insert(dn,usedMSISDNs))

   where
      ...
endproc
```

Figure 10 A LOTOS expressions that describes infinite behaviors.

3.3.3 Test Execution on LOTOS Specifications

In Section 6.3, we discuss preliminary results on using LOTOS specification to “recognize” scenarios that are more or less ate the same abstraction level as the specification. The purpose of this “recognition test” is to determine if the scenarios have been specified in the specification. The technique for scenario recognition test is based on test execution in LOTOS.

Normally, LOTOS test execution is performed as follows. The test is specified as a LOTOS process, say process *TestCase*. This specification can be in the simplest style, but its event structures must be the same as the ones used for the corresponding events in the specification. The specification to be tested is modified into a process, say *SpecUnderTest*. A new specification S is then created by composing *TestCase* and *SpecUnderTest* with the synchronization operator ||. If the combined specification

$$S = SpecUnderTest[g1,g2] \,||\, TestCase[g1,g2]$$

can be executed to the end of the scenario (i.e., to the end *TestCase*), then the test passes; otherwise it fails. In fact, because LOTOS specifications may be non-deterministic, three possible results from test execution have been defined: must pass, may pass, and reject [QPF88][NiH84].

3.4 Specification Styles

A major concern when producing LOTOS specifications is the selection of appropriate specification structures or styles. In [VSS89], four specification styles are described. They are of two kinds: intensional and extensional. Specifications in extensional styles describe a system only from the perspective of an external observer. The monolithic and constraint-oriented styles are example of extensional styles. Specifications in intensional style may include internal details of systems. The state-oriented and resource-oriented styles are two examples of this kind. Note that other styles have been proposed (e.g., slice style [Vig91] and status-oriented style [StL94]) but we shall only review the abovementioned ones.

3.4.1 The Monolithic Style

This style uses predominantly the sequential composition, choice and guard operators to present system behavior as alternative sequences of actions. Specifications in this style do not reflect system architecture nor do they differentiate between internal and external system behaviors.

The exclusion of more complex operators, such as the disable and the parallel operators, implies the absence of structures that aid human comprehension and tend to produce specifications that are large in size. Consequently, this style is most appropriate for simple specifications. However, the absence of more complex operators also means that state explosion is unlikely and makes this style more friendly to tools.

3.4.2 The Constraint-Oriented Style

This style enables specification of a system's external behavior in an abstract, implementation independent and modular fashion. The specification structure is based on the different constraints that result from performing "separation of extensional concerns," i.e., the conjunction of the constraints defines the external behavior.

3.4.3 The State-Oriented Style

The state-oriented style regards a system as a single resource whose internal state space is explicitly specified as a collection of guarded alternative sequences of observable interactions. This style is not suitable for large or complex systems for at least two reasons. First, its lack of structure makes human comprehension difficult. Second, this type of systems tend to have large state spaces and they are difficult to identify. This style is useful mainly when explicit states can be readily obtained from the problem definition.

3.4.4 The Resource-Oriented Style

In this style, the specification structure follows closely a system's actual architecture. Physical resources are represented by LOTOS processes that are composed together to reflect the overall behavior and architecture of the system. The processes themselves can be written in any style, and may include internal behaviors.

This style is appropriate for specifying distributed systems at the high design level, because it can capture both the structure and behaviors of a system. The specification technique described in Chapter 4 produces specifications of this style.

Specification Approaches for the Resource-Oriented Style

There are two basic approaches for developing resource-oriented specifications. A top-down approach specifies first the system's structure (thus resulting in a skeleton structure of the specification), then fills in the system's behavior by defining the interactions between its components. The behavioral specification can also be done in two ways. One way involves specifying the system's behavior one scenario at a time by distributing the events in the scenario among the processes representing the objects involved in that scenario. This approach is suitable when descriptions of the components' behaviors are scattered over a number of scenarios. An alternative way is to specify, independently and one at a time, the system's components by filling in the bodies of the empty LOTOS processes in the skeleton specification. This approach is dependent on having descriptions of each component not scattered over multiple scenarios.

Conversely, a bottom-up approach can be taken. In this approach, a system is specified by separately specifying its components in LOTOS processes, then composing these processes to obtain a specification of the whole system. This approach is probably not very practical as proper synchronization among processes is difficult to achieve. It may be practicable if the components are loosely coupled and have clear interfaces and if they do not interact in an intricate way. This approach also favors cases where the source information describes each component separately or when such descriptions could be easily obtained.

The choice of which approach to use in a specification exercise depends on many factors such as the size and complexity of the system, the kind of information available, etc. In Chapter 4, we describe a top-down approach for constructing resource-oriented specifications.

3.5 LOTOS techniques Used in the Specification of GSM

This section discusses some of the LOTOS specification techniques that were used in our specification of GSM (Chapter 5).

3.5.1 Specification Structure that Permits Dynamic Creation of Objects

For specification styles that use processes to represent objects, there are composition structures that permit to dynamically create an unbounded number of mutually independent objects. An example of such a structure is illustrated in Figure 11. This structure is a recursive interleave composition of processes; a process representing one object (*Object*) is combined in an interleave expression with the process that instantiates it (*Installer*).

Objects are created as follows. Initially, there are no objects. This can be specified by initializing the parameter (*usedObjIds*) that represents the set of object identities already in use with an empty set. A new object, identified by a unique identity, may be created (line 2) by instantiating *Object* with a value supplied by a process outside of the *Installer* process, possibly, by the environment.

```
1   process Installer[g](usedObjIds:SetObjIds):noexit:=
2      g ?newObjId:ObjectId [NotId(newObjId,usedObjIds)];
3      (  Object[g](newObjId)
4         |||
5         Installer[g](usedObjIds)
6      )
7      where
8         process Object[g](myId:ObjectId):noexit:=
9            ...
10        endproc
11  endproc (*Installer*)
```

Figure 11 Specification structure that permits dynamic creation of objects.

3.5.2 Specification of Infinite Size Relay Buffers

Objects created by a specification structure as the one shown in Figure 11 are mutually independent in their behaviors, and cannot communicate with one another. In LOTOS, if it is required to specify communication between different instances of an object of which an arbitrary number of instances can be created, a specific type of process must be used. This process represents a communication medium between the different instances of the replicated object. The LOTOS structure required to specify such a topology is shown in Figure 12. In this figure, the first event (line 10) of each alternative behavior represents the sending of a message by the sender object and the receiving of the message by the communication medium. The second event (line 11) in each behavior represents the medium sending the message to the originally intended recipient object. That is, the process `Buffer` is a *relay* process, it accepts messages from one object and transparently relay the message to the recipient. Two one-way gates are used to connect the two communicating processes to `Buffer` instead of a single two-way buffer because the latter solution has a side effect. To see the side effect, consider the two one-way gates `toBuf` and `fromBuf` in Figure 12. If these gates were replaced by a single gate, line 33 would be able to synchronize with line 10, which is equivalent to receiving a message from the buffer when no message has been sent to it.

Note that `Buffer` represents an infinite size buffer that can accepts any number of messages simultaneously and that may send them out in different orders from the order they were received. This specification structure may also used in the representation of objects that relay messages between objects of different kinds. For example, in the specification of GSM described in Chapter 5, it is the basic structure of the process that represents BTSs, which relay messages from MSs to BSCs and in the opposite direction.

```
...
hide toBuf, fromBufin

Installer[g,toBuf,fromBuf]
[toBuf, fromBuf]
Buffer[toBuf,fromBuf]

where
  process Buffer[toBuf,fromBuf]:noexit:=
    toBuf ? repient_id:ObjectId ?m:T;
    ( fromBuf !recipient_id !m;
      stop
      |||
      Buffer[toBuf,fromBuf]
    )
    []
    (*...
     other event structures*)
  endproc

  process Installer[g,toBuf,fromBuf](usedObjId:SetObjectIds):noexit:=

    g ?newObjId:ObjectId [NotId(newObjId,usedObjIds)];
    ( Object[g,toBuf,fromBuf](newObjId)
      |||
      Installer[g](usedObjIds)
    )

    where
      process Object[toBuf,fromBuf](myId:ObjectId):noexit:=
        toBuf !recipient_id !m;        (*sends message via Buffer*)
        ...
        []
        fromBuf !myId ?m:T;         (*receives message from Buffer*)
        ...
      endproc
  endproc (*Installer*)
```

Figure 12 Specification of an infinite size buffer.

3.5.3 Specification of Mobile Nodes (Gate Splitting)

As stated in Section 3.2.2, LOTOS gates can neither be passed in synchronizations nor created dynamically. This limitation would preclude the ability to specify the handover procedure (Section 2.4.2) if the interface between a network node and a mobile station is represented by a unique LOTOS gate. A very important event in handover is the transmission of a message by the currently serving cell (or BTS) to command the mobile station to start connection establishment with a new cell. The identity of the new cell is included in this command. Thus, if the interface to this new cell is represented by a LOTOS gate, it cannot be included with the event that represents the command.

In our specification, we employed the *gate splitting* technique [BFL95] to specify dynamic interfaces. This technique exploits the ability to pass data in LOTOS synchronizations and uses values of experiments in an event to *split* the gate associated with that event into multiple 'sub-gates'. For instance, the following two events

```
radio_interface !channel1 !message1;
radio_Interface !channel2 !message2;
```

occur on the same LOTOS gate, but the values of the first experiment differentiate the interfaces, i.e., the first event represents communication at the radio interface on channel 1 and the second event represents communication at the same radio interface but on channel 2.

Our specification of GSM uses this technique. The radio interface between all mobile stations and the PLMN is represented by one gate, and gate splitting is used to identify interfaces between particular pairs of mobile stations and BTSs (see Section 5.3.4).

CHAPTER 4

A LOTOS Modeling Methodology for Distributed Systems

4.1 Introduction

This chapter presents a technique for building specifications from use cases. It represents a major part of the contribution of this thesis, and is intended as a contribution towards a methodology for formal specification and validation of distributed systems. The technique is a result of our effort to rationalize the process of deriving a specification from a set of use cases provided in various formats. Although it reflects to some extent the process that was followed, it must be understood that we did not follow it closely. The output of the process is a LOTOS specification in the resource-oriented style at more or less the same abstraction level as the input use cases. The reasons for choosing the resource-oriented style are discussed in Section 3.4.4.

The technique is defined for distributed systems but is applicable to other types of systems as well. We utilized this technique to specify the standardized GSM mobile communication system.

The chapter is organized as follow. Section 4.2 presents the specification process, and Section 4.3 discusses its application to technical standards for distributed systems.

4.2 System Modeling In LOTOS: the Scenario-Oriented Approach

The process of developing formal system specifications from informal descriptions involves creativity and intuition, and is difficult to formalize and describe. LOTOS specifications are typically developed incrementally by an iterative process of specification and validation. The process first establishes a structure for the specification, then a subset of the system behavior is added on top of this structure, and the resulting specification is put through some kind of execution (i.e., step-by-step execution, expansion, etc.). The trace from the execution is compared (often manually) with the intended behavior. If the trace shows that the specified behavior does not correspond to the intended behavior, the specification is corrected and re-executed. When it is satisfied that the specified behavior is as intended, another "piece" of the behavior is added to the specification and it is validated again. The process iterates until the specification reaches the desired levels of accuracy and completeness, which are usually determined subjectively by the specifier.

This very general specification process can be performed with different degrees of rigor and formality. For small and highly abstract systems, an informal and free-style approach can efficiently produce quality specifications. With these systems, the specifier can often visualize the overall system structure and behavior and, therefore, can arrange the components and their interactions in his/her mind before deciding on a structure for the specification. It may even be possible to perform the specification and validation in one iteration. It seems that the critical factor is the ability of the specifier to hold an adequately complete picture of the system to be specified in his/her mind to be able to establish a specification structure that allows scenarios to be added incrementally without requiring major redesign of the specification.

Experience has shown that a specification whose structure is established before having a clear, stable and overall view of the features of the system is not robust to changes. When additional scenarios are incorporated into such specifications, major structural changes are often required.

Furthermore, changes require validation, and since the existing LOTOS tools that could be used to perform this validation still have limited capabilities, it is desirable to minimize the number of executions required. Thus, we want to have the specification as correct as possible the first time.

We believe that, in order to deal effectively systems of real size, the process has to be refined to a series of methodical steps. This thesis endeavors to formalize the process by identifying activities that are typically undertaken and formalizing as many of these activities as possible. Our intend is to produce a process that would allow even the most inexperience specifier to produce quality specifications with reasonable efficiency. A systematic approach, driven by the concepts of objects and scenarios, is described. An in-depth description of such a process would occupy many volumes and is beyond the scope of this thesis. We will describe only the coarse-grain process that was used to specify GSM.

Figure 13 illustrates the process, which is iterative and starts with informal system descriptions. The thick arrows represent the main paths of this analysis and specification process. The thick *light* path illustrates the analysis and modeling of the system's architecture and the thick *dark* path represents the analysis and modeling of its behavior. The thinner arrows indicate that execution of the LOTOS specification may reveal defects, incompleteness or inconsistencies in the specification and may require to revisit the source (informal) descriptions or the intermediate models. The circular paths formed by the dark arrows (both the thick and thin ones) indicate that the analysis and modeling of behaviors are done in iterations. That is, the thin arrows also indicate the next iteration of behavior analysis and modeling. The light arrows do not form a circular path because the architectural aspect of the system is considered only once (before or at the same time as the first iteration of the behavioral analysis). The architecture of the system is captured by the Object Behavior Models which is used to structure the LOTOS specification. The specification structure is not meant to change once it is established and is expected to be able to permit incremental additions of behavior in subsequent iterations. In practice, subsequent iterations often reveal the need for minor (hopefully) modification or refinement to the architectural picture.

The circular paths formed by the arrows represent *macro*-iterations of the process, which are cycles of the process. Within each macro-iteration, there may be many *micro*-iterations, which are iterations between adjacent models. For clarity, micro-iterations are not illustrated in Figure 13. For example, many micro-iterations of scenario elicitation are required to derive the Scenario Model. If the micro-iterations were to be shown for this example, the 'Scenario Elicitation' arrow would have to be made bidirectional.

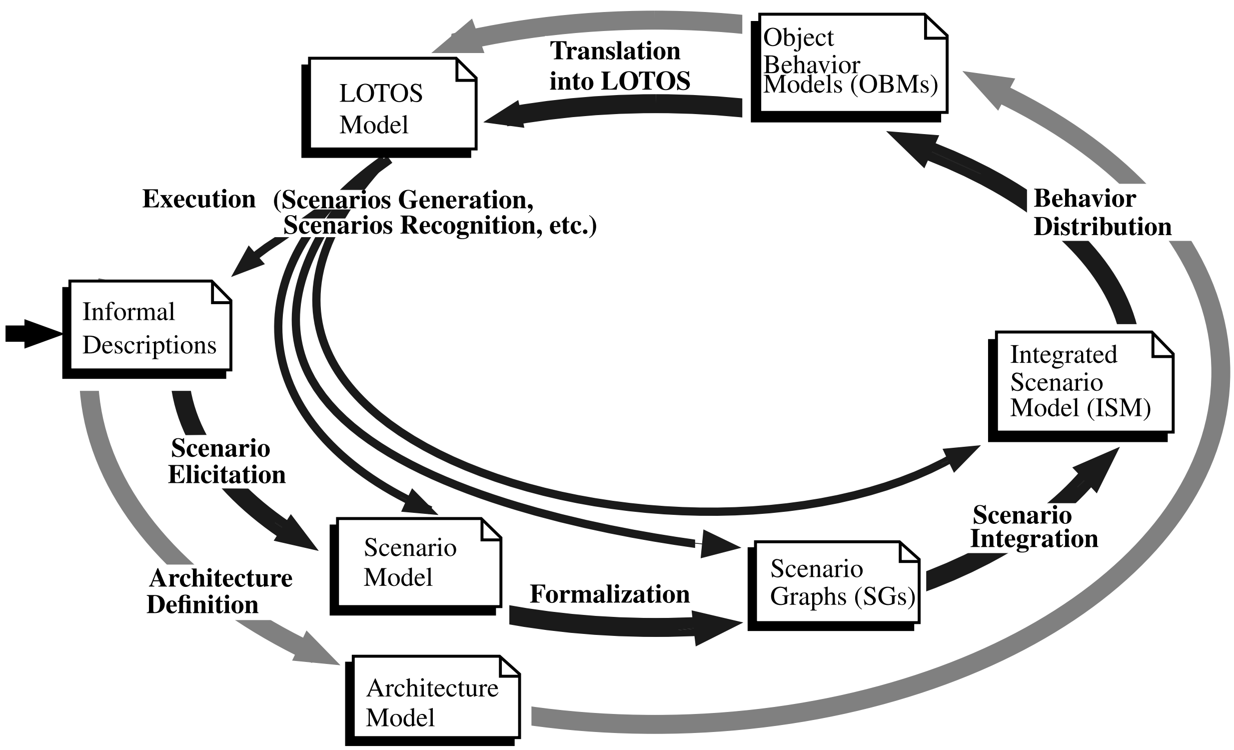


Figure 13 A systematic approach to specification development.

The process diagrammed in Figure 13 is based on the informal process discussed at the beginning of this section but with specific analysis artifacts and steps identified. It is divided into the following six steps:

1) Architecture Definition and Scenario Elicitation
2) Scenario Formalization
3) Scenario Integration
4) Behavior Distribution
5) Translation to LOTOS
6) Validation

The first step is similar to Jacobson's technique for developing requirements models. The main difference is in the abstraction level: SDA deals with systems at the requirements level (see Section 1.4) whereas here systems are viewed at the high design level.

The objective of the first step is to obtain a picture of the system's architecture and behavior by analyzing and describing them in terms of components and abstract scenarios, respectively. The results are the Architecture Model and the Scenario Model. The first model consists of a collection of objects and of their interconnections. The latter consists of a collection of scenarios, which conceptually forms a model of the system's behavior. Since the overall objective is to specify the system in LOTOS, this initial also includes consideration of LOTOS-related issues as well as those that help define the system. These issues are discussed in Section 4.2.1.

The second step seeks to formalize and prepare scenarios in the Scenario Model for integration and specification. Formalization, here, means elaborating scenarios to ensure that they are complete (i.e., possess all the information listed in Section 1.2.2), are at the same abstraction level, and use the same terminology. This step is based on the work of [RKW95]. Jacobson's SDA technique does not include this step, and use cases are merged at the informal level.

The products of formalization are rigorous scenarios descriptions with each such scenario represented by a labelled transition system called a Scenario Graph (SG). Scenario integration merges these SGs into a unified model, called the Integrated Scenario Model (ISM), which shows how different scenarios are related. The ISM is the unified "behavior fabric" of the system.

After the scenarios have been merged, the behaviors represented by the ISM are distributed among the objects, actors and system components, that contribute to these behaviors. The result of this operation are behavior models of individual objects called Object Behavior Models (OBMs). The Architecture Model identifies the objects that contribute to the behaviors of the ISM.

In the fifth step, a LOTOS specification (also called LOTOS model) is constructed from the Architecture Model and the OBMs. The Architecture Model with its objects and their interconnections determines what entities exist in the specification and how their representations are composed to form the structure of the specification. The OBMs determine the behaviors of these entities.

Translation of the Architecture Model and the OBMs into LOTOS may even be begun before all scenarios have been merged and distributed. The (partial) specification obtained can be used as a prototype of the system and can be executed to simulate the system. The simulation ‘plays’ out the behavior of the (partial) specification and may show how other scenarios can be incorporated. The execution also serves as validation (step 6) of the specified behavior against the intended behavior. Moreover, it may lead to discovery of new scenarios, reveal inconsistencies and incompleteness of existing scenarios or incorrect behavior in the integrated model. In Chapter 6, we discuss two scenario-based validation techniques: scenario generation and scenarios recognition. The latter involves using scenarios as test cases to determine if they are contained in a specification.

Although the process is described as starting with informal descriptions, in some cases the source information may already be in more formal notation. The fact that there is a distributed system to be specified implies the existence of a system design (at least, at the relatively abstract level); and system designs are rarely documented in narrative text alone. More often, their behaviors are described in formats such as scenarios and are represented by Time Sequence Diagrams, FSMs, etc., and their architectures denoted by some kind of diagrams. These notations may be close enough to LOTOS in terms of formality that some steps of the proposed process may be skipped. For instance, if the system’s behaviors are already in the form of scenarios, then scenarios elicitation is not needed.

4.2.1 Specification Issues

The early phase of the process includes consideration of issues that lead the specifier to precisely define the specification goals. It involves assessment of the general problem area, how the problem should be approached, the specific aspects of the system to be captured, and the expected uses of the resulting specification. These decisions are important because they affect the style and content, and therefore the usefulness, of the final specification as well as the difficulty of the process itself.

In general, the process should be guided by: 1) the purposes and expected uses of the specification, 2) the expressive power of LOTOS, and 3) the format of the available information. The effects of these three factors are not mutually exclusive, and together they define what properties of a system should be specified and how they should be represented.

In theory, the purpose of a specification should be the paramount factor in determining how it is produced, the choice of its style and what it will contain. But in practice, they are may be influenced more by pragmatic issues such as the expressiveness of specification language used, the availability of specification tools, the format of the source information, etc. For instance, even if the purpose of specification is to document all aspects of a system, this cannot be done because LOTOS is incapable of describing non-functional properties elegantly.

The purpose of a specification may not correspond to how it will be actually used. The latter is another factor that affects the specification process, and the style and content of the specification. As briefly alluded to in Section 3.3.1, some very important advantages associated with using LOTOS are related to the executability of its specifications. However, the abilities of existing tools and techniques are still limited, and full execution of some specifications is not possible or is too inefficient to be practical. Specifications, such as that in Figure 10, that describe infinite behaviors cannot be executed at all without limiting the behaviors to some finite size. In addition, some tools and techniques have preferences for certain specification structures and styles. For instance, interleave expressions tend to produce state explosion and are generally more difficult for tools and techniques that perform expansion to handle. These factors determine the *context* and *abstraction level* in which a system is specified as well as the *scope* and *style* of the resulting specification.

Context

One of the first activities to be performed is the identification of the context, or environment, in which the system is specified. This requires to decide what entities are parts of the system and what are outside of the system, i.e., we have to identify the entities that interact with the system. The identification of the context delineates the system and defines the environment in which it operates. It indicates the information that should be gathered to describe the system.

Abstraction Level

The abstraction level of a specification determines the amount of detail it contains. A chosen abstraction level must balance the ease of the specification process and the amount of detail required, while keeping in mind the objective of the specification. If a view that is too abstract is taken, the specification will not contain sufficient detail to be of much use. On the other hand, if a too detailed view is adopted, the specification process will be too laborious and thereby increasing the possibility of committing errors.

We believe that when specifying telecommunication systems such as GSM, a pure service-oriented view is be too abstract for the purpose of investigating *how* the system works. A specification at this level is not as useful as one that shows the interaction of the various parts of a system. These complex systems are usually built from components manufactured by different vendors, and specifications at the latter level of abstraction usually provide better pictures of how they interact.

Specification Scope

Specification scope determines which parts of a system and what aspect of these parts should be specified, i.e., the specifier must decide whether to specify all system properties or only some of them. This consideration must take into account the objective of the specification, what tools will be used, the information that is available, and the limitations of the specification language. Aspects that are not important or relevant in the use of the specification should be left out, and those that can not be represented in a natural way should also be omitted. We believe that if a language is not capable of describing elegantly system properties that are important, then a different, or an additional complementary language, should be used.

In the GSM example, the specifier can choose to specify every component and every layer of functionality or only some of these.

Specification Style

As discussed in Section 2.3, LOTOS specifications may be structured according to a number of styles, each accentuating different aspects of the system. Generally, strict adherence to one style is not possible, or even desirable. Because use of appropriate styles can clarify certain types of information, a particular style or combination of styles should be used whenever it presents the information more succinctly than other styles.

For example, for the purpose of documenting the system and showing how it works at a relatively abstract level, a specification of GSM should adopt the resource-oriented style (at least at the top level) because this style can best provide an architectural view of the system. In this style, the architectural components such as MSCs, HLRs, etc. can be modeled by LOTOS modules with well-defined interfaces.

4.2.2 Architecture Definition and Scenario Elicitation

This first step of the process aims to get an understanding of the system by eliciting and analyzing objects, their interconnections and interactions. The tasks in this step closely resemble those of SDA (Section 1.4.1), and consist of the following related activities:

1. Identification of actors and system architecture

2. Elicitation of scenarios

Figure 14 illustrates this step and the models it produces. In the figure, scenarios are drawn as "squiggly" lines for better visualization, but in practice they should be described in structured proses, to permit the creativity required for their identification.

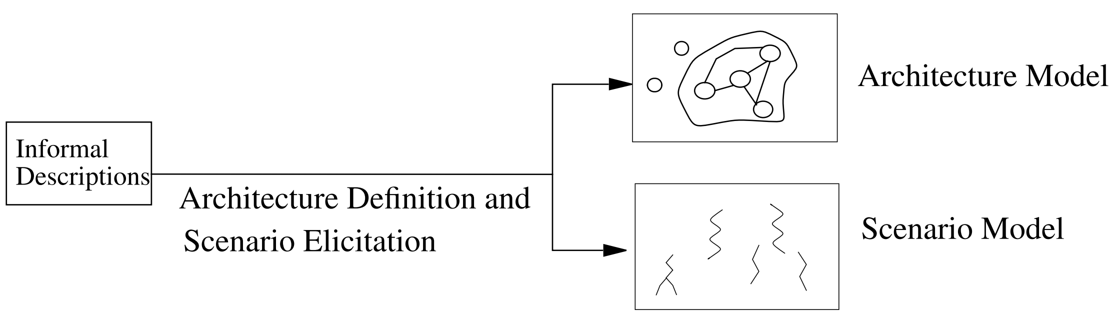


Figure 14 Architecture Definition and Scenario Elicitation.

Identification of Actors and System Architecture

The first task produces the Architecture Model which describes the architecture or structure of the system. This model consists of a set of objects and descriptions of their behaviors and interconnection topologies of components. This task is carried out by defining the context of the system. At the abstraction level considered in this thesis (that is, the high level design), this involves identifying users and system components. Generally, the question of how ‘deep’ in the system to look for components depends on all the factors discussed in Section 4.2.1. However, we do not recommend including too many layers as this will “clutter” the specification, and make it too big for tools to handle and decrease readability. Rather, if a detailed view of the system is desired, it is more beneficial to have several specifications. For instance, one specification to specify the top-level components and their interactions and others to specify separately the components.

Once the system components are known, their static relationships to each other are identified (see Section 1.2.1 for a discussion of the relations). These static relationships will be used to guide the composition of LOTOS modules that represent the components. Identification of a system’s architecture, thus, requires identification of its components and the static relationships among them.

The components are then analyzed to determine their behaviors. Since they are likely to have complex behaviors, only their general purposes (towards the functioning of the system as a whole) should be noted. It would not be justified to produce detailed descriptions as they have little use but require a lot of effort to accomplish. The descriptions of general responsibilities or roles can be used later to provide quick and empirical verifications that the components have been specified to the desired level of completeness. For example, the responsibility of the BSC component is to manage radio resources (such as allocating and deallocating radio channels) as well as to relay data and signals in MSC-MS communications. These responsibilities can be used later to provide a quick check on the LOTOS module that represents the BSC.

Once the system users have been identified, they are analyzed to determine potential roles they can take on. The roles are *actors*. Examples of actors in GSM are: a user who initiates a call, the recipient of a call, an external network that requests a connection to an MS for a mobile-terminated call, etc. Writing down brief descriptions for actors (similar to those for system components) helps to solidify understanding of their behaviors and is useful for uncovering behaviors of the system. Since actors exist outside of the system, only their interactions with the system need to be considered.

Elicitation of Scenarios

The second task identifies and describes internal and external system behaviors as a collection of scenarios, called the Scenario Model. This task is a creative and iterative one, often requiring several attempts before a stable picture of the system's behaviors is obtained. It can be expected that some scenarios will lead to further decompositions of objects or uncover new objects which will have to be incorporated into the Architecture Model.

Scenarios are obtained by examining the system services and the roles that users and system components can play. Each system service should lead to the production of at least one scenario. [Jac92] and [HSG95] provide more detailed descriptions on how to discover scenarios.

Conceptually, the collection of scenarios obtained represents the behavior of the system. But as a model, this collection suffers all the problems associated with representing a system by separate and partial models, see Section 1.2.3.

4.2.3 Scenario Formalization

The second step of the process analyses the scenarios to ensure that they are complete (with respect to Definition 1.2) and precisely stated, to prepare them for integration. This formalization activity produces a labelled transition system, called Scenario Graph (SG), for each scenario. Labelled transition systems (LTSs) provide a natural model for the semantic of LOTOS, and using them to denote scenarios facilitates transformation of the scenarios into LOTOS specifications. Figure 15 illustrates the scenario formalization process.

A SG is derived from a scenario by creating a labelled transition for each event in the scenario. After formalization, the SG of a scenario contains the exact behavior of the scenario, but more precise. In particular, it retains the ordering of events and pattern of flow.

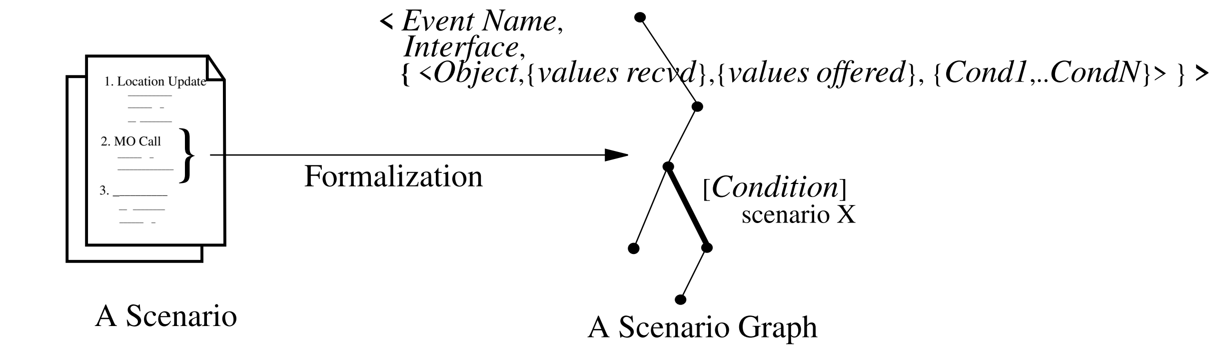


Figure 15 Scenario formalization.

The transition label is a 3-tuple that contains all the pieces of information associated with an event (see Definition 1.2). The format of the label, called *event label*, is inspired by LOTOS event structure, and is as follow:

< *Event Name*, *Interface*, { <*Object*,{*values recvd*},{*values offered*},{*Cond1*, ..,*CondN*}> } >

The first element (*Event Name*) in the label is the name of the event that may occur on the interface specified by the second element (*Interface*).

The third element is a set of 4-tuples, one for each object participating in the event. The first element in the tuple (*Object*) specifies the object involved. Since SGs are representations of abstract scenarios (as opposed to instances of scenarios), it is not the specific identity of an object that should be given but rather the component name, e.g., in the case of GSM, MSC or BSC are used instead of specific instances of these components. The second element is a set of data whose values will be obtained in the occurrence of the event, and the third element is the set of values that the object offers in the event. The last element is a set of conditions that must be satisfied in order for the object to be able to participate in the event. These conditions may apply on the *Event Name*, on values the object received in previous events, or on values it can potentially receive in the current event, or any combinations of these.

Like scenarios, SGs may also refer to other SGs and their transitions may be labelled with scenario names instead of event labels. Such labels may also be predicated by conditions that determine whether or not the named scenario may occur. The occurrence of a scenario that contains reference to another scenario is similar to procedure calls in programming. At the point where there is a reference to another scenario, control is transferred to that scenario. At the end of the ‘called’ scenario, control returns to the ‘calling’ scenario.

Figure 16 shows the scenario graph for the mobile-originated call scenario given in Figure 3. The darkened transitions represents ‘calls’ to other scenarios. Note that this representation of the mobile-originated call scenario does not contain the ambiguities and incompleteness (with respect to Definition 1.2) present in Figure 3.

The objective of producing scenario graphs is to force the specifier to defined completely the events in the scenarios (with respect to Definition 2). The graphs are a *means* to achieving this goal and not the goal itself. Indeed, other notation can be used in lieu of graphs but it was found that graphs are fast and easy to draw and they provide quick and easy visualization of scenarios.

Scenario graphs also bridge the formality gap between scenarios and LOTOS expressions. Compared to scenario, SGs are more formal because they specify events more precisely and, thus, they are easier to translate into LOTOS. They give a more formal interpretation of scenario execution (Section 1.2.2) by associating the occurrence of an event in a scenario to a state change in the graph.

Jacobson’s *uses* relation can be applied as part of formalization. This relation permits common segments in different scenarios to be extracted and specified just once, thus giving more modular and robust specifications. Any change performed on the common parts will affect all scenarios that *use* them. Note that this relation does not integrate scenarios in the sense that it connects them to form larger behaviors.

In conclusion, the formalization process has the following benefits:

- It allows scenarios to be described more formally but without concerns of LOTOS details.
- It forces the specifier to analyze scenarios in terms of the kinds of information given in Definition 1.1 and Definition 1.2, thereby producing more complete description of scenarios.

- Inconsistencies in a scenario may be revealed by the analysis.
- It may reveal and clarify ambiguities.
- It may reveal missing and superfluous information in scenario descriptions.
- It ensures that scenarios are described at a uniform abstraction level. For example, for a network system it ensures that all scenarios are described from a distributed view or from an end-to-end view.
- It ensures the consistent terminologies are used in the informal descriptions. The homogenization of terminology is important since a number of different people may have been involved in deriving the scenarios. Having complete and concise scenarios and uniform terminologies will ease the integration of the scenarios later on.
- There is strong and obvious traceability between the scenario graphs and the Scenario Model.

Like the set of scenarios from which it is produced, a set of SGs also forms a conceptual model of the system's behavior.

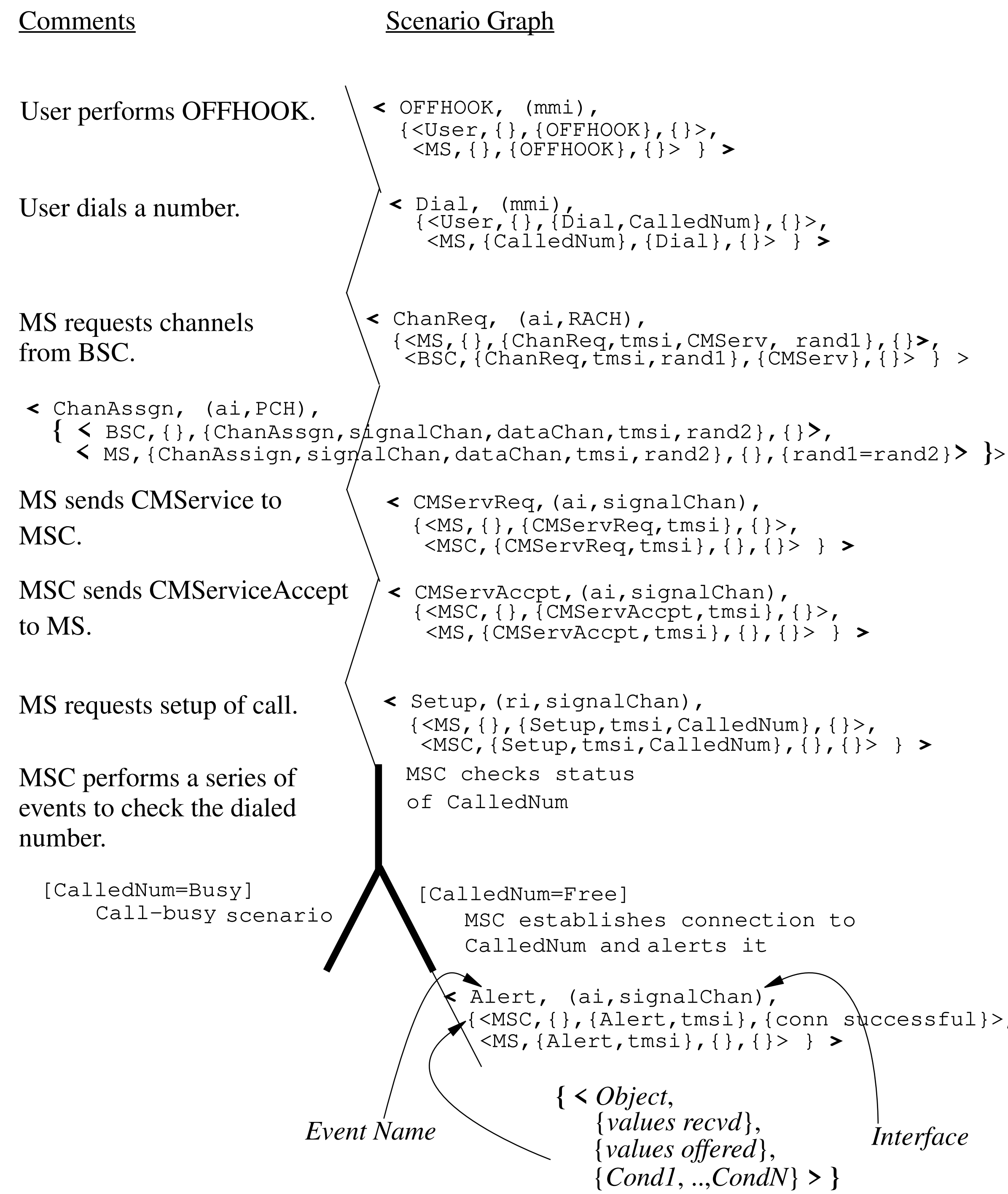


Figure 16 Scenario graph (SG) for the mobile-originated call scenario in Figure 3.

4.2.4 Integration of Scenarios

In scenario integration, we amalgamate the *partial* system models represented by SGs into one unified model. The result of this task is the Integrated Scenario Model (ISM), which is also a labelled transition system (LTS). The ISM denotes the ‘connected’ or integrated behavior of the system starting from some well-defined *idle system state*, represented by the root of the LTS. It is the “behavior fabric” for the whole system, i.e., it shows how the different scenarios of the system behavior are interwoven. It denotes all information described by the individual SGs, as well as new information such as the starting and terminating conditions of the scenarios. Figure 17a illustrates what an ISM may look like theoretically.

According to [RKW95], the behaviors of a systems starts from an *idle initial state* and eventually ends in a *terminate state* from which all the behaviors are once again possible, i.e., the initial and terminate states are identical. This highly simplified and abstract notion of system behavior is illustrated by Figure 17b. We believe that for most systems, especially large and complex ones like GSM and at the abstraction level considered in this thesis, such terminate states exist only in principle and they are difficult to define because they depend on many system variables. It is more realistic and practical to consider system behaviors as having multiple terminate states that may not be identical to the initial state. We are of the opinion that in the majority of cases, system behaviors are more like Figure 17a than Figure 17b.

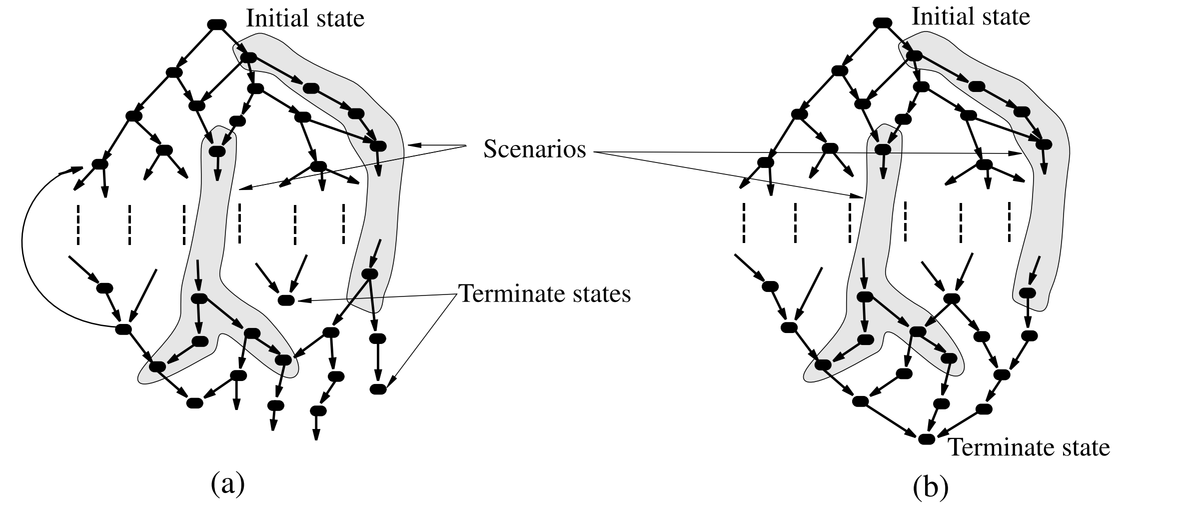


Figure 17 Two views of the Integrated Scenario Model (ISM).

It is important to empathize that the goal of scenario integration is to relate the scenarios one to another to form a unified behavior fabric for the system. The ISM is a visible representation of the conceptual behavior fabric. It is a means of the integration and not the end in itself. That is, integration does not require actually drawing graphs that look like those in Figure 17.

Integration of scenarios is the crux of the problem of producing a unified LOTOS model from scenarios. The feasibility of scenario integration is the observation that the scenarios are parts of description of the same system, and therefore (in principle) they could be mutually 'connected'.

Scenario integration is similar to the game of assembling jigsaw pieces to form a unified picture. In this analogy, scenarios are like the jigsaw pieces, the ISM is analogous to the picture that is obtained when all the jigsaw pieces have been fitted together, and the initial system state (the root of the ISM) is the very first jigsaw piece that makes up the assembled picture. For both processes, hard-and-fast step-by-step procedures for constructing the pictures cannot be given because whether a piece could be fitted into the (partial) picture depends on the shape or semantic of that piece.

This thesis describes a heuristic for scenario integration. The heuristic expands Jacobson's *extends* relation, which is a concept that basically states that a use case may be extended by the insertion of other use cases. This insertion operation is restrictive because it depends on scenarios being complete courses of events that describe the occurrence of some functionality (see Section 1.4.1). This thesis lists and discusses the possible relationships between scenarios which would allow one scenario to be meaningfully inserted into another scenario. The idea of scenario extension is related to scenario integration. When a set of scenarios are integrated, the resulting scenario is often an extension of each of the scenarios in the set, unless they are all identical. Note that the extension concept used here is not the same as the extension concept in LOTOS [BSC87].

The UORE process of [RKW95] gives a technique for integrating use cases (see Section 1.4.2). But UORE takes an over-simplified picture of behaviors by assuming that only one actor can be involved in a use case. Moreover, it produces a model that is still made up of many separate smaller models. [IYK90] attempts to extend a design by adding a LOTOS specification to another specification (see Section 1.4.3). The technique is concerned only with behavioral extension, the structural aspects of the system are not considered. Furthermore, it addresses the problem only partially because it assumes that the two partial specifications to be merged can be combined. [Rud92] is concerned mainly with re-using specifications in an OO environment. It tries to specify behaviors of new objects by inheriting existing behaviors. This approach also tries to preserve the structure of the existing behavior specification.

The problem addressed by this thesis is more general; the goal is to combine a *collection* of scenarios. We do not assume that *any* two scenarios can be merged because scenarios may have to be integrated in specific orders. The situation is similar to the jigsaw game; in order to be able to insert a jigsaw piece into the already assembled picture, at least one of the pieces to which that piece can be fitted must already be part of the assembled picture. Thus, the activity of scenarios integration consists of determining whether two scenarios could be integrated and how they can be merged, i.e., by what relationships are the scenarios related, if they are related. Since scenarios are merged at the semi-formal level (the SG level) and translation to the final notation (i.e., LOTOS) is done after the integration, the architectural properties of the system can be ignored in the integration process.

To see that the order of combining scenarios is important, consider the three GSM scenarios: Network Access (Section 2.4.2), Location Update and MT Call (Section 2.4.4). Recall that in the Phase 1 Recommendations an MS can invoke only one service per connection session. A connection between a mobile station and a PLMN is established by the Network Access scenario. Location Update and MT Call are services that could be invoked after a connection has been made. Therefore, two obvious integration are the concatenation of the Location Update scenario to the Network Access scenario and the concatenation of the MT Call scenario to the Network Access scenario. The MT Call scenario can also be concatenated to the Location Update scenario, but only after the Network Access scenario has been to appended to the latter. In this example, we assume that connection tear-down is part of the Location Update and MT call scenarios.

In practice, it is impossible to obtain all the behaviors of a system by describing individual scenarios. This means that it may not be possible to relate some scenarios to others because that part to which it can join is missing. Such a situation may lead the specifier to discover missing scenarios when attempting to integrate them.

More precisely, the integration process involves the following related tasks:

1) Identification of an idle initial system state.
2) Determination of relationships between scenarios.
3) Integration of SGs according to the determined relationships.

Identification of an idle initial system state

The process starts by identifying a well-defined *idle initial state* of the system to serve as a starting point from which behaviors start. To this initial state, scenarios are iteratively integrated.

In general, there are many system states that could serve as the initial state, and what states are available for this role depends on what is considered “the system”. In the case of GSM, if the “system” consists of a PLMN and mobile stations, then some possible initial states may be the following situations: 1) the PLMN has no subscriber mobile stations of its own and no MS from other PLMNs has registered (i.e., all the HLRs and VLRs are empty), 2) the PLMN has subscribers but no MS is currently accessing the network, and an MS has just been activated, and 3) the PLMN has subscribers but none has registered its location (i.e., there is subscription information in the HLR but none in any of its VLRs).

Determination of relationships between scenarios

After selecting an idle initial state, the ISM is built by adding one scenario at a time. Generally only some scenarios start from this initial state, others are branches or segments of other scenarios. Figure 17 shows what a set of SGs may look like (ideally) when combined. In practice, the ISM can be more compactly drawn as a collection of arrows connecting the names of scenarios that are related. Figure 17 would be what the integrated scenarios actually look like if the arrows and scenarios names were substituted with actual SGs.

In order to integrate the SGs, the relationships between the scenarios must first be determined, and these relationships determine how they will be merged. Determination of whether and how two scenarios are related requires comparison of different combinations of these scenarios against possible system behaviors. Unfortunately it does not seem possible to define this operation in terms of syntactic or structural characteristics of the SGs, and therefore it is not possible to give formal procedures to precisely integrate scenarios. The reasons for this is because scenarios come in infinite varieties; any causally-ordered flow of events qualifies as a scenario. This is the main reason why this thesis takes a pragmatic approach to scenario integration.

However, the determination of relationships between scenarios can be aided by LOTOS concepts because, after all, SGs are just LOTOS behavior expressions in the monolithic style. Thus, any LOTOS composition operations that could be applied to LOTOS behavior expressions could also be applied to SGs. Some of the relationships that may exist among scenarios are:

- Alternative: a scenario may be an alternative courses of other scenarios. For example, an MS cannot perform the MO Call and MT Call scenarios at the same time, so these scenarios are alternative behaviors. Alternative scenarios are composed in LOTOS by the choice operator.
- Subordinate: a scenario may be a subordinate of another scenario (let us call the latter a parent scenario). There are two kinds of subordinate courses: those that are executed every

time the parent scenario is executed, and those that are executed only if conditions (if they exist) are met. A subordinate course may be inserted at the beginning, at some point along the path, or at the end of the parent scenario. Often, the SGs that are most amenable to insertion into other SGs are those that describe small, general-purpose functions that could be reused in many SGs. An example of such scenarios in GSM is the authentication procedure that can be invoked in MO call, MT call, Location Update and other scenarios. Subordinate scenarios may be represented in LOTOS by process instantiation.

- Interrupt: scenarios may represent behaviors that interrupt other scenarios. This relationship is similar to a situation where an interrupt service routine (ISR) interrupts the normal flow of execution in response to an exception. Whether or not control returns to the interrupted scenario after the interrupting scenario finished is a matter of detail and depends on the behaviors involved. For example, a scenario that depicts how the network performs the MO Call scenario may be interrupted by a scenario that describes the behavior triggered by an onhook event. Interrupt scenarios may be expressed in LOTOS by the disable operator.
- Synchronization: distributed systems often have concurrent behaviors. This means that a set of scenarios that describe such systems can occur simultaneously, and these parallel scenarios may communicate. This can be represented as synchronization, using the LOTOS parallel composition operators. Note that the synchronization relation can exist between scenarios or between different instances of the same scenarios. In fact, at the level of abstraction considered in this thesis there are four levels of concurrency: 1) concurrency among scenarios, 2) concurrency among instances of a scenario, 3) concurrency among objects, and 4) concurrency in the behavior of one object. Concurrency among objects and in the behavior of an object is defined implicitly in scenario descriptions. For example, two objects that are involved in two concurrent scenarios (one in each scenario) can operate in parallel; and an object that is involved in two scenarios that can be executed in parallel is a concurrent object. Since their descriptions are implicit in the scenario descriptions, consideration of these forms of concurrency can be delayed until the behav-

iors of the scenarios have been distributed among the objects involved, i.e., until all the behaviors of an object in different scenarios are brought together (see Section 4.2.5).

Since scenarios may overlap and/or intersect each other, the above relationships may exist between whole scenarios, segments of different scenarios, or between some whole scenarios and segments of others. Also, these relationships are analogous to the ones proposed by [Amy94] for use case maps.

As stated previously, there are no syntactic or structural features of scenarios that could be utilized in an automated way to relate them to each other. However, there is one characteristic of scenarios that may indicate that they are related. This characteristic is *common system state* [RKW95]. This is a system state from which execution may continue in different ways, or in other words, a node in the SG where it may branch. For instance, a common state in GSM is that state that is reached just after dedicated channels are granted. This state exists in several scenarios. In this state, the mobile station can performed MO call, MT call, Location Update, etc.

Some common states may be found by considering nodes in different SGs that have the same pre-sequence. In practice, since scenarios do not start from the same system state, the majority of common states can not be found this way. More often, the specifier has to analyze the semantics of the scenarios to determine common states and other types of relationships.

Integration of Scenario Graphs

The relations defined in the previous section are not class relations (in the OO sense), and therefore they may not apply to all instances of a scenario. (Recall that scenarios are defined as class concepts.) This implies that when scenarios are connected to each other, the points of connection may need to be predicated by conditions such that when the combined scenario is executed these conditions determine whether or not the relation applies to that particular execution. For example, consider the two scenarios MO call and User Authentication. The former describes the connection establishment procedure for MO calls, and the latter describes how users are authenticated. During call setup, a PLMN may or may not authenticate a user depending on the information the MS is able to supply and on the policy of the network. The authentication procedure is thus an optional subordinate course of the MO call procedure, and a predicate is needed to state under what conditions it is executed.

Because scenarios are class concepts, we do not have to be concerned with getting scenarios that do not make sense when these scenarios are formed by the integration of other scenarios. To understand what is meant by this, suppose that scenarios are not class concepts but instead they describe behaviors of specific objects (i.e., instances of objects). Suppose also that two such scenarios are, say, the MO call scenario and the MT call scenario. The MO call scenario shows an object instance, say John, making a telephone call; at the same time, the MT call scenario describes John (the same John) receiving a telephone call. Note that these scenarios form a valid set that describes a system's behavior because they are separate scenarios. The details of the scenarios are not important in this counter-example. Now, since MT call is the second half of MO call, it is intuitive to integrate (append) the latter to the former. The complete call scenario that results from such an integration would not be what was intended. But if the two scenarios refer to the object *person* instead of a specific instance of *person* (John)*,* there would be no semantic difficulty in the scenario resulting from their integration because different instances of person can be used to instantiate *person*.

Although scenario integration is discussed as a separate step that precedes translation to LOTOS, in practice not all of the scenarios have to be integrated before specification can begin. In fact, it may not be possible to integrate all the scenarios before translation into LOTOS because relationships between some scenarios may not be found. The integration of scenarios may be carried out at the same time that they are formally specified. But it is recommended that scenarios that describe the main behaviors of the system be integrated early to rapidly build a stable picture of the system's behaviors. In this way, a LOTOS specification of the main behaviors can be developed early, and execution of this specification may reveal how other scenarios could be incorporated.

A number of cycles of integration, specification and execution may be required (as shown in Figure 13) in order to integrate all or most of the scenarios. The intermediate and partial specifications are executed with the aids of tools. The execution can also be used to check that the integration and the translation were done correctly. This is analogous to integration testing in system development. New scenarios may also be uncovered by the execution.

4.2.5 Distribution of Behavior

As part of the process of developing a resource-oriented specification, the behaviors represented by the ISM is distributed among the objects involved. In this distribution, a behavioral model, called the Object Behavior Model (OBM), is derived from the ISM for every object that participates in the ISM. These objects should exist in the Architecture Model; if there is any discrepancy between the objects in the ISM and those in the Architecture module, it should be resolved at this point. Thus, this step consolidates the structural and behavioral models, and moves the system model closer to the LOTOS model.

The OBMs are labelled transition systems that are derived from the ISM by partitioning each ISM event (i.e., transition label) into OBM events (transition labels) of the objects that participate in that event. The labels of OBMS have the following format:

< *Event Name*, *Interface*, {*values recvd*},{*values given*},{*Cond1*, ..,*CondN*} **>**

The meanings of the fields are the same as those of the labels in the ISM (see Section 4.2.3). *Event Name* is the name of the event in which the object participates, on the interface specified by *Interface*. The set {*values recvd*} is the set of data whose values will be obtained in the occurrence of the event. The set {*values offered*} is the set of values that the object offers in the event. The last element of the label is a set of conditions that must be satisfied in order for the object to be able to participate in the event. This set may be only a subset of the conditions found in the corresponding ISM label, it may even be empty. For example, the condition {‘CalledNum=Free’} in the mobile-originated call scenario of Figure 16 is evaluated by the network, and therefore will not appear in the object behavior model for a mobile station (as is shown in Figure 19). Like the conditions of an ISM label, they may apply on *Event Name*, {*values recvd*}, {*values offered*}, or any combinations of these. Note that the name of an object does not have to be included in the OMB label because an OMB belongs to only one object.

All the elements in the label of an OBM transition are copied unchanged from the elements by the same name of the corresponding ISM transition except possibly the *Interface* field (this exception will be explained later). The example in Figure 18 illustrates the partitioning of the sixth event in the mobile-originated call scenario in Figure 16. The event involves the mobile station (MS) and the mobile switching center (MSC). Therefore the OBM for the MS and the MSC should contain a version of this event.

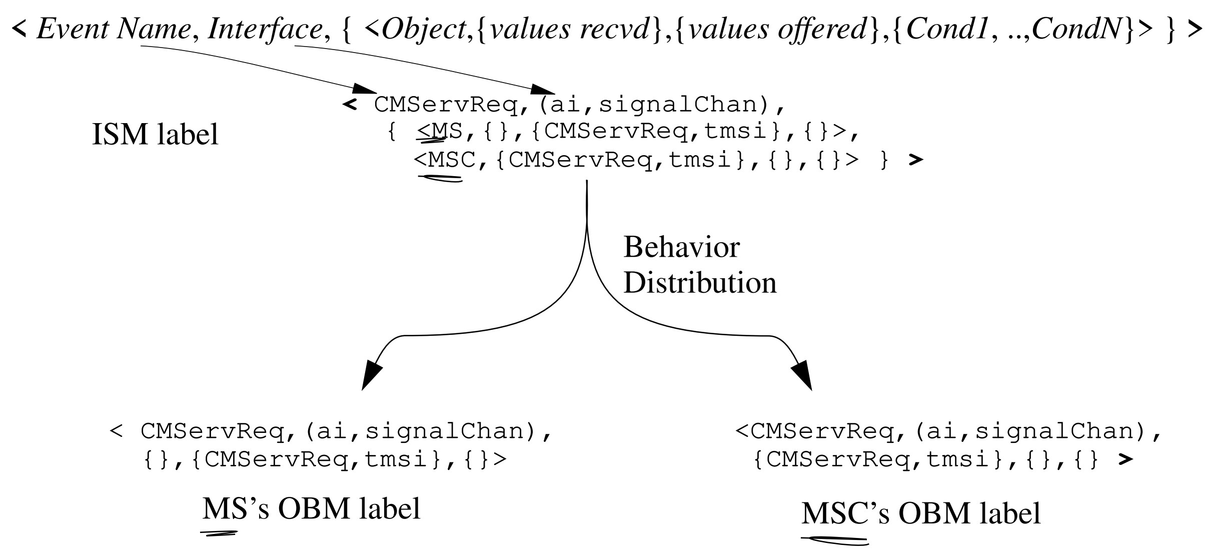


Figure 18 Behavior Distribution.

Figure 19 shows the complete OBMs of the MS and MSC derived from the scenario in Figure 3. These LTSs show the behaviors of the MS and the MSC in the MO call scenario. Dashed lines indicate labels that were derived from the same ISM event. Note that analogous to a scenario, an OBM of an object may also referred to 'scenarios' of that object's behavior (shown as highlighted transitions).

This partitioning of ISM events into OBM events could be thought of as distributing the responsibility for that event among the responsible objects. It is important to ensure that the specification structure permits the event to occur *viz*. there is no synchronization problem among the objects participating in the event. Otherwise, this is a defect. This mainly requires ensuring that the objects are composed with the appropriate operators and that matching structures are used in the bodies of the LOTOS processes representing the objects. This implies that the specification structure (specifically the gates and processes) must be defined before or at the same time that the OBMS are derived. But it does not mean that the objects' dynamic relations (behavior) alone determine how the LOTOS processes are composed. Rather, the static relations among the objects determine which processes should synchronize with each other; the behaviors of the objects determine whether the *Interface* element of the ISM label has to be refined (see Section 3.5.3 and Section 5.3.4) to ‘smaller’ gates or whether completely new extra gates are needed in order to ensure the required synchronizations. The derivation of OBMs, thus, also determines the LOTOS event structure that will appear in the specification.

The events in the OBMs retain their relative ordering and relationships. This means that if an object is involved in more than one branch of an ISM, then its derived OBM should retain these alternative behaviors. These OBMs represent the objects’ individual contributions in the ISM of the whole system. This distribution, in effect, partitions the system behaviors, the ISM, into behaviors of individual objects. The individual behaviors of the objects can then be specified in LOTOS.

The total behaviors of each object is now brought together and localized in OBMs (in the ISM, the behaviors of an object is dispersed among behaviors of other objects). At this point, the descriptions of object’s general responsibility (Section 4.2.2) can be used to quickly verify that intended the behaviors of the object have been included. This could also lead to discovery of new scenarios. Also, concurrency issues between objects and within each object can be addressed (see "Determination of relationships between scenarios" on page 70).

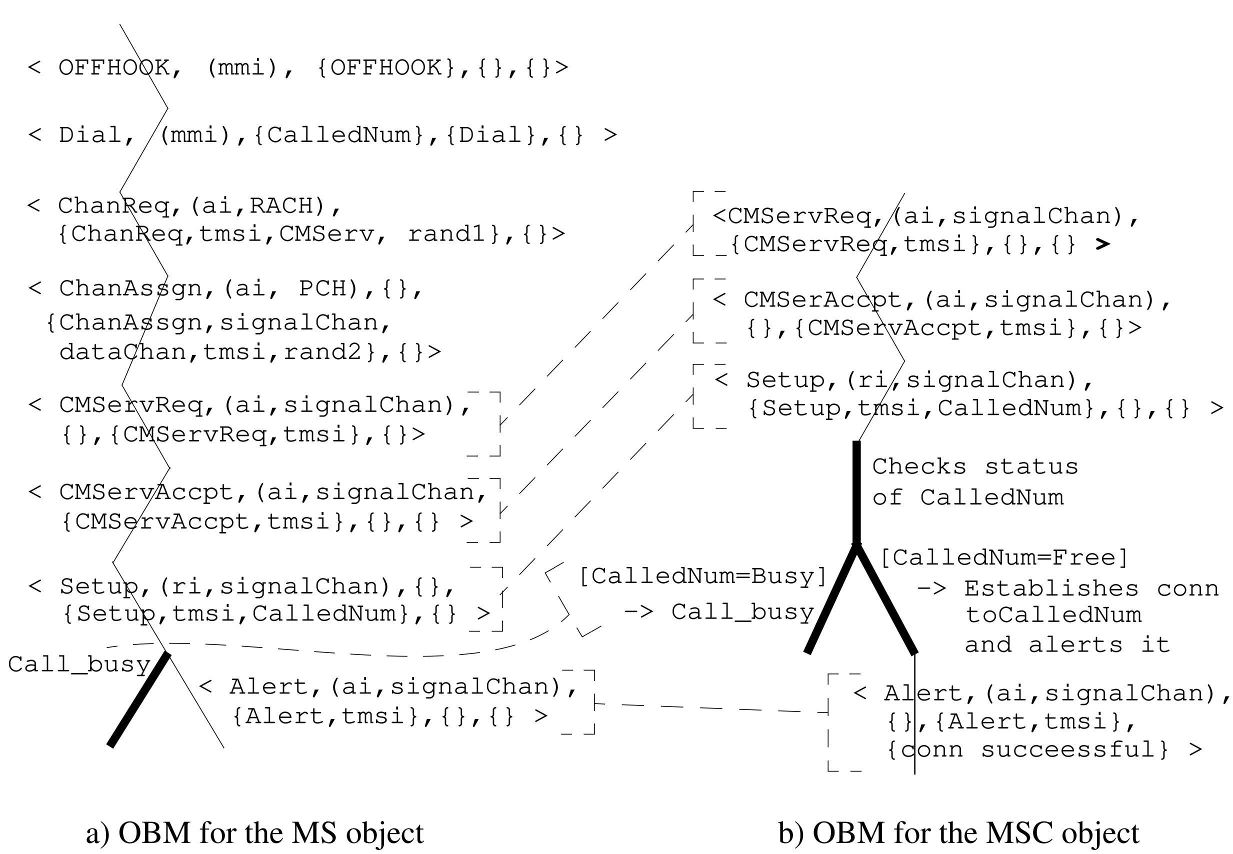


Figure 19 Object Behavior Models (OBMs) for the MS and MSC objects derived from the mobile-originated call scenario in Figure 3.

4.2.6 Translation to LOTOS

As discussed in Section 3.4.4, a top-down approach is suitable for developing resource-oriented specifications. This approach starts with definition of the top-level process structure, then behavior is 'fill in' over this structure. Data and data structures are specified whenever they are needed, their specification should not precede the specification of system structure or behavior.

First, the boundary of the system is specified. In order to describe the external behavior of a system, we have to also specify the behaviors of its users. In LOTOS, there are several ways to describe a system's users, leading to different top-level structures of the specification.

- They may be modeled implicitly; the system’s interactions with its users occur via external gates, i.e., gates that are not hidden. In this case, the users are parts of the environment. This approach is unsuitable when there are many users or when they interact with the system in complex ways because the environment or the specification user has to supply the events to synchronize with the specification.
- The users are modeled explicitly by top-level LOTOS processes. In this case, the system’s interactions with its users are represented by synchronization on (first-level) hidden gates. The processes representing the users need only describe their behaviors, and should be written in any of the extensional styles (Section 3.4). The physical and/or logical connections between the users and the system determine how the processes are composed (i.e., they determine what operators should be used and which gates the processes should synchronize on) and give the topmost structure of the specification. This approach is suitable when the system users interact with the system in complex ways.

After defining the system boundary, the top-level system architecture can be specified. Architectural components are mapped to LOTOS processes. Their interconnections can be represented in two ways; they could be represented *implicitly* with gates and by the way the processes are composed, or *explicitly* with gates and ADTs as in [HaL92].

At this point, it may be required to decide whether to have a dynamically configurable topology of components or a fixed topology. A configurable topology is one which allows new components to be created dynamically. It is obtained by dynamic instantiation of LOTOS processes that represent nodes in the topology. The LOTOS technique described in Section 3.5.1, that uses embedded recursive process definition, can nicely represent a restricted class of dynamically configurable tree-like configurations. Application of this technique is shown in Section 5.3.3 where it is used in a specification that represents a dynamically configurable GSM network.

Object instances have identities that distinguish them from other instances of the same object. These identities are specified with ADTs and can be associated with the objects by including them in the parameter list of the corresponding LOTOS processes.

The result of the activities thus far is a skeleton process structure that mirrors actual system structure. The OBM of each object can then be used to define the bodies of the processes.

Different LOTOS event structures may be required. The formats of event structures should be based on the format of the OBMs' labels. The right balance between the number of *different* structures and the clarity and accuracy of the specification has to be used. Experience has shown that many specification faults are due to events that should synchronize but do not because of unmatched event structures.

Names of events and data must be modeled with the ADT component of LOTOS. Some data and data management structures that are not strictly mentioned in any scenario but are needed to manage information may have to be specified, e.g., structures to keep track of busy lines in the specification of a telephone system.

4.3 Deriving LOTOS Specifications from Technical Standards

In order to provide frameworks for building open systems, standards have to specify a wide range of technical issues, some at abstract levels and others at very detailed levels. Consequently, standards often contain vast amounts of information. This information are the constraints that an implementation must satisfy in order to ensure interoperability with conforming implementations.

However, despite copious information, standards may be incomplete in their coverage. This may seem contrary to the purpose of having standards but sometimes certain technical issues are left out because they are not required to enable openness. Incompleteness may also be due to unintentional oversight, or due to a lack of consensus among the parties responsible for defining a standard. On the other hands, standards may describe several different ways of implementing the same functions.

Thus, it is not unusual that standardization documents contain vast amount of information yet can still be incomplete, and possibly ambiguous. These characteristics make the derivation of LOTOS specifications from standards difficult as the specifier must sift the information to extract relevant descriptions. Specifically, the specifier must deal with the questions of what information in the standard to include in the specification, whether all permitted implementation options should be specified, and what to do when required descriptions are not defined by the standard. Dealing with these questions means considering the specification issues discussed in Section 4.2.1. That is, the objectives of the specification, the limitations of the language and the tools that will be used determine the answers to these questions.

It is widely accepted that analysis should deal only with abstract, implementation-independent descriptions. It is possible for LOTOS specifications to be quite abstract, if they are not required to be executable. But in order to build an executable model, it is necessary to specify execution mechanisms, which by necessity will involve mechanics. In this sense, an executable LOTOS specification may have to be less abstract and more implementation-oriented than prose descriptions found in the standards. However, standards usually already set a great deal of implementation constraints. Thus, it may even be unavoidable for specifications derived from standards not to contain some implementation details. We believe that such specifications should not be more abstract than the standards because implementation solutions have to satisfy the constraints set by the standard. We followed this belief in the specification of GSM.

CHAPTER 5

LOTOS Specification of GSM

5.1 Introduction

This chapter presents a LOTOS specification of GSM that was derived by the method given in Chapter 4. However, the actual construction process of the specification will not be discussed in detail because this would be extremely long and difficult to express. In fact, the specification was produced before the formulation of the method. That is, the method is a *posteriori* rationalization of the activities performed to derive the specification.

The specification was derived from a set of scenarios that was generated manually, a book containing informal high-level descriptions [MoP92], and from the GSM standardization document [ETSI92]. It consists of about 3000 lines of LOTOS code and comments, and resides in two files: one contains the ADT specification, and the other contains the behavior part. The data types specified in the ADT file are included in the behavior file as LOTOS libraries. The Unix 'make' utility was used to maintain and update the dependency between these two files.

The chapter is organized as follows. Section 5.2 discusses the issues that lead to identification of the specification goals. The specification of the GSM architecture is given in Section 5.3 and the specification of the behavior is given in Section 5.4. Section 5.5 presents a discussion on the ADTs used. In this chapter, thin italic *`font`* is used to denote verbatim extracts from the specification.

5.2 Specification Issues

According to the process in Chapter 4, some of the very first activities of producing a LOTOS specification are to identify specification goals, and to establish the abstraction level and the context (or environment) in which the system is viewed, and the specification style and scope (Section 4.2.1). Our main objective for wishing to specify GSM was to derive an executable model that also documents its structure and behaviors. Since GSM is a large and complex system, this goal had to be moderated by practical issues such as the ease of specification, the suitability of LOTOS, the readability of the specification that would result, and the limitation of tools that will be used. These practical considerations motivated the adoption of an abstraction level that focuses on the coarse-grained behaviors and structure of the system.

We recognized that large LOTOS specifications tend to be difficult to read and understand, and therefore wished to structure our specification in such a way as to maximize readability and comprehensibility. We found structuring along the physical and functional dimensions to be helpful in understanding distributed systems. Furthermore, since the standard and the other source documents describe GSM along these lines, we felt it was only natural to follow these structuring techniques in the specification. Hence, our (refined) goal was to capture the ordering of interactions between the Radio Resource (RR), Mobility Management (MM) and Communication Management (CM) entities of the GSM physical components. Specifically, we wished to specify interworking of the RR, MM and CM protocols: these protocols are RIL3-RR, RSM, BSSMAP, RIL3-MM, RIL3-CC, and MAP. The Transmission Layer was not specified, except for a few Protocol Data Units (PDUs) that are intricately tied to some RR, MM or CM functions. We also chose to specify the deployment (i.e., dynamic creation) of Public Land Mobile Network (PLMN) nodes and of mobile stations because LOTOS is particularly suitable for this type of specifications and because it would give the ability to investigate different network configurations.

In principle, the RR, MM and CM protocols could be completely described (with respect to the source documents) by specifying all the scenarios that use them. However, we believe that specifying all these protocols in their entirety is both undesirable and impractical. It is undesirable because the resulting specification would be too large for most tools to handle, and it is impractical because the amount of detail would be overwhelming. The benefits would not justify the effort, and the resulting specification would likely contain many defects. Moreover, it is unsure how completeness can be determined. As a result, our specification does not all possible scenarios. Those scenarios or parts of scenarios that are not included pertain to behaviors that rarely occur or that are not essential to understand the system at the chosen abstraction level. Thus, our specification scope was determined as much by the expected usefulness of the specification as by the practical concern of being able to produce it.

To facilitate understanding, the interworking of the protocols are associated with the service they realize. That is, the specification is structured in such a way that it shows the interworking of the protocols when a system service is invoked at the system boundary. The resource-oriented style (Section 3.4.4) was chosen for the highest level because it gives visual clues of the interconnections of the system's components. The specification of the components is in a combination of monolithic and state-oriented styles.

5.3 Specification of the GSM Architecture

The top-level structure of the specification is based on an Architecture Model (Section 4.2.2) of GSM, which is influenced by the two ways of viewing the system (Section 2.3 and Section 2.4). The physical view identifies GSM subsystems and the physical machines as objects in the Model. The subsystems are: PLMN, Network Switching Subsystem, Base Station Subsystems, and Operation Sub-System. The physical machines are: Mobile Switching Centers, Gateway MSCs, Home Location Registers, Visitor Location Registers, Base Station Controllers, Base Transceiver Stations, and Mobile Stations. It also determines the containment relationship between the subsystems and the machines, as well as the connectivity between the machines.

The functional view decomposes each physical machine into a CM, MM, and RR entity. Thus, the objects in the Architecture Model for GSM are the subsystems, the physical machines and their logical CM, MM and RR entities. The relationships in the Model are the containment relationships between the subsystems and the machines, and between the machines and their CM, MM and RR components, and the connect relationships between the CM, MM, or RR components of different machines.

In the resource-oriented specification style, these objects are modeled by LOTOS processes. The containment relationships are represented by process embedding, and the connects relationship are modeled by process synchronization. The structure of the whole specification is given later in Figure 21.

The top-level architecture of the specified system is illustrated in Figure 20, and its corresponding specification is shown in Section 21. The system consists of a Public Land Mobile Network (PLMN), an Operation Sub-System (OSS) and an arbitrary number of Mobile Stations. This system interacts with its environment at four points.The three subsystems are discussed below and their specifications are given in subsequent sections.

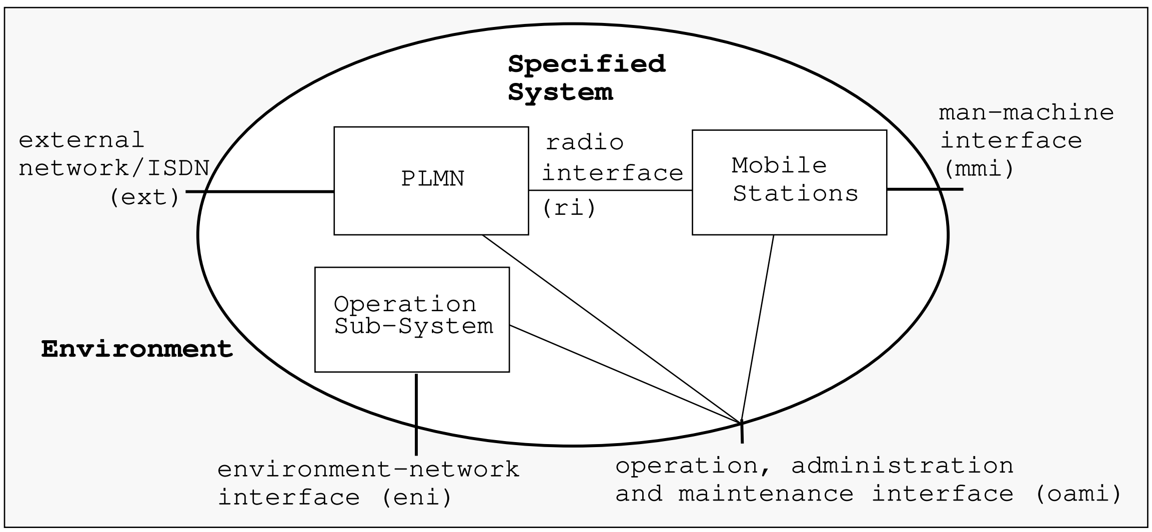


Figure 20 A pictorial view of the specified system.

The Public Land Mobile Network (PLMN) subsystem interacts with the external network at the `ext` interface. In the real world, different protocols are used at the `ext` interface for different external networks. In our specification, the ISDN network was taken to be the only external network and (a portion of) its call management protocols, known as ISUP[1], is specified. ISDN was chosen because descriptions of its interactions with a PLMN was readily available [MoP92]. Other networks, such as PSTN or PSPDN, could have served equally well to illustrate the PLMN's communication with external networks. In fact, *all* networks that

1. ISUP = Integrated Service (Data Network) User Part

could interwork with PLMNs could have been specified. But this would have just increased the complexity of the specification with few additional benefits because the main focus of the specification were the radio interface and the interfaces between the PLMN components and not so much the PLMN's external interfaces.

The OSS, which implements some functionalities of the OAM layer (Section 2.4.5), interacts with its users (network operators and administrators) at the operation, administration and maintenance interface, *oami*. The rationale for specifying OSS is because of the need to have LOTOS constraints that perform housekeeping duties on the creation of PLMN components and MSs, rather than a desire to specify the operation and administration of PLMNs. The gate *eni* represents the interface at which a user may control the deployment of PLMN objects and mobile stations. The role of this gate is made clear in Section 5.3.1.

The Mobile System subsystem represents an arbitrary number of mobile stations. Each MS interacts with its human user at the man-machine interface *mmi*. The main reason for including mobile stations as part of the system and not make them part of the environment was because they have complex behaviors. The radio interface is probably the most interesting and complex interface in GSM. To consider mobile stations as parts of the environment would mean that the environment (i.e., the specification user) has to supply events that are expected at the radio interface when the specification is executed. This is likely to be a difficult task that even GSM experts would have difficulty with because many protocols exist at this interface. Consequently, mobile stations were specified as components of the specified system and the radio interface *ri* became internalized.

5.3.1 Specification of the Operation Sub-Subsystem

The GSM standard defines few requirements for the operation, administration and maintenance of the PLMN. The major reason for this is that these issues do not affect system interoperability in terms of providing smooth and transparent services across PLMNs to the users.

Two of the functions of PLMN operation, administration and maintenance are the deployment of new network nodes to increase coverage area, and the subscription of new users. Process *OSS* (Figure 22) controls the creation of NSS and BSS machines and mobile stations, as well as the insertion of identities of newly created MSs into the PLMN's master database, the HLR. A

```
specification GSM[eni,ext,mmi,oami]: noexit

    ...(*data type definition*)

    hide ri in
    behavior
      (PLMN[ext,oami,ri]
      |[ri]|
      MobileStations[mmi,oami,ri]({} of MSISDNSet)
      )
      |[oami]|
      OSS[eni,oami](true)

    where
        ...
    endspec
```

Figure 21 Top-level structure of the LOTOS specification of GSM.

specific machine is created in processes *PLMN* and *MobileStations* when the environment supplies a value to denote its identity. All such events synchronize with *OSS* at the operation, administration and maintenance interface *oami*. This synchronization enables *OSS* to control the deployment of network elements. For instance, when the variable *opn_session* is true MSs and PLMN components may be created (lines 2-17). The value of *opn_session* can be changed at the environment-network interface *eni*. This ability to control the creation of objects is useful when the specification is executed as it limits the number of events that are available at a given state and thus allows attention to focus on more interesting events. Without this control mechanism, creation of components would be available all the time.

OSS also specifies the enrolment of users in the network. When a user comes to the network administrator to subscribe for services, the administrator records his subscription information in the subscriber database (the HLR). In the specification, this is represented by the creation of a new MS (line 11) followed by the synchronization of *OSS* with the LOTOS process that represents the HLR. This synchronization represents the insertion of a record (line 12) of the newly created MS (i.e., its MSISDN, IMSI, and MSC location) into the HLR.

```
process OSS[eni,oami](opn_session:Bool):noexit:=
  [opn_session] ->
    (
      (*allow creation of MSC by sychronizing with MSCVLR*)
      (*See Figure 24, line 17*)
      oami ?newmsc:MSCId;
      OSS[eni,oami](opn_session)
      []
      ...                                  (*creation of BSCs, BTSs *)
      []
      oami ?dn:MSISDN ?im:IMSI;                       (*creation of new MS*)
      oami  !INSERT_SUBSCRIPTION  !hlrRec(im,dn,noMsc);  (*insert  rec  in HLR*)
      OSS[eni,oami](opn_session)
      []
      eni !close_OSS_session;         (*disallow creation of components*)
      OSS[eni,oami](false)
    )
    []
    [not(opn_session)] ->
      eni !open_OSS_session;            (*allow creation of components*)
      OSS[eni,oami](true)
```

Figure 22 Specification of the Operation Sub-System (OSS).

5.3.2 Specification of Mobile Stations

The process *MobileStations* in Figure 23 allows an arbitrary number of mutually independent mobile stations to be dynamically created. It utilizes the process composition structure presented in Section 3.5.1. Initially, the process is initialized with an empty set *{}* for the parameter *UsedMSISDNs* to signify that no mobile station exists. A new mobile station, identified by a unique directory number, may then be created (line 3 in Figure 23) by instantiating *MStation* with an International Mobile Subscriber Identity (IMSI), Temporary Mobile Subscriber Identity (TMSI) and mobile station ISDN number (MSISDN). The MSISDN number (also known as the directory number) has to be supplied by the environment (user) at the gate *oami* (line 2), and the IMSI is derived from this number by a function specified in ADT (in the real world, IMSIs and MSISDNs are related). No TMSI is assigned to an MS when it is created but one will be allocated after it has performed a successful location update. The datum *noTMSI* represents 'no TMSI' and is used as a place holder. Similar data are specified for other identities, e.g., *noIMSI*, *noMSISDN*, etc.

```
process MobileStations[mmi,oami,ri](usedMSISDNs:MSISDNSet):noexit:=
  oami ?dn:MSISDN [(NotIn(dn,usedMSISDNs)) and (dn <> noMSISDN)];
  ( MStation[mmi,ri](IMSIof(dn),noTMSI,dn)
    |||
    MobileStations[mmi,oami,ri](Insert(dn,usedMSISDNs))
  )
  where
    ...
endproc (*MobileStations*)
```

Figure 23 Specification of an arbitrary number of mobile stations.

5.3.3 Specification of the PLMN

The specified PLMN follows the canonical architecture of GSM (Figure 5), and contains all signalling interfaces that could be found in any implementation. Recall that the GSM Recommendations leave open the number of components that may exist in a PLMN. The specified network contains one GMSC, one HLR, and unbound numbers of MSCs, VLRs, BSCs, and BTSs. The number of VLRs that can be created always equals the number of MSCs because a VLR is associated with every MSC. The top-level specification of a PLMN is shown in Figure 24.

A graphical representation of the structure of the whole specification is given in Figure 25. To see the similarity between this structure and the GSM's network topology, compare Figure 5 and Figure 25. It can be seen that the lines connecting the boxes mirrors the interconnections between the GSM machines in Figure 5. In this figure, LOTOS processes are drawn as solid boxes and synchronization is denoted by connecting lines. Dashed boxes represent parentheses in the behavior expressions. Little solid rectangles denote points of one-to-one communication, where one of the processes from one side of the rectangle can synchronize with one of the processes connected to the other side. The shaded box represents a process that does not correspond to a physical GSM machine (the necessity for this process is explained below). Arrows denote points of synchronization where data are passed only in one direction. The technique for specifying an unbound number of objects is used to specify a configurable network of MSC/VLR pairs and BSSs. In Figure 24, process `MSCVLR_Installer` represents an "installer" entity that deploys GSM machines in the PLMN. It allows any number of MSC/VLR pairs and their associated Base Station Subsystems (BSSs) to be dynamically created or

```
process PLMN[ext,oami,ri]:noexit:=
  hide di,ei1,ei2,gi1,gi2,zi in

  ( GMSC[ci,ext,zi]
    |[zi]|
    MSCVLR_Installer[di,gi1,gi2,ei1,ei2,ri,oami,zi]({} of MSCIdSet)
    |[ei1,ei2,gi1,ei2]|
    MSCVLR_Comm[ei1,ei2,gi1,gi1,gi2]
  )
  |[di]|
  HLR[ci,di,oami](hlr1,{} of HlrRecSet)

  where
  ....
  process MSCVLR_Installer[di,gi1,gi2,ei1,ei2,ri,oami,zi](..):noexit:=
    hide ai,bi in
    oami ?newmsc:MSCId;              (*synch with OSS,Figure 22,line 6*)
    ( VLR[bi,di,gi1,gi2](msc)
      |[bi]|
      MSC[ai,bi,ei1,ei2,zi](msc)
      |[ai]|
      BSS_Installer[ai,ri,oami](msc, {} of BSCIdSet)
    )
    |||
    MSCVLR_Installer[...](..)                          (*recursive call*)

    where
    process BSS_Installer[ai,ri,oami](..):noexit:=
      hide abisi in
      ( BSC[ai,abisi](bsc)
        |[abisi]|
        BTS_Installer[abisi,ri,oami](bsc,{} of BTSIdSet)
      )
      |||
      BSS_Installer[ai,ri,oami](..)

      where
          ....
  endproc (*MSCVLR_Installer*)
endproc (*PLMN*)
```

Figure 24 Specification of a configurable Public Land Mobile Network (PLMN).

deployed. Recall that each MSC/VLR must be connected to at least one BSS in order to be able to communicate with mobile stations. Process *BSS_Installer* is a similar installer process that allows the creation of any number of BSSs. The same interleave recursive structure is used to specify an unbounded number of configurable BSSs, each consisting of a BSC and an arbitrary number of BTSs. The embedding of *BSS_Installer* in the body of *MSCVLR_Installer* means that it is invoked with each creation of MSC/VLR. Since

BSS_Installer specifies an arbitrary number of BSSs, any number of base station subsystems can be associated with each MSC/VLR. In this sense, *MSCVLR_Installer* specifies a configurable network, i.e., any number of MSC/VLR pairs can be created (in *MSCVLR_Installer*) and each pair can be connected to any number of BSSs (through *BSS_Installer*).

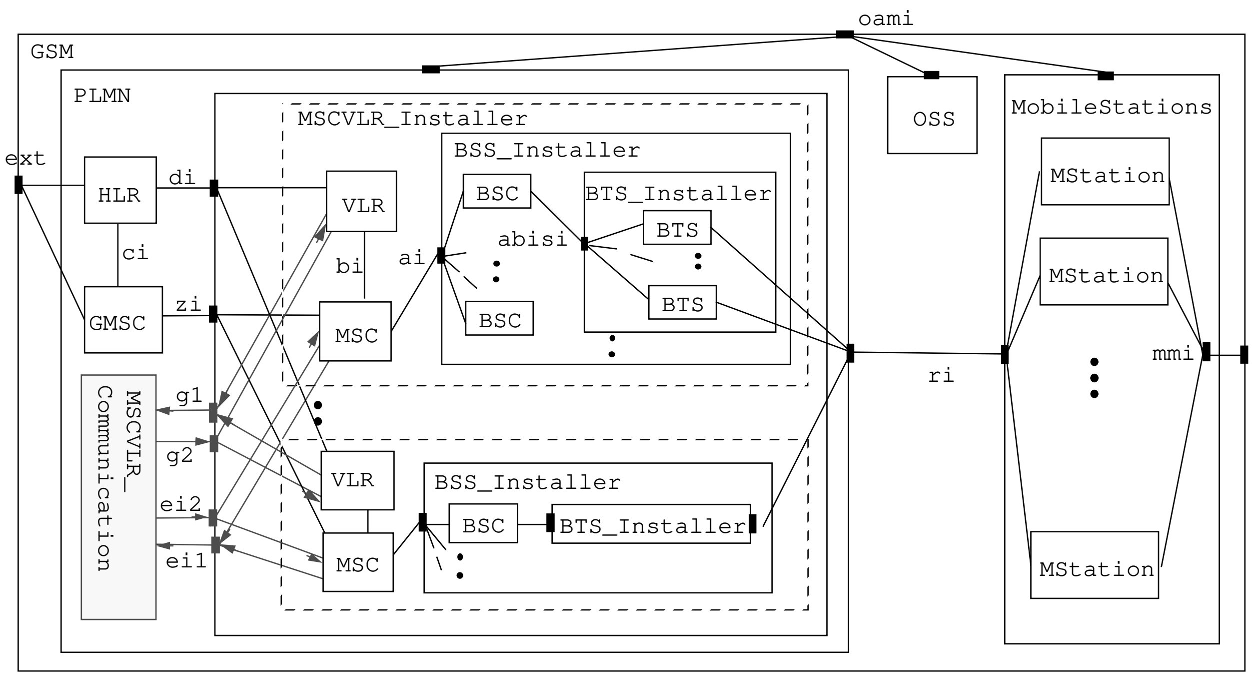


Figure 25 Graphical representation of the top-level structure of the specification.

This specification structure of embedded installer processes might appears at first glance to be inconsistent with the GSM architecture, which considers MSC/VLR, BSC and BTS machines as being at the same abstraction level. Specifically, it is inconsistent with the *contains* (Section 4.2.1) relationships between some components. The embedding of *BSS_Installer* in *MSCVLR_Installer* and of *BTS_installer* in *BSS_Installer* gives the impression that BSSs are sub-components of MSCs/VLRs and BTSs of BSCs, respectively. However, such an interpretation is not entirely inaccurate since MSCs and BSCs depend on the VLRs and BTSs, respectively, for their operations, i.e., MSCs/VLRs depend on BSSs to provide stable links to mobile stations, and BSCs need BTSs to do the actual transmission and reception of messages to/from mobile stations.

The declaration of gate `ai` at the `MSCVLR_Installer` level enables a different instance of the interface to be created (although they have the same name) for each MSC/VLR instance and its BSSs. This means that an MSC/VLR pair can communicate only with the BSSs with which it shares an interface.

The specification structure of `MSCVLR_Installer` builds correctly communication paths from MSCs to BTSs (e.g., between an MSC and its BSCs, and between a BSC and its BTSs), but does not allow instances of the same component to communicate. This is correct for BSC and BTS components, but is incorrect for MSC and VLR whose instances need to communicate to carry out their functionalities. That is, the *connects-to* static relationship (Section 4.2.1) between instances of the same components need to be represented.

In the GSM specification, the processes that enables inter-MSC and inter-VLR communications are `InterMSC_Comm` and `InterVLR_Comm`, respectively. These processes are specified using the specification structure given in Section 3.5.2, and are subprocesses of in a process called `InterMSCVLR_Comm`, see Figure 26. `InterMSC_Comm` accepts messages sent by sender MSCs at gate `ei1` (line 9) and passes them to recipient MSCs via gate `ei2` (line 10); `InterVLR_Comm` behaves the same way, but uses gates `gi1` and `gi2`. In a sense, these processes could be considered to model the SS7 transport network that is utilized by NSS components. There is an expression for each message format (i.e., event structures) that could be passed between MSCs and between VLRs. The event structures include the identities of the senders and recipients of the messages so that `InterMSCVLR_Comm` knows to which recipients to relay a message (more on event structures in Section 5.3.4). Note that this process may not send out messages in the same order as they are received. This is an appropriate representation because messages may travel in SS7 at different speeds.

```
process InterMSCVLR_Comm[ei1,ei2,gi1,gi2]:noexit:=
   InterMSC_Comm[ei1,ei2]
   |||
   InterVLR_Comm[gi1,gi2]

   where

   process process InterMSC_Comm[ei1,ei2]:noexit:=
      ei1 !MAPE ?snder:MSCId ?recver:MSCId ?msg:HO_Primitive;
      ( ei2 !MAPE !snder !recver !msg;
         stop
         |||
         InterMSC_Comm[ei1,ei2]
      )
      endproc

      ...
endproc (*InterMSCVLR_Comm*)
```

Figure 26 Inter-MSC and inter-VLR communications.

5.3.4 Specification of Interfaces

In LOTOS, interfaces between objects are modeled by gates. In our specification, the majority of the gates represent interfaces between components rather than between specific instances of components. That is, these gates model interfaces that may have multiple instances of objects on one or both sides. An object on one side of such a gate can synchronize with any one of a number of objects on the other side at a time. Multi-way synchronization was not used; all synchronization occurred between two processes. For events that occur on gates that have multiple objects on any of its side, *gate splitting* (see Section 3.5.3) was utilized to create a different 'channel' for each pair of objects. This technique allows to distinguish instances of components on one side of a gate. The experiments that are used to split the gates are the identities of the senders and recipients of messages (more on this later).

Figure 25 gives a graphical illustration of the gates used in the specification. (Remember that little solid rectangles denote points of one-to-one communication; one of the processes on one side may synchronize with one of the processes on the other side). The gates could be grouped into the following categories, according to the number of objects on each of their two sides.

One-one Gates: These gates model interfaces that have only one instance of a component on either sides. The interface between the HLR and GMSC, and between each MSC and its associated VLR are examples of interfaces of this kind. Each of these interfaces is represented by one LOTOS gate. The HLR-GMSC interface is represented by the gate `ci`. The HRL-GMSC interface is specified thus because the specified PLMN has only one HLR and GMSC (see Section 5.3.3). Interfaces between different pairs of MSCs-VLRs are modeled by separate and independent gates, although they share the same name `bi`. This way of specification is a result of the decision to associate one VLR with each MSC (Section 5.3.3).

One-many Gates: These gates model interfaces that have one instance of a component on one side and multiple instances of another component on the other side. The interfaces between the HLR and all the VLRs, between the GMSC and all the MSCs, between each MSC and all the BSCs under its control, and between each BSC and all the BTSs it controls are each represented by one LOTOS gate. This does not mean that the communication mode is one-to-many (or broadcast). Rather, what it means is that each HLR, GMSC, MSC and BSC on one side of the interface can synchronize respectively with only one of the many VLRs, MSCs, BSCs, and BTSs on the other side. The HLR-VLR, and GMSC-MSC interfaces were represented in this manner because the PLMN has only one HLR and GMSC and indeterminate numbers of VLRs and MSCs. The MSC-BSC and BSC-BTS interfaces were specified thus because it is the most appropriate structure when specifying a configurable network.

Many-many Gates: These gates model interfaces that have multiple instances of a component on each side. The radio interface between all the BTSs and all the MSs is represented by one gate, `ri`. At this interface, one MS communicates with exactly one BTS at a time. This rather highly abstract representation is necessary in order to be able to specified the handover scenario (Section 2.4.2). As discussed in Section 3.5.3, it is not possible to specify the handover scenario if the interface between an MS and each BTS is represented by a gate.

One-way Gates: All gates of the above three kinds are bidirectional. Processes involved in a synchronization on such a gate can both offer and receive data. This is the semantics of LOTOS gates. In the specification, some gates (specifically, `e1`, `e2`, `g1` and `g2`) were used only as one-way gates. Gates `e1` and `g1` are the points at which MSCs and VLRs, respectively, pass data to process `MSCVLR_Communication`. Gates `e2` and `g2` are the points at which `MSCVLR_Communication` passes data to MSCs and VLRs, respectively. One-way gates are necessary to allow inter-MSC and inter-VLR communication. Section 3.5.2 discusses more on these gates.

Event Structures

As discussed in Section 3.2.2, a LOTOS denotation of an event is structured and consists of a gate and an optional list of experiments or data fields. In our specification, we took advantage of the composite nature of LOTOS events and used experiments to denote information associated with events, see Definition 4.1. Many different event structures were used, but they are all based on the following general format

```
gatename !protocol_name !sender_id !recipient_id !channel_id !primitive ...
```

The experiment *protocol_name* identifies the protocol to which the primitive belongs. The meanings of *sender_id*, *recipient_id*, and *primitive* are obvious. The ellipses in the general event structure indicate other information that might be sent with the primitive. For example, because most signalling messages pertain to specific mobile stations, the identities of MSs are sometimes included in the events.

Note that gate-splitting is used here. The experiments *sender_id* and *recipient_id* differentiate the interface represented by the gate *gatename* into 'sub-gates' that represent interfaces between particular pairs of objects. For instance, the events shown below represent two distinct interfaces, one between a BTS named *bts1* and an MS with TMSI *tmis1*, and another between *bts1* and *tms2*.

```
ri !RIL3CC !bts1 !tmsi1 !chan1 !Alerting;

ri !RIL3CC !bts1 !tmsi2 !chan1 !Alerting;
```

Note that the experiment *chan1* *further* differentiates the interface between *bts1* and *tmsi1* into channels (chan1 represents radio channel 1). The above events denote the transmission by *bts1* of the RIL33-CC Alerting primitive to the mobile stations *tmsi1* and *tmsi2* to command them to ring.

Events that represent communication in the downlink direction (from the network to a MS) do not have the *sender_id* experiment because the identity of the sender is obvious. For example, a BTS always knows which BSC sends it a message since it is connected to only one BSC.

5.4 Specification of GSM functions

The previous section described the specification of the GSM network architecture. In this section we present the specification of GSM's dynamic properties, i.e., its behavior. This specification was derived from a Scenario Model of GSM. Different kinds of information were extracted from the source documents. The book [MoP92] of informal descriptions was useful for understanding the services offered by GSM and the general mechanisms by which these services are rendered. The available scenarios provided concrete details on how the GSM components interact to provide services, and were the main input for this part of the specification. In particular, they gave the specific protocol primitives that are exchanged between components. The standardization document was consulted to determine parameters and data that are passed with the primitives and alternative or variances of scenarios. It and the book provided information that allowed to integrate the scenarios and filled in 'holes' in understanding.

Consistent with the distributed perspective, our specification regards GSM services as realized by the co-operation of distant objects. According to this view, the behaviors of the specification are the interactions between LOTOS processes that represent the objects. An interaction or communication involves two kinds of objects: peer *end-point* object that interact to provide a service, and *relay* objects that transport messages between end-point objects. Relay objects are specified as infinite-sized buffers that relay messages in a transparent manner (i.e., they do not look at the contents of the messages), and they may send messages in different orders from the order in which they were received. The specification of relay objects follows the structure given in Section 3.5.2.

The general form of the specification of scenarios is illustrated in Figure 27, arrows represent messages. Their top-level structures of processes of objects are influenced by the grouping of scenarios into layers (see Figure 6 on Page 25), each process is composed of three sub-processes that represent the RR, MM and CM entities. The figure depicts a CM and an RR scenario; the relay nodes for the CM scenario are the BSC and BTS, and that for the RR scenario is the BTS. The specification of the end-point objects determine the order of sending and receiving of messages between these objects. Communication between layers within a component is not shown. The specifications of individual GSM components are discussed in the sub-sections to follow.

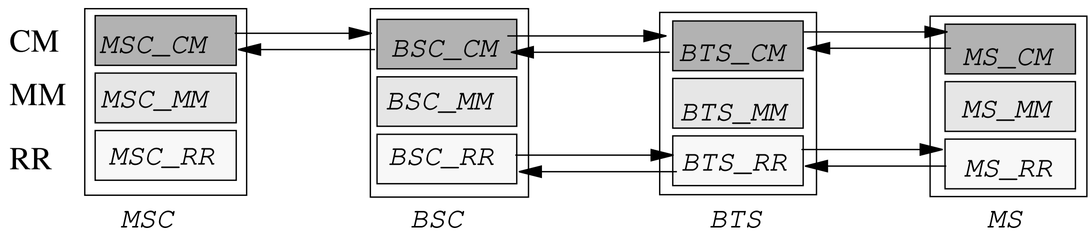


Figure 27 Specification of GSM scenarios from a distributed perspective. Each block represents a LOTOS process.

5.4.1 Specification of Cell and Mobility

The essence of GSM (or any cellular system) are the concepts of cells and mobility. The specifications of *geographical areas* (cells) and *mobility* are interesting, and show the strength of LOTOS and the resource-oriented style. They are specified indirectly through the specification of other concepts.

In the specification, cells are modeled by the identities of the BTSs that provide radio coverage for those cells. The existence of a cell is implicitly associated (in the mind of the specification user) with the existence of its BTS, i.e., there is no LOTOS text that explicitly specifies cells. The roaming of an MS within a cell is represented implicitly by the corresponding `MStation` knowing the identity of that cell's BTS. The specification of MSC areas (Section 2.3.3) follows a similar approach; the identities of MSCs are used to denote MSC areas. This means that a mobile station's current location is specified by giving the identities of the MSC and BTS with which it is in communication.

The mobility or *physical displacement* of mobile stations within and across cells is also specified implicitly. In our specification, this concept is modeled by the occurrence of events that are triggered by mobility. It is specified as follows; certain MS events (crossing of cell boundaries, etc.) trigger the execution of the handover and/or location update scenarios (described in Section 2.4.2 and Section 2.4.3, respectively). At appropriate points, the specification gives the possibility to perform these scenarios by offering to synchronize with the environment. Thus, whether the scenarios are executed or not depend on the environment. If a scenario begins execution, then the environment must have supplied the appropriate events, and from the points of view of both the system and the environment, the triggering events must have occurred. This approach specifies indirectly mobile station roaming, through the *possibility* to perform location update and handover at appropriate points in the behavior of the system.

5.4.2 Specification of the Mobile Station

The process `MStation` in Figure 28 models mobile station, and its sub-processes `MS_RR`, `MS_MM` and `MS_CM` represent the RR, MM and CM entities, respectively. Since most of the GSM scenarios involve the MS at one end (and an infrastructure machine at the other), the specification of mobile station is probably the most complex.

The specified behavior of mobile stations is as follows. The first scenario performed by `MStation` after its instantiation/creation consists of choosing a cell and listening to its broadcast channels. The selection of a cell is specified by a non-deterministic choice of BTS identities (line 28). After cell selection, the MS performs location update (line 13) which, if successful, results in a TMSI being assigned. Once the MS has been registered, denoted by its TMSI not being equal to `noTMSI`, it enters *idle state* and can initiate any scenario (line 6).

From the idle state, the MS can initiate a scenario by first obtaining radio channels. This is specified in `MS_RR`. Once radio channels have been granted, the MS goes into dedicated mode, and `MS_RR` informs either `MS_MM` or `MS_CM` to this effect by synchronizing at gate `MS_SAP1` or `MS_SAP2`, respectively (lines 15-19). If the scenario to be performed is an MM (CM) scenario, then the synchronization is with `MS_MM` (`MS_CM`) and the other process remains blocked. Upon completion of the scenario, `MS_MM` (`MS_CM`) synchronizes again with `MS_RR`**.** `MS_MM (MS_CM)` then returns to the block state and `MStation` re-enters into idle state. This cycle can continue for any number of scenarios. If the `IMSIDetach` procedure is performed (line 24), then the MS is deactivated. In such a case, the SIM must be re-inserted, a cell selected and the MS re-registered with the network before the MS can again perform any scenario.

In the specification of MO and MT calls, a handover may occur only after an end-to-end connection has been established. During a connection, if the MS is regarded as roaming within one cell, then no action is required; if it is thought of as having crossed a cell boundary, then a handover is required. Recall that mobility is specified indirectly (Section 5.4.1). Since a mobile station's mobility is not explicitly specified, whether a handover is needed or not is determined by the environment.

```
process MStation[mmi,ri](im:IMSI,tm:TMSI,dn:MSISDN,cell:BTSId): noexit:=
  hide MS_SAP1,MS_SAP2 in

  [(noBts <> cell)] ->                    (*If MS is in touch with a cell*)
    ( [(noTMSI <> tm)] ->                          (*If MS has registered*)
        ( choice serv:Service [] exit(serv)   (*choose scenario to exec*)
          []
          ri !RILCC !cell !dn !PAGCH !PageCmd; (*accept a paging cmd*)
          exit(PAGING_RESP)
        )
      []
      [(noTMSI == tm)] ->                        (*MS has not registered*)
         i; exit(LOCUPD)                            (*do location update*)
    ) >> accept srv:Service in
      ( MS_RR[mmi,MS_SAP1,MS_SAP2,ri](im,tm,dn,noCksn,cell,srv)
        |[MS_SAP1,MS_SAP2]|
        ( MS_MM[mmi,MS_SAP1,ri](im,dn)
          |||
          MS_CM[mmi,MS_SAP2,ri](im,dn)
        )
      )                         (*procedure performed was not IMSIDetach*)
      >>( [srv <> IMSIDetach] -> MS[mmi,ri](im,tm,dn,cell)
          []                                  (*IMSIDetach was executed*)
          [srv == IMSIDetach] -> i; MS[mmi,ri](im,noTMSI,dn,noBts)
        )
  []
  [(noBts == cell)] ->                      (*MS not in touch with a cell*)
    ( choice bts:BTSId []                                  (*choose a cell*)
        [bts <> noBts] ->
          ....                       (*get sync'ed, listens to broadcast*)
          MStation[mmi,ri](im,tm,dn,bts)
        []
        [bts == noBts]->                     (* 'invalid' cell was chosen*)
          i;                                  (*sync'zn chanl not found*)
          MStation[mmi,ri](im,tm,dn,bts)             (*choose cell again*)
    )
  where
    ......
endproc (*MStation*)
```

Figure 28 Specification of the Mobile Station coponent.

5.4.3 Specification of the BTS

In the real world, base transceiver stations are more than ‘dumb’ transmitters and receptors of radio signals, they also perform computations. For instance, they must process commands from their controllers, the BSCs. Most of the computations they perform are related to RR scenarios. For MM and CM scenarios, BTSs behave mainly as transport nodes that simply relay messages.

The specification of BTS is given in Figure 29. Processes *BTS_MM* and *BTS_CM* represent the MM and CM behaviors, respectively. They processes relay MM and CM messages without examining their contents. Their specification structures are similar to that of relay objects given in Section 3.5.2.

BTS_RR describes the BTSs’ participation in RR scenarios such as broadcasting synchronization information, processing mobile stations’ requests for radio channels, ciphering and handover.

```
process BTS[abisi,ri](bts:BTSId):noexit:=
   BTS_RR[abisi,ri](bts)
   |||
   BTS_MM[abisi,ri](bts)
   |||
   BTS_CM[abisi,ri](bts)

   where

   process BTS_RR[abisi,ri](bts:BTSId):noexit:=
      BTS_Operations[abisi,ri](bts)
      |||
      BTS_Broadcast[ri,abisi](bts)

      where
         ...
   endproc (*BTS_RR*)
endproc (*BTS*)
```

Figure 29 Specification of the BTS coponent.

5.4.4 Specification of the BSC

Base station controllers are the machines that are responsible for performing most of the RR functions. As such, the specification of their RR layer is more complex than that of any other component, with the exception of the MS. RR scenarios are specified by several LOTOS processes, as shown below.

```
process BSC[ai,abisi](bsc:BSCId):noexit:=
  hide intraBSC in

  BSC_RR[ai,abisi,intraBSC](bsc)
  |[intraBSC]|
  BSC_CM[ai,abisi,intraBSC](bsc)
  |||
  BSC_MM[ai,abisi](bsc)

  where
    process BSC_CM[ai,abisi,intraBSC](bsc:BSCId):noexit:=
      BSC_CM_Dnlink[ai,abisi,intraBSC](bsc)
      |||
      BSC_CM_Uplink[ai,abisi,intraBSC](bsc)

      where
      ...
    endproc (*BSC_CM*)

    process BSC_RR[ai,abisi,intraBSC](bsc:BSCId):noexit:=
      BSC_Broadcast[abisi,oami]      (*Broadcast synch info, paging,..*)
      |||
      BSC_Access[abisi,ai](bsc)        (*Network Access, channel alloc*)
      |||
      BSC_Ciphering[ai,abisi](bsc)     (*Comnd BTSs to cipher transm'n*)
      |||
      BSC_HO[ai,abisi,intraBSC](bsc)                        (*Handover*)
    where
        ...
    endproc (*BSC_RR*)
    ...                              (*definitions of other processes*)
endproc
```

Figure 31 Top-level specification of the BSC component.

Process `BSC_Access` specifies the behavior of BSCs in Network Access scenarios (Section 2.4.2), and includes descriptions of radio channel allocations. The rules for the allocation of signalling and data channel are not fully defined in the GSM standard, implementors are required to define their own policies of when to grant or refuse channels. This means that we are required to devise an algorithm in order to have an executable model. Fortunately, LOTOS supports non-determinism and we are able to define a non-deterministic algorithm that is not very restrictive with respect to the policy defined in the standard. The specified algorithm is as follows: when a mobile station requests radio channels, a BSC may choose to assign the channels immediately, to defer the assignment until later, to reject the channel request, or to simply ignore the request if CM services or location update is specified as the purpose of the request (see “Network Access Procedure” on page 26.) Which one of these options is taken is left for the environment to decide. However, when the request is for IMSI Detach (Section 5.4.2), *BSC_Access* cannot refuse a channel request and must assign a signalling channel immediately. The reason for this decision is that a mobile station that performs this scenario typically has been switched off (the SIM is removed or some kind of de-activate code has been entered) and therefore cannot receive refusals from the network. For channel requests that come from MSCs, the algorithm assigns channels immediately. MSCs request channel assignment in situations where data channels were not assigned initially during Network Access or when the assigned channel is not of the right type (i.e., it is too slow for the invoked service), or when new channels are required during handover.

BSC_Ciphering interprets ciphering requests (commands) from the MSC/VLR and commands the MSs appropriately.

One of the major responsibilities of BSCs is to perform handover scenarios to maintain stable connections with MSs during calls. The internal gate *intraBSC* permits *BSC_RR* to receive a message indicating that a connection has been established so that handover may take place if necessary. Once *BSC_HO* receives information on *intraBSC* that a connection has been established, the possibility to perform handover is offered to the environment (see Section 5.4.1).

BSC_HO is composed of two sub-processes (*Init_HO* and *Resp_to_HOCmd*) that represent the two roles (BSC-old, and BSC-new) played by the BSC in handover. *Init_HO* represents BSC-old, and describes the behavior of a BSC when it initiates a handover. Depending on the type of handover (Section 2.4.2), this process may involve *MSC*s and another *Init_HO*.

Resp_to_HOCmd represents the role of BSC-new. It describes the behavior of the BSC when it is commanded by another BSC to takeover the management of a communication. It and *Init_HO* make use of a general process, called *EstNewPath*, that establishes a path between BSC-new and the MS, i.e., *EstNewPath* is an example of subordinate scenarios (see “Determination of relationships between scenarios” on page 70)

BSCs are not very involved in MM scenarios. As such, the descriptions of the MM entity are straightforward. The structure of *BSC_MM is* similar to that of relay objects given in Section 3.5.2.

BSCs are more involved in CM scenarios than in MM ones. They participate in some CM scenarios and play the role of relay agents for others. *BSC_CM* is composed of *BSC_CM_Dnlink* and *BSC_CM_Uplink*. In addition to relaying some CM messages in the downlink direction, *BSC_CM_Dnlink* also deals with MSCs's requests for channel assignments in cases where data channels were not assigned at initial access (Network Access). It also performs MSCs' commands to page mobile stations in MT-call scenarios, and to use certain keys to cipher transmissions. In addition to its relay function, *BSC_CM_Uplink* synchronizes with *BSC_RR* when a 'Connect' message from an MS destined for an MSC is received. The synchronization informs *BSC_RR* that a connection has been established and that handover becomes possible.

5.4.5 Specification of the MSC

MSCs operate closely with their associated VLRs, and together they display complex behaviors. In fact, the operations of the MSCs are governed by data in the associated VLRs. In the specification, this means there is a large number of synchronizations between the processes representing these components. The specification of MSCs is shown in Figure 35. Note that for reason of simplicity, we did not include gateway functions in the specification of MSCs, and the specification of GMSCs (Section 5.4.8) does not include MSC functions.

Like most processes representing other network components, *MSC* is composed of three sub-processes: *MSC_RR, MSC_MM* and *MSC_CM* (lines 4-9). *MSC_RR* describes two RR scenarios that involve MSCs, namely cipher mode setting (Section 2.4.3) and handover. MSCs can play one of three roles (Section 2.4.2) in handover, as anchor-MSC, new-MSC, and as the MSC that is currently serving the MS. These three roles are specified in separate sub-processes of *MSC_HO* (line 15).

MSCs are highly concurrent systems, they are able to simultaneously participate in many scenarios, and the structure of *MSC_MM* reflects this. The interleave expressions of the recursive processes *MSC_MM_LocUpd*, *MSC_MM_IMSIReq*, *MSC_MM_AllocIMSI* and

MSC_IMSIDetach (lines 21-29) mean that MSCs are always ready to execute any number of instances of MM scenarios.

Execution of the location update procedure (a procedure of the MM layer) is not complete until the set cipher mode scenario (a scenario of the RR layer) has been executed successfully. Gate *MS_SAP1* serves as a synchronizing point where *MSC_MM* can inform *MSC_RR* of the completion of location update so that cipher mode setting can commence in *MSC_RR*.

MSC_CM specifies the MSCs' behavior in call management. This process is composed of the two sub-processes *MSC_CM_MO* and *MSC_CM_MT*. The former specifies mobile-originated calls and the latter mobile-terminated calls. These two subprocesses are composed in an interleave expression to allow *MSC* to manage any number of MO and MT calls simultaneously.

```
process MSC[ai,bi,ei1,ei2,zi,ext](msc:MSCId):noexit:=
   hide MS_SAP1C in

   (  MSC_MM[ai,bi,MS_SAP1C](msc)
      |[MS_SAP1C]|
      MSC_RR[ai,bi,ei1,ei2,MS_SAP1C](msc)
   )
   |||
   MSC_CM[ai,bi,zi,ext](msc)

 where
   process MSC_RR[ai,bi,ei1,ei2,MS_SAP1C](msc:MSCId):noexit:=
      MSC_Ciphering[ai,bi,MS_SAP1C](msc)
      |||
      MSC_HO[ai,bi,ei1,ei2](msc)
   where
      ......
   endproc

   process MSC_MM[ai,bi,MS_SAP1C](msc:MSCId):noexit:=
      MSC_MM_LocUpd[ai,bi](msc)
      |||
      MSC_Authen[ai,bi](msc)
      |||
      MSC_MM_IMSIReq[ai,bi](msc)
      |||
      MSC_MM_AllocIMSI[ai,bi,MS_SAP1C](msc)
      |||
      MSC_IMSIDetach[ai,bi](msc)
   where
            ...
   endproc (*MSC_MM*)

   process MSC_CM[ai,bi,zi](msc:MSCId):noexit:=
      ai !BSSMAP ?bsc:BSCId !msc, ?bts:BTSId ?tm:TMSI ?chan:Channel
               !CMServ !CompleteLayer3Info;
      (  MSC_CM_MO[ai,bi,zi](msc,bsc,bts,tm,chan)
         |||
         MSC_CM[ai,bi,zi](msc)
      )
      []
      zi !msc !IAM ?dn:MSISDN;     (*from GMSC*)
      ( MSC_CM_MT[ai,bi,zi](msc)     (*zi:interface between GMSC & MSCs*)
         |||
         MSC_CM[ai,bi,zi](msc)
      )
   where
      .......
   endproc
```

Figure 32 Specification of the MSC component.

5.4.6 Specification of the VLR

A VLR serves a database function, and its information is accessed frequently by the associated MSC. Therefore, a significant message traffic exists between the two objects. In our specification, the VLR plays a larger role than simply a database; in MM scenarios, it behaves as if it is a peer end-point entity of the *MStation*. The top-level specification of the VLR is shown below.

The gate *intraVLR* allows communication between the process that represents the storage part (*VLR_Table*) and those that represent the behavior parts (*VLR_MM* and *VLR_CM*) of the VLR component. *VLR_Table* represents the internal storage structure of the database. The actual storage structure is modeled in ADTs as a Set of type *VlrRecSet*. An instance of this type is included in the parameter list of *VLR_Table* (line 1). The behavior of this structure is modeled by a collection of alternative expressions that represents the operations of the database, i.e., Insert, Remove, etc.

VLR_MM is composed of processes which represent location update and security-related scenarios. *VLR_CM* specifies MSC-VLR communications in the executions of CM scenarios. The interleave composition of *VLR_MM* and *VLR_CM* means that VLR has concurrent behavior. However, the alternative definition of *VLR_Table* ensures that it does not lead to inconsistent data. The structure of *VLR_Table* is similar to that of the process representing the HLR in Figure 34. Thus, sequential access to VLR data is ensured by internal behavior.

```
1  process VLR[bi,di,gi1,gi2](msc:MSCId, vlr_content: VlrRecSet):noexit:=
2     hide intraVLR in
3
4     VLR_Table[bi,intraVLR,di,gi1,gi2](msc,vlr_content)
5     |[intraVLR]|
6     (  VLR_MM[intraVLR,bi,di,gi1,gi2](msc)
7        |||
8        VLR_CM[intraVLR,bi](msc)
9     )
10    where
11       .....
12 endproc
```

Figure 33 Specification of the VLR component.

5.4.7 Specification of the HLR

The specified PLMN has one HLR, modeled by the process `HLR` (Figure 34). The definition of `HLR` is similar to that of `VLR_Table` except that `HLR` interacts directly with external objects, which means that it cannot have concurrent behavior or risk having inconsistent data. The storage structure is specified in ADTs as a Set of type `HlrRecSet`. Each subscription in the HLR is specified as a 3-tuple of MSISDN, IMSI and MSC identity (see Figure 36). The first two elements identify the MS, and the last element identifies the MSC areas in which the associated MS is currently roaming.

The external behavior of the HLR is defined as a collection of alternative behavior expressions, each specifying an operation that the database can perform. The expressions are recursive so that at the end of each operation all the operations are once again available. For example, `HLR` can synchronize with a process (line 2) and modifies the storage structure accordingly (line 3). The actual operations (Insert, Remove, etc.) on the database are not visible from the outside (line 3). These operations are also specified in ADTs.

Thus, the specification of VLR and HLR shows two ways of modeling databases in LOTOS; one approach allows concurrent accesses to the database but still can maintain consistency of data, the other permits only one operation on the database at a time.

`HLR` can accept and process new subscription records from the network administrator (line 2), commands from VLRs to update records during handover (line 5), and a GMSC's requests for routing information (i.e., the MSC area of in which a particular MS is roaming) (line 13).

```
process HLR[di,oami](myid:HLRId,content:HlrRecSet):noexit:=
  oami !INSERT_SUBSCRIPTION ?rec:HlrRecord;  (*agree to insert a record*)
  HLR[ci,di](myid,Insert(rec,content))              (*actual insertion*)
  []
  di !MAPD ?msc:MSCId !myid !UpLoc ?im:IMSI ?dn:MSISDN;      (*VLR->HLR*)
  ( [IsInHLR(dn,content)] -> ...
    []
    [Not(IsInHLR(dn,content))] ->
       (*...HLR must have crashed and lost some info. Insert
        *info on mobile station 'dn' back in HLR, and
        *responds normally...*)
   )
  []
  ci !MAPC !RIL3CC !myid !SendRoutingInfo ?dn:MSISDN;       (*From GMSC*)
  ([IsInHLR(dn,content)]->
    ( let msrn:MSCId=LocAreaOf(dn,content) in
      [noMsc <> msrn] -> ...(*Send Routing Num to GMSC*)
      []
      [noMsc == msrn] -> ...(*Routing Num not in HLR, query VLR*)
    )
    []
    [not(IsInHLR(dn,content))] ->...(*No recorde of MS in HLR*)
  )
  (**[]
   This choice represents an HLR crash and some loss of info.
   i;
   choice lostrec:IMSI []
   HLR[di,oami](myid,RmHLRrec(lostrec,content)) **)
```

Figure 34 Specification of the HLR component.

5.4.8 Specification of the GMSC

In the standard, a GMSC is really that subset of an MSC's functions that allow it to serve as a gateway to external networks. In our specification, GMSCs are modeled as separate entities from MSCs. A skeleton of the specification is given in Figure 35. The gates *ext*, *ci*, and *zi* represent interfaces with the external network, HLR and MSCs, respectively. In the standard, no name is given for the interface between GMSC and MSC because they are specified as one component. *GMSC* can handle arbitrary numbers of MT-call setup requests from the external network and MO-call setup requests from the MSCs. For MT calls, *GMSC* queries *HLR* to determine the identity of the *MSC* to which the calls should be routed.

```
process GMSC[ci,ext,zi]:noexit:=
     ext !RIL3CC !IAM ?dn:MSISDN;    (*receives MT-Call from ext. network*)
     (* call setup..
        (..release by external nework..exit
         []
        ..release by MS..exit
        )
        |||
        GMSC[ci,ext,zi]
      *)
     []
     (*..behavior for MO call..*)
endproc
```

Figure 35 Specification of the GMSC component.

5.5 Specification of data types

In LOTOS, data and data structures are specified using the abstract data type (ADT) part of the language. In our specification, data type specifications can be grouped into three categories: object identities, protocol primitives, and complex data structures.

The first category of data type specification represents simple types that consist mainly of collections of unique elements and the equality and inequality operators. The elements of these types are used as identities of GSM objects, radio channels, random numbers, cipher modes, etc. Most of these types were specified by mapping to a generic type for which the equality (*eq*) and inequality (*ne*) operators have been defined. This mapping is a form of re-use (Section 3.2.1) of specification.

Since the behavior part of our specification allows any number of instances of some GSM components to be (dynamically) created, the ADT specification of these objects' identities must also permit dynamic creation of unique elements. Such ADT specification is possible by mapping to the *Natural* number specification provided with the LOTOS standard or even by specifying the types in the same way as *Natural*'s. In the ADT specification of natural numbers, a number *N* is specified by the function *succ()* applied on the representation of the number *N*-1, with the number 0 represented by the symbol *0*. That is, the number 1 is represented by *succ(0)*, the number 2 by *succ(succ(0))*, the number 3 by *succ(succ(succ(0)))*, etc.

However this way of specification is difficult for human to work with, i.e., when he is required to enter large ‘values’. Thus, instead we chose to specify GSM object identities of the form shown below for MSC identities. This specification allows only ten names to be created because there are only ten digits. Although this ADT specification is not strictly consistent with the behavior specification (the former allows only ten names to be created while the latter allows any number of objects to be created), it causes no difficulty in most practical cases because in these cases no more than one or two instances of each GSM components are created in the specification execution. The current state of LOTOS tools is such that creating more than a few instances of objects as complex as the GSM components increases the state space of the specification to a size the tools cannot handle efficiently. Furthermore, the largest number of any one GSM component required to illustrate all GSM scenarios is three, i.e., the three MSCs in handover (see Figure 7).

```
type MSCId is Digit, Boolean
sorts MSCIdSort
opns
msc : Digit -> MSCIdSort
_eq_, _ne_: MSCIdSort,MSCIdSort ->Bool
eqns forall n1,n2:Digit
ofsort  Bool
(n1 eq n2) => (msc(n1) eq msc(n2)) = true ;
(n1 neq n2) => (msc(n1) eq msc(n2)) = false;
msc(n1) neq msc(n2) = not(n1 eq n2);
endtype
```

In our specification, the names of the PLMN objects are required to reflect their interconnection because they are used for routing. For example, an MSC receiving a message from a BSC must be able to determine which of its BSCs sent the message so that it can respond to that BSC. That is, names of objects are used to identify communication paths. This requirement is met by including the name of the MSC to which a BSC is connected in the name of the BSC and the name of the BSC to which a BTS is connected in the name of the BTS, i.e., BSC and BTS identifiers have the following forms, respectively. *n* and *m* are digits.

```
bsc(msc(l), n)
bts(bsc(msc(l),n),m)
```

The second category of data type specification represent data types that denote protocol primitives. These types contain elements that represent primitives or Protocol Data Units (PDUs), functions that extract parameters from PDUs, and a function that can compare primitives of the same type. Most of the primitives were specified as elements of two main types. This approach resulted in a few large data type specifications which are less conducive to readability than many smaller ones. In retrospect, we believe the primitives should have been grouped according to the protocols they belong to (i.e., one data type for each protocol) because this would enhance readability of the specification.

The third category of data type specification represents two kinds of complex data types: *Record* and *Set*. The *Record* data types are used to represent HLR and VLR records, and are specified as tuples. The specification of the *Record* data type is illustrated by the specification of HLR records given in Figure 36, lines 20-28.

The *Set* data types are used to represent HLR and VLR databases (see Section 5.4.6 and Section 5.4.7) and for housekeeping duties such as holding object identities that have already been assigned. This complex data type is specified as actualizations (lines 31-37 in Figure 36) of a generic *Set* type (lines 1-17), i.e., there is a specification of a *Set* type whose content type can be actualized (or instantiated) by specific types.

```
type Set is Boolean
   formalsorts Element
   formalopns
      _ eq _ : Element, Element -> Bool
      _ neq _ : Element, Element -> Bool

   sorts Set
      opns
         {} : -> Set
         Insert: Element,Set -> Set
         (*...other Set operations *)
      eqns
         forall x,y:Element, s:Set
         ofsort Set
            Insert(x,Insert(x,s)) = Insert(x,s);
            (*definitions of other Set operations *)
      endtype (* Set *)

(*Specification of HLR record type*)
Type HlrRecord is IMSI,MSCId,Boolean
sorts HlrRecord
opns
  hlrRec: IMSI, MSCId -> HlrRecord
_ == _ : HlrRecord,HlrRecord -> Bool
  _ <> _ : HlrRecord,HlrRecord -> Bool
eqns
   ...
endtype(*HlrRecord*)

(*Specification of set-of-HLR-record type*)
type HlrRecSet0 is Set actualizedby HlrRecord using
sortnames
  HlrRecord for Element
 opnnames
   == for eq
  <>  for neq
endtype (*HlrRecSet0*)
```

Figure 36 ADT specification of record and set types.

CHAPTER 6

Applications of LOTOS Specifications

6.1 Introduction

The major motivations for using LOTOS to describe systems are the possibility to produce precise and unambiguous specifications as well as the feasibility to use tools to help in analyzing these systems. The executability of LOTOS specifications offers many advantages. Execution enables to determine scenarios allowed by a specification and those that are prohibited. It provides a means for improved understanding as well as for validation. In this chapter, we report on preliminary results on using LOTOS specifications to: 1) derive scenarios, and 2) recognize scenarios. We will also discuss briefly how LOTOS specifications may be used in other development activities.

A LOTOS specification may be executed in a number of ways (we call them *execution modes*.) We will show how the *step-by-step* and *symbolic execution* modes could be applied to produce and recognize scenarios. The tools that we used to perform step-by-step and symbolic execution are called ISLA [Hus88] and SELA [Ash92], respectively. These tools are members of the toolkit ELUDO (Environnement LOTOS de l'Universite d'Ottawa) [STS94]. Many other LOTOS tools exist (e.g., LOLA, SMILE, CEASAR) but our work concentrated on ELUDO.

This chapter is organized as follows. Section 6.2 presents a discussion on automatic generation of scenarios. Subsection 6.2.1 compares LOTOS execution traces with scenarios. Subsections

6.2.2 and 6.2.3 investigate how step-by-step and symbolic execution modes, respectively, can be used to generate *traces* (defined below) that correspond to scenarios. Section 6.3 outlines ways to use specifications to recognize scenarios that may be at slightly different abstraction levels, and Section 6.4 gives brief descriptions of other applications for LOTOS specifications.

6.2 Scenario Generation from LOTOS Specifications

Recall that the method given in Chapter 4 seeks to derive LOTOS specifications from scenarios. The goal of scenario generation is to re-generate scenarios, or paths in the Integrated Scenario Model (ISM), from specifications. Specifically, we want to derive sequences of events that are causally ordered, i.e., sequences that describes specific behaviors of a system. The concept of execution trace provides a direction to pursuit this goal.

6.2.1 Execution Traces and Scenarios

In LOTOS theory, the *environment* of a process[1] *P* consists of a collection of processes with which *P* can potentially interact, together with an unspecified, possibly human, external observer process who watches *P* and records, one at a time, the names of events performed by *P*. Thus, whenever *P* interacts with the external observer, it performs an observable event. If *P* performs two observable events simultaneously, then the observer may record them in any order. The sequence of event names recorded by the observer process in a certain time interval forms a *trace* or behavioral history of the process *P*. More precisely, a trace is a finite-length sequence of observable atomic events [Gal89]. Note that there are two important implications resulting from the way observable events are recorded. One is that traces are linear (i.e., they do not show alternative behaviors), and the other is that concurrent behaviors tend to have many possible traces because events from different execution threads may interleave in numerous ways. In fact, the latter implication is the one that can cause state explosion (Section 3.3.2).

The trace concept is intuitively related to the scenario concept, but they are not identical. In this section, we relate the two by examining their similarities and differences with respect to Definitions 1.1 and 1.2. Our goal is to determine if traces could be considered as scenarios.

1. Note that a LOTOS specification is just a special process.

LOTOS traces and scenarios are similar in that they are made up of events. In LOTOS, an event is represented by a denotation consisting of a gate and a list of experiments[1] or by the internal event *i*. In scenarios, events are described in natural language and contain the information listed in Definition 4.1. According to this definition, the description of an event must include the identities of the objects that are involved in its occurrence as well as the roles played by these objects (e.g., as senders or receivers of messages). Role is not an inherent concept in LOTOS (in the sense the no language construct was included in the design of the language for the special purpose of denoting roles), but if appropriate event structures are employed, it is possible to denote an object's role in an event. For example, the event structure used in the GSM specification specifies identities of objects and their roles (see "Event Structures" on page 94).

Descriptions of events in a scenario must also specify data that are exchanged among the objects involved, as well as the direction of these exchanges. In LOTOS, the direction of information exchange can be conveniently represented; the object that offers data in a synchronization can be considered the source, and the object which receives data as a result of a synchronization can be thought of as the recipient. The denotation of roles of objects also indicates the direction of information flow.

LOTOS traces and scenarios are also similar in that their events are ordered, although their ordering criteria are different. LOTOS traces are ordered temporally (i.e., by the sequence in time in which events are recorded by the observer process), whereas events in scenarios are ordered by causality. This means that events in a trace are functionally unrelated if they come from different, parallel behaviors[2]. In a scenario, all events are logically related, i.e., there is a causal relationship between each event (except the first event) and previous events in the sequence. This difference seems to occur only in traces from specifications that describe concurrent systems. In non-concurrent systems, only one system service can be invoked at a time and therefore all events collected during the execution of a service are functionally related, e.g., they co-operate to deliver that service. Thus, the order of the events in such a trace coincides with that of the corresponding scenario. In concurrent systems, a number of services may be invoked simultaneously, and events from different execution threads are recorded into

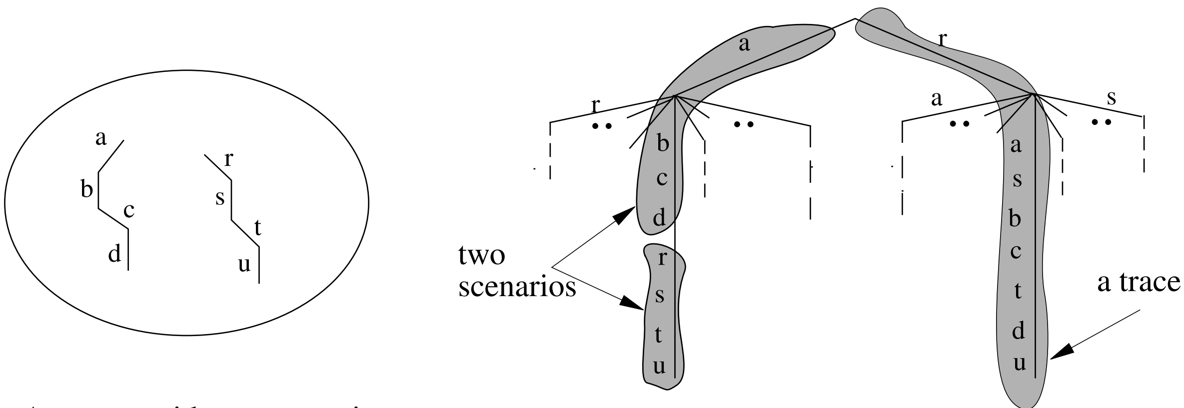
1. See Section 3.2.2 for definition of the term 'experiment'.
2. By behaviors, we mean courses of events each triggered by a service invocation, i.e., scenarios.

the same trace. Therefore, such traces may contain events that are not causally related. Although the *relative* order of events of *one* execution thread still matches the order of events in the corresponding scenario, the trace as a whole does not correspond to any (single) scenario of the system.

However, a small number of the possible traces of a concurrent system do have *segments* that correspond to scenarios. These traces are ones in which the events from individual execution threads are captured consecutively with no intervening events from other threads. That is, such traces correspond to chains of scenarios. Figure 37 illustrates the difference between scenarios and traces of a concurrent system.

According to Section 1.2.2, scenarios are class concepts in the sense that they describe behaviors of classes of objects rather than of specific objects, i.e., specific object identities and data values are not given. Instances of scenarios are created by assigning specific identities and values to the objects and parameters involved in the scenarios. On the other hand, no notion of class or instance is associated with LOTOS traces. By definition, a trace is simply a temporally-ordered sequence of observable events [Gal89]. Another difference between LOTOS traces and scenarios is that the former are always linear sequences whereas the latter may contain branches (as we described them in Section 1.2.2).

The conclusion from this comparison of trace with scenario is that the two concepts are similar in some respects but differ in others. From the consideration of their differences, we can stipulate at least two conditions that must be satisfied in order for LOTOS traces to correspond to scenarios. The criterion for correspondence is that the two concepts must convey exactly the same information. First, the specification from which a trace is derived must use event structures that denote all information required to describe an event completely. Secondly, the trace must be captured when the specification does not exhibit concurrent behaviors, e.g., when only one system service has been invoked.



A system with two scenarios or threads of execution.

A partial behavior tree for the system shown on the left.

Figure 37 Comparisons of scenarios and traces.

6.2.2 Scenario Generation by Step-by-step Execution

Step-by-step execution is carried out by computing all possible events at the current system state. Then one event is selected, usually by the user, to be ‘executed’. The selection of an event corresponds to traversing a transition from the current state to a new state in the LTS of the specification, and represents the execution of the event. The possible events at the new state are then re-computed, and an event is again selected for execution. The process may continue until no further event is possible, i.e., until an explicit stop or a deadlock is reached. Executed events and the order in which they occurred are recorded in a trace by the environment. In this mode of execution, the environment (or the specification user) may be required to supply values for variables whose values cannot be derived from the specification. [HH88] discusses step-by-step execution in detail.

We utilized the ISLA [HH88] tool to execute the GSM specification and to generate scenarios. ISLA enables step-by-step execution to be performed on specifications or on specific processes of a specification. During execution, the user may backtrack to any event in the trace and execute different branches (or the same branch with different values), thus producing (portions of) the LTS of the specification.

Because specific values must be used in this execution mode, variables in the specification that represent data or object identities are assigned values. Therefore, step-by-step execution traces that have the appropriate event structures and event order, correspond in fact to *instances* of scenarios (as oppose to scenarios).

A trace generated by ISLA is shown in Figure 38. This trace corresponds to the Location Update scenario described in Section 2.4.3. The trace shows a mobile station (MS) sending a channel request over the radio interface (line 2). The reason for the request is to do location update, represented by `LOCUPD`. A BSC identified by `bsc(msc(1),1))` then orders a BTS (identified by `bts(bsc(msc(1),1),1)`) to activate a channel named `dchanF2` (line 4). Once it receives an acknowledgment from the BTS (line 5), the BSC sends an immediate assignment indication with the assigned channel identifier (`dchanF2`) to the MS (lines 6-7). The first thing the MS does on the new channel is send a SABM to establish a link layer connection for signalling messages (lines 9-12). The MS then goes into dedicated mode (line 16) and sends a location update request to the MSC (line 17). In Appendix A, we give more traces.

```
(* Tree generated by Isla *)

1    choice rnd=rand2
2    ! ri !RIL3RR !bts(bsc(msc(1),1),1) !RACH !rand2 ?fn,fn=fn4 !ChanReq !LOCUPD;
3    | | hidden abisi !RSM !bts(bsc(msc(1),1),1) !bsc(msc(1),1) !rand2 !fn4 !ChanReqd !LOCUPD;
4    | | | abisi !RSM !bsc(msc(1),1) !bts(bsc(msc(1),1),1), !fn4 !ChanActivation ?dchan,dchan=dchanF2 [noDChan <> dchan]
5    | | | | hidden abisi !RSM !bts(bsc(msc(1),1),1) !bsc(msc(1),1) !fn4 !ChanActivationAck;
6    | | | | | hidden abisi !RIL3RR !bsc(msc(1),1) !bts(bsc(msc(1),1),1) !fn4 !ImmdAssignment;
7    | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),1),1) !PAGCH !fn4 !ImmdAssignment !dchanF2;
8    | | | | | | | internal exit(dchanF2,bsc(msc(1),1))
9    | | | | | | | | hidden ri !RIL3MM !noTMSI !bts(bsc(msc(1),1),1) !dchanF2 !SABM !MS_INFO;
10   | | | | | | | | | hidden abisi !RSM !bts(bsc(msc(1),1),1) !bsc(msc(1),1) !noTMSI !dchanF2 !LOCUPD !EstablishInd;
11   | | | | | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),1),1) !noTMSI !dchanF2 !UA !rand2;
12   | | | | | | | | | | | hidden ai !BSSMAP !bsc(msc(1),1) !msc(1) !bts(bsc(msc(1),1),1) !noTMSI !dchanF2 !LOCUPD
!CompleteLayer3Info;
13   | | | | | | | | | | | | internal exit
14   | | | | | | | | | | | | | internal exit(rand2, dchanF2)
15   | | | | | | | | | | | | | | internal exit(dchanF2)
16   | | | | | | | | | | | | | | | hidden MS_SAP1 !dedicated !noTMSI !LOCUPD !dchanF2 !bts(bsc(msc(1),1),1) !noCksn;
17   | | | | | | | | | | | | | | | | hidden ri !RIL3MM !dchanF2 !bts(bsc(msc(1),1),1) !LUReq(dn(hlr1, 9) !noTMSI !noCksn) [true];
```

Figure 38 Trace generated by ISLA showing the location update scenario

One drawback of ISLA is that it requires human assistance. When executing large specifications, this can be very time-consuming and frustrating for the user. It is especially so because each time a modification is made to a specification, it has to be re-executed from the beginning. This is the main reason why our process in Chapter 4 strives to prevent as many defects as possible in each iteration (by formalizing scenarios, etc.). It is also a reason for minimizing the number of iterations (and executions). In the production of the GSM specification, we performed the analysis and specification cycle (Figure 13) several times. In executions subsequent to the first one, we sped up execution by removing some of the behaviors that had already been exercised previously, i.e., certain behavior expressions were 'commented out'.

As a validation tool, step-by-step execution is a technique that can increase confidence in the correctness of a specification, but it does not allow to conclude with absolute certainty that a specification is correct. It permits only to empirically check the specified behaviors against the intended ones. This mode of execution may be thought of as *simulation* because it allows to play out the behaviors of a system.

6.2.3 Scenario Generation by Symbolic Execution

Symbolic execution is another mode of execution supported by the ELUDO toolkit. Whereas step-by-step execution exercises a path through a specification with a set of data values, symbolic execution uses symbols to represent arbitrary values for variables whose specific values cannot be derived from the specification. This has the effect of exploring all behavior paths or traces of a specification simultaneously. Its output, therefore, is a behavior tree. The computation of each path in this tree may generate a *path condition* whose solutions determine the feasibility of the behavior represented by the path. For example, the path condition for the behavior expression

```
[x==y] ->
   g !x;
   ( g ?val1:Boolean; stop
     ||
    g ?val2:Boolean [not(val2)]; stop
   )
```

is the set `{x==y, val1=not(val2)=true}`. The abstract data type evaluator used to solve such conditions must be able to determine whether the expression is satisfiable by some values of the variables. However, since LOTOS permits undecidable and unsolvable conditions, complete computation of some conditions may not be possible. For tools, this presents the problem of whether or not to output paths having such conditions.

Thus, a behavior tree produced by symbolic execution may not correspond to the Integrated Scenario Model (Section 4.2.4) for two reasons. One is that it may contain paths that represent impossible behaviors, i.e., those paths that are included in the tree only because the ADT evaluator cannot solve their path conditions. The second reason is due to the LOTOS representation of concurrent scenarios by interleaved events (see Section 3.3.1). Parallel scenarios, which are each represented by a path in the ISM, are represented in the behavior tree as interleaved sequences. The behavior tree contains all possible interleaved sequences, and none of these sequences are scenarios. A small number of the sequences may have segments that do correspond to scenarios. They are the segments in which events from the same scenario occupy consecutive positions (see Figure 37). However, because symbolic values are used, those segments that correspond to scenarios are truly scenarios and not scenario instances as step-by-step execution traces are.

SELA is the member of the ELUDO toolkit that performs symbolic execution. It permits execution of whole specifications, of specific processes in the specifications, or it can be started from a system state reached by step-by-step execution. Since behavior trees may have large depths and widths, possibly even infinite (see Figure 10), this tool provides options for limiting tree sizes.

Figure 39 contains a partial behavior tree of the LOTOS process that represents mobile station. The process is given in Figure 28. The tree was generated by SELA with the width and depth limits set at 5 and 7, respectively. A corresponding graphical representation is given in Figure 40, the numbers in brackets relates the transitions to the line numbers in Figure 39.

```
bh0   * 1 [noBts <> BTSId&0] ->[noTMSI <> TMSI&0] ->choice Service%1
bh1   * | 1 i {enable: exit (Service%1) }
bh2   * | | 1 choice rnd%1
bh3   * | | | 1 ri !RIL3RR !BTSId&0 !RACH !rnd%1 ?fn=FrameNum@3 !ChanReq !Service%1Service%1
bh4   * | | | | 1 ri !RIL3RR !BTSId&0 !PAGCH !FrameNum@3 !ImmdAssignment ?dchan=DChannel@4
bh5   * | | | | | 1 ri !LAPDm !TMSI&0 !BTSId&0 !DChannel@4 !SABM !MS_INFO [...]
bh6   * | | | | | | 1 ri !LABDm !var_46 !var_47 !var_48 !UA ?rand=RandNum@5 ==> continue
      * | 2 ri !RIL3CC !BTSId&0 !MSISDN&0:MSISDN !PAGCH !PageCmd
bh7   * | | 1 i {enable: exit (PAGING_RESP) }
bh8   * | | | 1 choice rnd%2
bh9   * | | | | 1 ri !RIL3RR !BTSId&0 !RACH !rnd%2 ?fn=FrameNum@4 !ChanReq !PAGING_RESP
bh10  * | | | | | 1 ri !RIL3RR !BTSId&0 !PAGCH !FrameNum@4 !ImmdAssignment ?dchan=DChannel@5
bh11  * | | | | | | 1 ri !LAPDm !TMSI&0 !BTSId&0 !DChannel@5 !SABM !MS_INFO ==> continue
      * 2 [noBts <> BTSId&0] ->[noTMSI == TMSI&0] ->i {specified explicitly}
bh12  * | 1 i {enable: exit (LOCUPD) }
bh13  * | | 1 choice rnd%3
bh14  * | | | 1ri !RIL3RR !BTSId&0 !RACH !rnd%3 ?fn=FrameNum@2 !ChanReq !LOCUPD
bh15  * | | | | 1 ri !RIL3RR !BTSId&0 !PAGCH !FrameNum@2 !ImmdAssignment ?dchan=DChannel@3
bh16  * | | | | | 1 ri !LABDm !TMSI&0 !BTSId&0 !DChannel@3 !SABM !MS_INFO [...]
bh17  * | | | | | | 1 ri !LAPDm !var_152 !var_153 !var_154 !UA ?rand=RandNum@4 ==> continue
      * 3 [noBts == BTSId&0] ->choice BTSId%1
bh18  * | 1 [BTSId%1 <> noBts] ->ri !RIL3RR !BTSId%1 !BCCH !Broadcast_info
bh19  * | | 1 ri !RIL3RR !BTSId%1 !FCCH !F_Burst
bh20  * | | | 1 ri !RIL3RR !BTSId%1BTSId%2 !SCH !S_Burst [...]
bh21  * | | | | 1 [noBts <> var_182] ->[noTMSI <> var_180] ->choice Service%2
bh22  * | | | | | 1 i {enable: exit (Service%2BTSId%1) } ==> continue
      * | | | | | 2 ri !RIL3CC !var_182 !var_181:MSISDN !PAGCH !PageCmd ==> continue
      * | | | | 2 [noBts <> var_182] ->[noTMSI == var_180] ->i {specified explicitly}
bh23  * | | | | | 1 i {enable: exit (LOCUPD) } ==> continue
      * | | | | 3 [noBts == var_182] ->choice BTSId%2
bh24  * | | | | | 1 [BTSId%2 <> noBts] ->ri !RIL3RR !BTSId%2 !BCCH !Broadcast_info ==> continue
      * | | | | | 2 [BTSId%2 == noBts] ->i {specified explicitly} ==> continue
      * | 2 [BTSId%1 == noBts] ->i {specified explicitly} ==> again bh21[...]
```

Figure 39 Behavior tree generated by SELA from the specification of MS.

The tree shows the behaviors at the radio interface of one MS starting at the idle state, the state in which it is not in a connection with the network. The highlighted path focuses on one scenarios from this state. The first transition of this path shows that the MS may access a service if it is roaming in a service area (i.e., in a cell) and has registered with the network. After receiving input from the environment on the service (represented by the symbolic value `Sevice%1`) to perform, the MS executes an internal event (represented by `i`). Then it picks a random number to transmit with the request for a radio channel. A channel (represented by the variable `dchan`) is immediately assigned by the network. The MS then sends a link layer SABM on this channel to establish a link layer connection for signalling messages. Although

SABM is a PDU which belongs to the transmission plane, it is included in the specification because of its important role in resolving contention that can result due to simultaneous multiple accesses. The Unnumbered Acknowledgment (UA) to SABM from the network to the MS carries the random number which the MS transmits with its channel request. An MS will leave the channel if the received random number does not match the one it has sent (see Section 2.4.2 for a more detailed description).

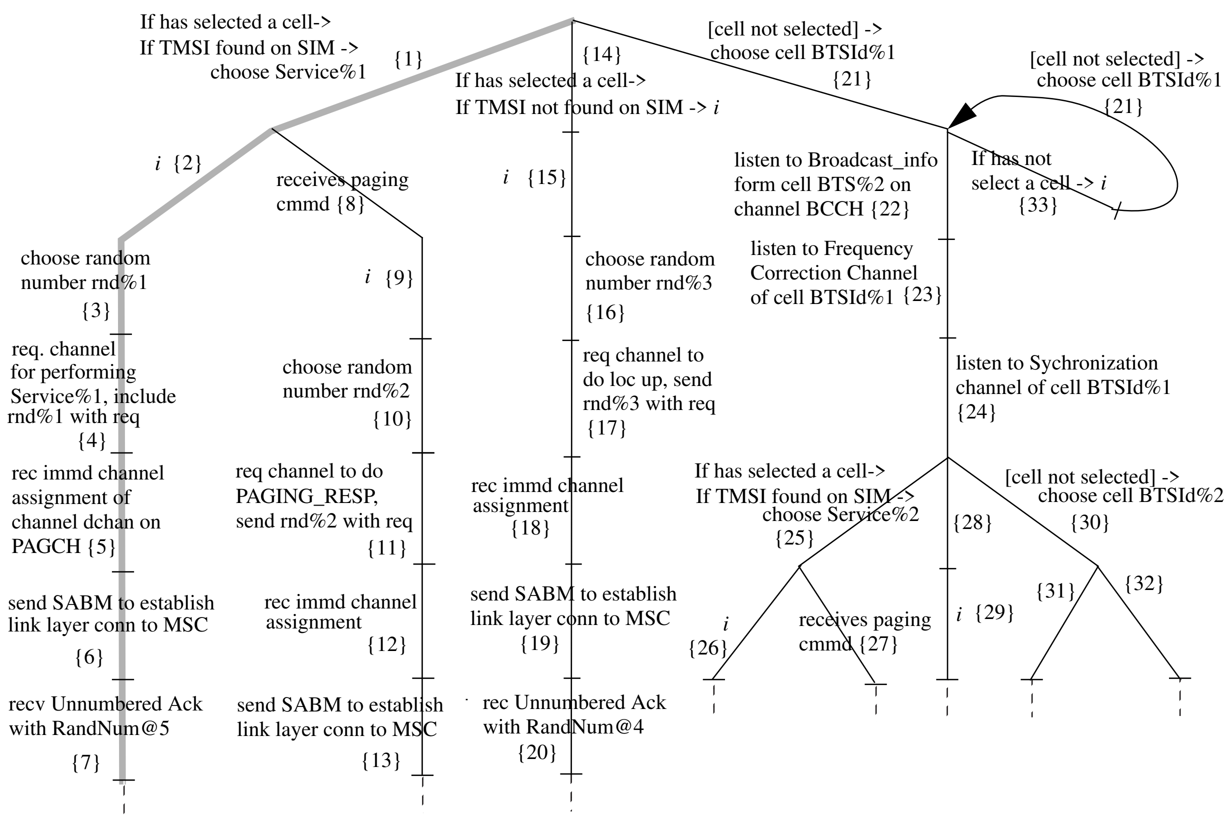


Figure 40 Graphical representation of the partial behavior tree of the MS specification

In principle, symbolic expansion can be used to detect missing branches/scenarios in specifications. Because all traces of a specification are generated, scenarios that have not been specified can be detected by their absence in the behavior tree. In reality, this is applicable only with small and simple specifications for which it is possible to generate the complete behavior trees. For specifications of systems of real size, it is not applicable. In these cases, scenarios represented by a specification may not show up in its behavior tree because they are outside of the set tree width and depth. Likewise, symbolic execution can potentially reveal incorrectly linked scenarios. But due to the fact that it may produce infeasible paths, detection of these errors is difficult.

6.3 Scenario Recognition Tests

The executability of LOTOS specifications also enables to determine whether or not certain scenarios and instances of a scenarios are represented by a specification. This determination does not even have to involve scenarios and specifications that are at exactly the same abstraction level. We believe it is more common that a collection of scenarios and a LOTOS specification do not describe a system with the same degrees of detail and completeness. For example, one of them may include optional events while the other does not. This may happen if the specification and the scenarios were derived by different people.

Thus, there is usefulness in such a determination, which we call such a test *recognition test*. The purpose of recognition test is to confirm that scenarios have been specified, irrespective of the abstraction level. Note that it is much more difficult to perform this test for scenarios because for every variable in a scenario, we must check that the value domain that could be used to instantiate the scenario is the same as the domain that could be accepted by the specification. But since the equivalence relationship between scenario and specification we seek is very ‘loose’ anyway, we do not distinguish between scenarios and scenario instances in this section.

If we desire to deal only with scenarios and specification at the same abstraction level, then the LOTOS testing technique presented in Section 3.3.3 can be used. In the ‘usual’ way of testing a specification, a test passes only if the specification has a trace that is identical to the test. In a recognition test, we want the test to pass even though the trace only approximates the test.

One way to perform this recognition test is simply to generate traces from the specification and comparing them to the scenario. If at least one trace can be generated that corresponds to a scenario, then we can declare that this scenario is contained in the specification.

Alternatively, we can use variants of the existing LOTOS testing technique. One variant still follows the testing procedure outlined in Section 3.3.3, but uses an extra process, let us call it a dummy process, to synchronize with events that one specification offers but the other does not. That is, one of the following setups can be used:

S = SpecUnderTest[g1,g2] || (Scenario[g1,g2] || Dummy_Process[g1,g2])

S=(SpecUnderTest[g1,g2] || Dummy_Process[g1,g2]) || Scenario[g1,g2]

Another variant of the testing procedure uses the selective synchronization operator |[]| instead of the full synchronization operator. This variant is applicable if the events, for instance, *SpecUnderTest* contains events on other gates that are not in *Scenario*. In such a case, only the gates that are common to both processes are listed in |[]|.

```
S = SpecUnderTest[g1,g2,g3] |[g1,g2]| Scenario[g1,g2]
```

Figure 41 gives a concrete, but simplified, example to illustrate the technique. In this example, the mobile station (MS) can communicate with its human user at the man-machine interface *mmi* and with the PLMN at the radio interface *ri*. The specification under test has been modified to the process *SpecUnderTest* and the scenario that is used to test this specification is specified by the process *Scenario*. The scenario (or test case) describes the user turning on the MS (line 43) and dialing a number (line 44); and the MS then sending a service request to the network (line 45), to which the network replies with an accept message (line 46). The specification describes this mobile-originated call in more detail. It shows the MS sending a request for radio channels (line 34) and the network's reply (line 35) before displaying the MS sending a service request message (line 36). Thus, the path in *SpecUnderTest* that *Scenario* is to test has two more events than the scenario. These two 'extra' events are absorbed by *Dummy_Process* and the whole expression can be executed to the end.

```
specification test_execution[mmi,ri]:noexit

  (* The ADT specification is moved out of the specification
   * under test and is placed here. *)

  SpecUnderTest[mmi,ri]()
  ||
  ( Test_Case[mmi,ri]()
    |||
    Dummy_Process[ri]
                                            (*can have as many instances of this process as required*)
  )
  where

  Process Dummy_Process[ri]:noexit:=
    consume[ri](ChanReq)        (*can consume as many events as needed*)
    >> consume[ri](ChanAssign)
    >> TEST_PASSES;
       stop

    where

    Process consume[g](parm1:ParmType]:exit:=
      g !parm1;
      exit
      (*[]...can have as many alternatives as required *)
    endproc (*consume*)
  endproc(*Dummy_Process*)

  process SpecUnderTest[mmi,ri](...):noexit:=
    mmi !Power-on;                     (*User turns power on*)
    mmi !Dial ?calledNum:DNSort;       (*user dials number*)
    ri  !ChanReq;                      (*MS requests radio channels*)
    ri  !ChanAssign;                   (*PLMN assigns channels*)
    ri  !CMServReq;                    (*MS sends CM Service Request*)
    ri  !CMSerAccept;                  (*PLMN accepts CM Service request*)
    stop
    (*[] other behaviors of the mobile station*)
  endproc(*SpecUnderTest*)

  Process Scenario[mmi,ri](...):noexit:=
    mmi !Power-on;                     (*User turns power on*)
    mmi !Dial ?calledNum:DNSort;       (*User dials number*)
    ri  !CMServReq;                    (*MS sends CM Service Request*)
    ri  !CMSerAccept;                  (*PLMN accepts CM Service request*)
    stop
  endproc (*Scenario*)
endspec
```

Figure 41 A LOTOS test case execution technique.

The disadvantage of these techniques is that they require *apriori* knowledge of the behaviors described by the specifications to be tested. In fact, the technique illustrated in Figure 41 will not work if the process `Dummy_Process` is not coded with full knowledge of the extra events involved. One of the problems is that non-determinism which occurs when `Dummy_Process` and `Scenario` are capable of synchronizing on the same action. In such a case, `Dummy_Process` may absorb the events that should instead synchronize with `Scenario`, and thus the test does not work as intended. The example shown in Figure 41 is a particularly difficult one because the events to be skipped are on the same gates as other events of the scenario. If they were on distinct gates (or if their event structures are different), then `Dummy_Process` could simply absorb the extra events on these different gates, and the possibility of non-determinism would not exist.

6.4 Other Uses of LOTOS Specifications

There exists many other possible applications of LOTOS specifications. In the subsections to follow, we briefly discuss some of them.

6.4.1 Design Documentation and Communication

Since LOTOS enables unambiguous representations, there exist obvious advantages to using this FDT to document systems and for communication. An application of LOTOS specifications that is yet untried but holds great potential is to use them as standardization documents. The level of detail and granularity conveyed by LOTOS specifications is difficult to create and maintain in informal documentation forms. Furthermore, since LOTOS specifications allow to describe systems in an integrated manner, as opposed to descriptions with partial models such as scenarios, they are more maintainable and evolvable. Moreover, the executability of the language may allow automatic comparisons or consistency checking among successive specifications of evolving standards.

6.4.2 Prototyping

A LOTOS specification may be considered an executable prototype of the system it describes. All the benefits associated with prototyping are present. A specification may be used to identify new requirements, evaluate and experiment with design alternatives, and give early exposure to high-risk design items. For instance, specifications of combinations of services may be used to determine if they interact in unintentional ways, a problem known as feature interaction. [Fac95] gives a method of using LOTOS to detect feature interactions.

6.4.3 Test Case Generation

LOTOS enables to systematically derive tests from specifications for black-box testing. We note that the main difference between traces and test cases (or more specifically, test inputs) is that the former may consists of events from a number of execution threads while a test input typically contains only events from one execution thread. A number of test generation methods have been proposed. In [GuL89], a semi-formal approach is described for deriving test cases from LOTOS specifications. The method involves transforming a LOTOS specification into a symbolic execution tree, and converting this tree into a state diagram. Test cases can then be derived by generation methods for FSMs. In [Sch93], a method for data flow-based test derivation is proposed. By performing data flow analysis of a LOTOS specification, all associations among 'inputs' and 'outputs' are revealed and thereby criteria for test selection can be derived. [Wez95] presents an algorithm for deriving so-called *conformance testers* from LTSs. This method is based on [Bri88]'s theory of canonical testers. According to this theory, a process *P1* "conforms" to a process *P2* if *P1* deadlocks "less often" than *P2* with respect to traces common to *P1* and *P2*. An implementation under test (IUT) conforms to a specification if no unexpected deadlocks occur when the IUT is run concurrently with a process, called *canonical tester*, consisting of traces derived from the specification. This area of research holds great promises and is the subject of continuing investigation.

6.4.4 Learning Tool

LOTOS specifications and the specification process are useful tools for learning the designs and behaviors of systems. As was shown in Chapter 4, the process of specifying a system requires the specifier to think and analyze (and therefore, learn) about the system in such a way as to relate together different partial descriptions and produce a unified picture. In this way, designers are forced to learn how different scenarios of system usage fit together. This piecing together of scenarios also helps to reveal inconsistencies among and incompleteness in these partial models.

The executability of the language also has useful implications for learning. Execution of a specification can be regarded as a simulation of a system. For instance, step-by-step execution animates a specification's behaviors and the requirement for human assistance permits one to actively learn the interaction between a system and its users. In fact, more sophisticated simulation tools that can display the actual physical components and behaviors of a system can be envisioned. Figure 42 shows the interface of an animation tool for LOTOS specifications being investigated at the University of Ottawa. The figure depicts a synchronization among a test process, a process representing a Base Transceiver Station (BTS) named `bts1`, and a sub-process of a process that represents a Base Station Controller (BSC) named `bsc1` on an event that denotes the sending of a Location Update Request from `bts1` to `bsc1`. The parameters of the message state that the MS tow which the message pertains to has not been allocated a Temporary Mobile Station Identifier (see “Location Update Procedure” on page 29) nor an encryption key (see “Security Management” on page 30).

Although the formal nature of LOTOS is beneficial in that it provides unambiguous notation and facilitates formal reasoning, it decreases readability and understandability of specifications. LOTOS specifications are generally considered ‘dense’ and difficult to understand. A possible solution to facilitating comprehension is to automatically generate graphical representations of the specifications. For LOTOS, a graphical notation, called G-LOTOS [GLOTOS], was standardized with the language for this purpose. Of course, realistic representations such as the one shown in Figure 42 also ease and enhance understanding.

Scenarios facilitate understanding by allowing one to concentrate on narrow aspects of a system’s behavior at a time. Thus, another way to aid understanding of specifications is to generate scenarios from the specifications.

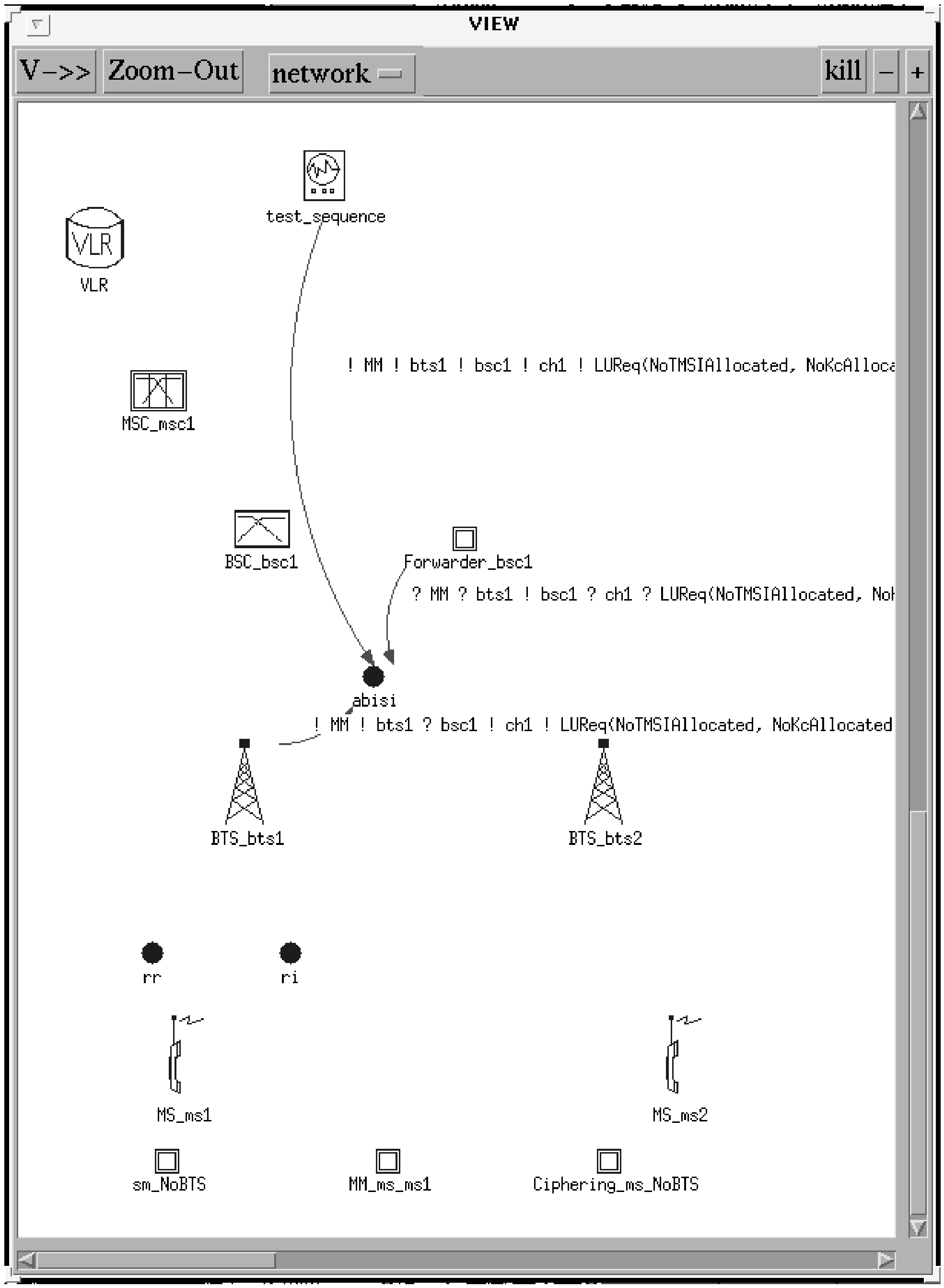


Figure 42 The display of an experimental animation tool for LOTOS specifications developed at the University of Ottawa.

6.4.5 LOTOS-Based System Development

A LOTOS-based development methodology that exploits the executability of specifications is based on the concept of gradual refinements. This methodology is the outcome of the European LOTOSphere projects [LOT92]. Briefly, the gist of the method is as follows. First, user requirements are transformed into a formal specification. This specification, generally being very abstract, is then successively refined in a number of steps. The constraint oriented style is used at the most abstract level, while the monolithic and state-oriented styles are considered the most concrete. In each step, tool-guided correctness-preserving “stye transformations” are used to refine the specification. Once sufficient details have been introduced, the LOTOS specification is converted into an implementation. Currently, there exists tools (such as LOLA [QPF88], CEASAR [Gar94]) that are capable of generating C codes for a restricted type of LOTOS specifications. Figure 43 illustrates the process. The challenges of this methodology are the transformations, which must ensure consistency among successive specifications.

In the context of GSM, and with technical standards in general, one may not always be required to start with informal requirements as more concrete descriptions of the system are already be available. In such cases, it is only logical to base development on these descriptions.

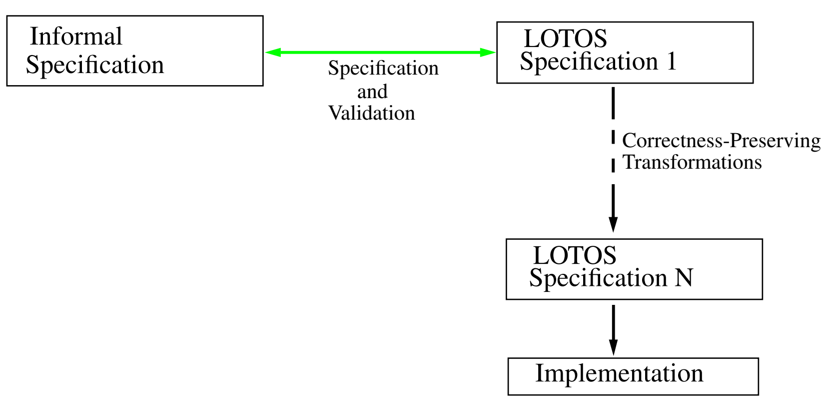


Figure 43 A LOTOS-based development methodology.

CHAPTER 7 Contributions and Future Work

In this chapter we summarize and evaluate the results that have been reported in this thesis. Section 7.1 contains an overview of our contributions. In Section 7.2, we discuss some possible future research areas, and outline an integrated specification and validation environment for LOTOS specifications.

7.1 Contributions

The major contributions of this thesis are:

- Definition of scenario

  Most definitions of use cases and scenarios that exist in the literature leave room for interpretation. For instance, they only state that use cases are sequences of events, and do not specify whether they are class concepts nor define the relationships that exist, if any, among events in each sequence (see Section 1.2.2). The definition we presented is a combination of existing definitions and it states precisely all aspects associated with scenarios. Some may not agree with our definition (particularly our view of scenarios as class concepts), but it should be noted that the given definition is a consolidation of existing definitions and therefore it does not differ drastically from them. It was essential that we give as precise as possible a definition of scenarios because they are the driving force of the proposed specification process. We would not have been able to give the specification and scenario generation methods if their definition was left unclear or imprecise.

- Specification Process

  The scenario-oriented specification method presented in Section 4.2 represents an effort to identify and organize the activities associated with producing LOTOS specifications. We believe that writing specifications is one of the most basic and important activities of using LOTOS, and that the analysis and documentation of this process is a prerequisite for adoption by industry. Although the goal of deriving a formal process was not completely achieved, the proposed process is a first step towards establishing a systematic procedure for producing LOTOS specifications.

- LOTOS Specification of a Mobile Communication System

  A LOTOS specification of GSM was generated as part of this thesis (see Chapter 5 and Appendix B). We believe that this formal specification of a system of real size and the techniques and language 'tricks' that were used are interesting in their own right (Section 3.5, Section 5.3.3, Section 5.3.4 and Section 5.4.1). Others have presented specifications of similar systems (e.g., [CaV90] and [OrP90]) but their specifications dealt only with specialized parts of the systems.

- Scenario Generation/Recognition Techniques

  Our comparison of scenarios and LOTOS traces led to the identification of conditions under which traces correspond to scenarios (Section 6.2.1). We also outlined two approaches (using step-by-step execution and symbolic execution) for generating scenario-like sequences of events from LOTOS specifications (Section 6.2.2 and Section 6.2.3) and for using specifications to recognize scenarios that are at slightly different abstraction levels (Section 6.2.3).

7.2 Future Work

The proposed method of transforming collections of scenarios into formal models represented by LOTOS specifications, as presented by this thesis, is based on the formalization and integration of scenarios. The process is rather laborious and needs to be automated in order to be of practical value when dealing with systems of real size. Full automation may be impossible. However, we suggest that further work be directed towards finding and automating, as much as possible, partial techniques for formalizing and amalgamating scenarios.

Our dissection and formalization of the process is a step towards this goal. The given process produces a number of intermediate models that perhaps could serve as the means of going from collections of partial models to LOTOS specifications. Indeed, the automation of some of the steps seems quite promising. For instance, the automatic consolidation and translation to LOTOS of the Architecture Model and the Object Behavior Models (OBMs), and the automatic derivation of the latter from the Integrated Scenario Model (ISM) seems to be only a matter of syntactical manipulation.

The difficulty of automating the proposed specification process appears to be the formalization of scenarios and their integration into the ISM. The work of Buhr *et al.* on utilizing Use Case Maps (UCMs) as the means to constructing high-level system designs, might be illuminating to these difficulties. The work of [ABBL95] seems to be a further step in this direction.

The problem of automatic generation of scenarios from LOTOS specifications is related to the problem of generating tests, with the difference being the criteria that make a sequence of events a scenario or a test case. Similarly, the 'scenario recognition test' execution techniques presented in this thesis are only variants of LOTOS test execution techniques. As such, LOTOS test generation and execution principles need to be investigated to determine their applicability to scenarios.

At the time of writing, LOTOS tools that allow to execute specifications in other modes beside step-by-step and symbolic execution were just becoming available. The applications of these tools for the generation scenario-like sequences appears hopeful but remains to be investigated. Goal-oriented execution tools, which allow to construct a path between two points, seems particularly promising.

References

[Amy94] D. Amyot, “Formalization of Timethreads Using LOTOS”, Master’s Thesis, University of Ottawa, 1994.

[Ash92] P. Ashkar, “Symbolic Execution of LOTOS Specification”, Master’s Thesis, University of Ottawa, 1992.

[BFL95] T. Bolognesi, D. De Frutos, R. Lanyerak, D. Latella, “Correctness Preserving Transformations for the Early Phases of Software Development” in *LOTOSphere: Software Development with LOTOS*, T. Bolognesi and C. Vissers (Editors), Kluwer Academic Publishers, 1995, 161-180.

[BLT90] T. Bolognesi, F. Lucidi, S. Trigila, “From Timed PEtri Nets to Timed LOTOS” in *Protocol Specification, Testing and Verification*, L. Logrippo, R.L. Probert and H. Ural (Editors), North-Holland, 1990.

[BoB87] B. Bolognesi, E. Brinksma, “Introduction to the ISO Specification Language LOTOS” in *Computer Networks and ISDN Systems* 14, 1987, 25-59.

[Bou91] R. Boumezbeur, “Design, Specification, and Validation of Telephony Systems in LOTOS”, Master’s Thesis, TR-91-30, University of Ottawa, 1991.

[Bri88] E. Brinskma, “A Theory for the Derivation of Tests” in *Protocol Specification, Testing and Validation VIII*, North-Holland, 1988, 63-74.

[BSC87] E. Brinskma, G. Scollo, C. Steenbergen, “LOTOS Specifications, their Implementations and their tests” in Protocol Specification, Testing and Verification VI, B. Sarikaya, G. V. Bochman (Editors), North-Holland, 1987, 349-360.

[BuC95] R. J. A. Buhr, R. S. Casselman, A Use Case Map Approach to High Level Design of Object Oriented Systems, Copy Edit Book Draft for Prentice-Hall, 1995.

[Bustard93] D. W. Bustard, A. C. Winstanley, "Making Changes to Formal Specifications: Requirements and an Example" in *Lecture Notes in Comp. Sc.*, Vol. 717, I. Sommerville and M. Paul (Editors), 1993.

[Car95] J. M. Caroll, "Making Use a Design Representation" in *Communications of the ACM*, Vol. 37, No. 12, Dec. 1994.

[CaV90] R. C. Cam and S. T. Vuong, "A Formal Specification in LOTOS of a Simplified Cellular Mobile Communication System" in *Formal Description Techniques II,* S. T. Vuong (Editor), North-Holland, 1990, 485-499.

[CPT92] Lo/WP1/T1.2/N0043/V03, "Catalogue of LOTOS Correctness-Preserving Transformations", T. Bolognesi (Editor), LOTOSphere Project (ESPRIT 2304), 1992.

[DaF95] J. S. DaSilva, B. E. Fernandes, "The European Research Program for Advanced Mobile Systems" in *IEEE Communications*, February 1995.

[Eer94] H. Eertink, "Simulation Techniques for the Validation of LOTOS Specifications", Ph. D. Thesis, Universitty of Twente, 1994.

[EM85] H. Ehrig, B. Mahr, Fundamentals of Algebraic Specification-1, Springer-Verlag, 1985.

[ETSI92] European Digital Cellular Telecommunication System (phase 2). Mobility Application Part GSM 09.02 Version [4.0.0] June 92.

[Fac95] M. Faci, "Detecting Feature Interactions in Telecommunication System Designs", Ph. D. Thesis, University of Ottawa, 1995.

[FLS89] M. Faci, L. Logrippo, and B. Stepien, "Formal specifications of telephone systems in LOTOS" in *Protocol Specification, Testing and Verification*, Vol. 9, E. Brinksma, G. Scollo, and C. A. Vissers (Editors), North-Holland, 1989, 25-34.

[EM85]

[Gal89] S. Gallouzi, "Trace Analysis of LOTOS Behaviour", Master's Thesis, University of Ottawa, 1989.

[GLOTOS] Document ISO/IEC JTC 1/SC21 N3253.

[Gar94] H. Garavel, CAESAR Reference Manual, Version 3.7, Laboratoire de Genie Informatique, Institure I. M. A. G., Grenoble, France.

[GuL89] D. Guerachi, L. Logrippo, "Derivation of test cases for LAP-B from a LOTOS specification," in *Proc. FORTES '89*, S. T. Vuong (Editor) Vancouver, B.C. Canada, 1989, 489-508.

[HH88] M. Haj-Hussein. "An Interactive System for LOTOS Application (ISLA)", Mater Thesis, University of Ottawa, 1988.

[HL92] M. Haj-Hussein, L. Logrippo, "Specifying and Validating Distributed Algorithms in LOTOS" in *Reseaux et Informatique Repartie*, Vol. 2, No. 2, 1992.

[HaLS92] M. Haj-Hussein, L. Logrippo, and J. Sincennes, "Goal Oriented Execution for LOTOS" in *Proc FORTE '92*, Lannion, France, M. Diaz and R. Groz (Editors)., 1992, 305-320.

[Hoa85] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.

[HSG94] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, C. Chen, "Formal Approach to Scenario Analysis" in *IEEE Software*, 1994, 33-40.

[ISO89] Open System Interconnection, "LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour," International Standard IS 88007.

[IYK90] H. Ichikawa, K. Yamanaka, Y. Kato, "Incremental Specification in LOTOS" in *Protocol Specification, Testing and Verification*, L. Logrippo, R. L. Probert, H. Ural (Editors), North-Holland, 1990.

[Jac92] I. Jacobson et al., Object-oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.

[Led92] G. Leduc, "An Upward Compatible Timed Extension to LOTOS" in *FORTE'91*, K. Parker, G. Rose (Editors), North-Holland, 1992.

[LHF92] L. Logrippo, M. Faci, M. Haj-Hussein, “An Introduction to LOTOS: Learning by Examples” in *Computer Networks ISDN System*, Vol. 23, 1992, 325-342.

[LOT92] Lo/WP1/T1.1/N0045/V04, J. Quemada, G. Yadan, “The LOTOSphere Design Methodology: Basci Concepts”, L. F. Pires (Editor), 1992, LOTOSphere Project (ESPRIT 2304).

[MoJ94] S. Mohan, R. Jain, “Two User Location Strategies for Personal Communications Services,” in *IEEE Personal Communications*, First Quartier 1994.

[MoP92] M. Mouly, M.-P. Pautet, “The GSM System for Mobile Communications,” ISBN 2-9507190-0-7, 1992.

[MaM88] J. A. Manas, T. de Miguel-More, “From LOTOS to C” in *Formal Description Techniques*, pages 79-84, K. J. Turner (Editor), North-Holland, 1988.

[Mil80] R. Milner, “A Calculus of Communicating Systems” in *Lecture Notes in Computer Science*, No. 92, Springer-Verlag, 1980.

[NiH84] R. de Nicola, M. C. B. Hennessy, “Testing Equivalences for Processes” in *Theoretical Computer Science*, 34(1,2), Nov. 1984, 83-133)

[OrP90] F. Orava, J. Parrow, “Algebraic Description of Mobile Networks: An Example” in *Protocol Specification, Testing and Verification*, L. Logrippo, R.L. Probert and H. Ural (Editors), North-Holland, 1990, 275-291.

[PKT92] N. Plat, J. van Katwijk, H. Toetenel, “Application and Benefits of Formal Methods in Software Development” in *Software Engineering Journal*, September 1992, 335-346.

[PiS91] L. F. Pires, W. L. Souza, “Step-wise Refinement Design Example using LOTOS” in *Formal Description Tecniques III*, J. Quemada and E. Vasquez (Editors), North-Holland, 1991, 255-262.

[QAP95] J. Quemada, A. Azcorra, S. Pavon, “The LOTOSphere Design Methodology” in *LOTOSphere: Software Development with LOTOS*, T. Bolognesi, J. van de Lagemaat and C. Vissers (Editors), Kluwer 1995, 29-58.

[QPF88] J. Quemada, S. Pavon, A. Fernandez, “Transforming LOTOS Specifications with LOLA - The Parameterised Expansion” in *Formal Description Techniques*, K. J. Turner (Editor), North-Honland, 1988, 45-54.

[QuF87] J. Quemada, A. Fernandez, “ Introduction to Quantitative Relative Time in LOTOS” in *Protocol Specification, Testing and Verification*, H. Rudin, C. H. West (Editors), North-Holland, 1990, 105-121,

[Rap95] J. Rapeli, “UMTS: Targets, System Concept, and Standardization in a Global Framework” in *IEEE Communications*, February 1995.

[STS94] B. Stepien, J. Tourilhes, J. Sincennes, “ELUDO: The University of Ottawa LOTOS ToolKit” , University of Ottawa, April 1994.

[StL94] B. Stepien, L. Logrippo, “Status-Oriented Telephone Service Specification” in *Theories and Experiences for Real-Time System Development*, T. Rus and C. Rattray (Editors), AMAST Series in Computing, Vol. 2, 1994, 265-286.

[RKW95] B. Regnell, K. Kimbler, A. Wesslen, “Improving the Use Case Driven Approach to Requirements Engineering” in *Proceedings of the 2nd Internatioal Symposium on Requirements Engineering*, York, UK, Marck 1995.

[ReC90] J. Regnier, W. H. Cameron, “Personal Communication Services-The new POTS”, IEEE

[Rud92] S. Rudkin, “Inheritance in LOTOS” in *Formal Description Techniques IV*, K. R. Parker and G. A. Rose (Editors), North-Holland, 1992, 409-424.

[Sch93] H. van de Schoot, “Validation activities for LOTOS based on static data flow analysis”, Master’s Thesis (MI- 93-36), University of Twente, Enschede, The Netherland, 1993.

[VTZ94] W. van Hulzen, H. Tilanus, H. Zuidweg LOTOS Extended with Clocks, to appear in S. Voung (Editor) Formal Description Techniques, North-Holland, 1990.

[Vig91] M. Vigder, “Slicing a System: a Design Methodology with Examples in LOTOS”, Technical Report SCE-91-03, Carleton University, Ottawa, Canada.

[VSS89] C. A. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma, “Specification styles in distributed systems design and verification” in *Thereotical Comp. Sc.*, 1991, 179-206.

[VSA89] C. A. Visser, G. Scollo, R. B. Alderden, J. Schot, L. F. Pires, “The Architecture of Interaction Systems-The Structuring of Distributed Systems”, Lecture Notes, Part 2, Enschede, The Netherlands.

[VPL92] C. A. Vissers, L. F. Pires, J. van de Lagmaat, “LotoSphere: An Attempt Towards a Design Culture” in *LOTOSphere: Software Development with LOTOS*, T. Bolognesi, J. van de Lagemaat and C. Vissers (Editors), Kluwer 1995, 3-25.

[Wez95] C. D. Wezeman, “Deriving Test from LOTOS Specification” in *LOTOSphere: Software Development with LOTOS*, T. Bolognesi, J. van de Lagemaat and C. Vissers (Editors), Kluwer, 1995, 295-317.

Appendix A

Traces Generated by the Tool ISLA

In this section, we give some more traces of the GSM specification generated by the ISLA [HH89] tool of the ELUDO toolkit [STS94]. These traces are captured from step-by-step execution of the GSM specification, in which only one mobile station is created (see Section 5.3.3) to minimize the number of available events at each system state. Because only one mobile station exists, a trace contains only events that are triggered by that MS invoking a service and, therefore, it corresponds to an instance of some scenario (see Section 6.2.1and Section 6.2.2).

The traces given below have been edited to fit the pages and to improve readability. We give a step-by-step narration for one trace and provide time sequence diagrams for the others.

Trace 1: Network Access, Location Update - TMSI Unknown

Figure 44 shows the Network Access and Location Update scenarios, described in Figure 2.4.2 and Figure 2.4.3, respectively. A corresponding time sequence diagram is given in Figure 45.

In this particular scenario, the mobile station, identified by the directory number *ms4*, is newly created (line 2). It tunes to the BroadCast CHannel and listens in for cell identification, Random Access CHannel (RACH), and other broadcast information (line 3). It then sends a channel request, included with this request is a random number (*rand2*) and the reason for the request which is location update (line4). A BSC, named *bsc(msc(1,2))*, receives the request (line 5), and decides on an Immediate Assignment. The identities of the granted channels (*dchanF2*) are transmitted to *ms4* on the *common* Paging Access Grant CHannel (PAGCH). Note that there is a small probability that other mobile stations, who have requested radio channels and are listening on the PAGCH, might mistakenly interpret the paging as intended for themselves. Thus, more than one MSs might start transmitting on the *dchanF2*, resulting in a collision of messages. The first message *ms4* sends on the new channel is its terminal information (line 12). In response to the first message on the allocated channel, the BTS *bts(bsc(msc(1,2),1)* sends back the random number (line 13). If a mobile station finds that this random number is not the same as the one it has sent, it leaves the channel (and possibly starts the procedure of requesting new channels). In the trace, *ms4* receives back the same random number as the one it sent, and so it goes into dedicated mode (line 19). ms4 then sends a location update request to MSC *msc(1)*. Since a Temporary Mobile Subscriber Identity was not included with the request, the VLR ask ms4 to provide its identifier. ms4 replies with its International Mobile Subscriber Identifier (IMSI) *imsi4* which, in our specification, was derived from the directory number *ms4*.

Trace 2: Mobile Terminate Call - Late Channel Assignment

The scenario represented by the trace in Figure 46 is described in Section 2.4.4. A time sequence diagram corresponding to the trace is shown in Figure 47.

Note that the parameter `noDChan` in the trace does not mean that no channels have been granted, rather it means that no *voice* channel have been assigned but *signalling* channels have been granted.

Trace 3: Location Update with IMSI, Authentication, Set Cipher Mode

The trace in Figure 48 shows the following behavior:

1. The MS requesting location update (lines 3-5),
2. The MSC asking its VLR to update its database (line 6),
3. The VLR informs the HLR to update its database (lines 7-11)
4. The VLR authenticate the MS (lines 15-27)
5. The VLR sets the encryption mode with the MS (lines 28-35).

Note that the VLR did not update in step 2 above. In this scenario, the VLR will update its database after the Set Cipher Mode scenario has ended which, unfortunately, also happens to be the end of our trace.

```
1  GSM19[eni,ext,mmi,oami]()
2  | oami ?dn = ms4 [NotIn(dn, { }) and dn <> noMSISDN];
3  | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !BCCH !Broadcast_info;
4  | | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !RACH ?rand,rnd = rand2 !ChanReq !LOCUPD, [rnd <> noRand];
5  | | | | hidden abisi !RSM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !rand2 !ChanReqd !LOCUPD;
6  | | | | | hidden abisi !RSM !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !rand2 !ChanActivation ?dchan,dchan = dchanH2, [dchan <> noDChan];
7  | | | | | | hidden abisi !RSM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !rand2 !ChanActivationAck;
8  | | | | | | | hidden abisi !RIL3RR !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !rand2 !ImmdAssignment;
9  | | | | | | | | internal exit(dchanH2)
10 | | | | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !PAGCH !rand2 !dchanH2;
11 | | | | | | | | | | internal exit(dchanH2, bsc(msc(1),2))
12 | | | | | | | | | | | hidden ri !RIL3MM !noTMSI !bts(bsc(msc(1),2)1) !dchanH2 !LOCUPD ?rnd,rnd = rand2 !MS_INFO;
13 | | | | | | | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !noTMSI !dchanH2 !LOCUPD !rand2;
14 | | | | | | | | | | | | | internal exit(LOCUPD, dchanH2, bts(bsc(msc(1),2)1), noBts)
15 | | | | | | | | | | | | | | internal exit(LOCUPD, dchanH2, bts(bsc(msc(1),2)1))
16 | | | | | | | | | | | | | | | hidden abisi !RSM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !noTMSI !dchanH2 !LOCUPD !EstablishInd;
17 | | | | | | | | | | | | | | | | internal exit
18 | | | | | | | | | | | | | | | | | hidden ai !BSSMAP !bsc(msc(1),2) !msc(1) !bts(bsc(msc(1),2)1) !noTMSI !dchanH2 !LOCUPD !CompleteLayer3Info;
19 | | | | | | | | | | | | | | | | | | MS_SAP1 !dedicated:MSState !noTMSI !LOCUPD !dchanH2 !bts(bsc(msc(1),2)1) !noCksn:CKSN [LOCUPD == LOCUPD or LOCUPD == IMSIDe-
tach];
20 | | | | | | | | | | | | | | | | | | | hidden ri !RIL3MM !dchanH2 !bts(bsc(msc(1),2)1) !LUReq(noTMSI, noCksn) [noTMSI == noTMSI];
21 | | | | | | | | | | | | | | | | | | | | hidden abisi !RIL3MM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !dchanH2 !LUReq(noTMSI, noCksn);
22 | | | | | | | | | | | | | | | | | | | | | hidden ai !RIL3MM !bsc(msc(1),2) !msc(1) !bts(bsc(msc(1),2)1) !dchanH2 !LUReq(noTMSI, noCksn) [not(IsPrimitive(AuthRespx, LUReq(noT-
MSI, noCksn)))];
23 | | | | | | | | | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH2 !ULArea !noTMSI !noCksn:CKSN;
24 | | | | | | | | | | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH2 !ProvIMSI;
25 | | | | | | | | | | | | | | | | | | | | | | | | hidden ai !RIL3MM !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH2 !IdReq;
26 | | | | | | | | | | | | | | | | | | | | | | | | | hidden abisi !RIL3MM !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH2 !IdReq;
27 | | | | | | | | | | | | | | | | | | | | | | | | | | hidden ri !RIL3MM !bts(bsc(msc(1),2)1) !dchanH2 !IdReq [not(IsPrimitive(AuthReqx, IdReq))];
28 | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden ri !RIL3MM !dchanH2 !bts(bsc(msc(1),2)1) !IdResp(imsi4);
29 | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden abisi !RIL3MM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !dchanH2 !IdResp(imsi4);
```

Figure 44 Trace corresponding to the concatenation o the Network Access and Location Update-TMSI Unknown scenarios.

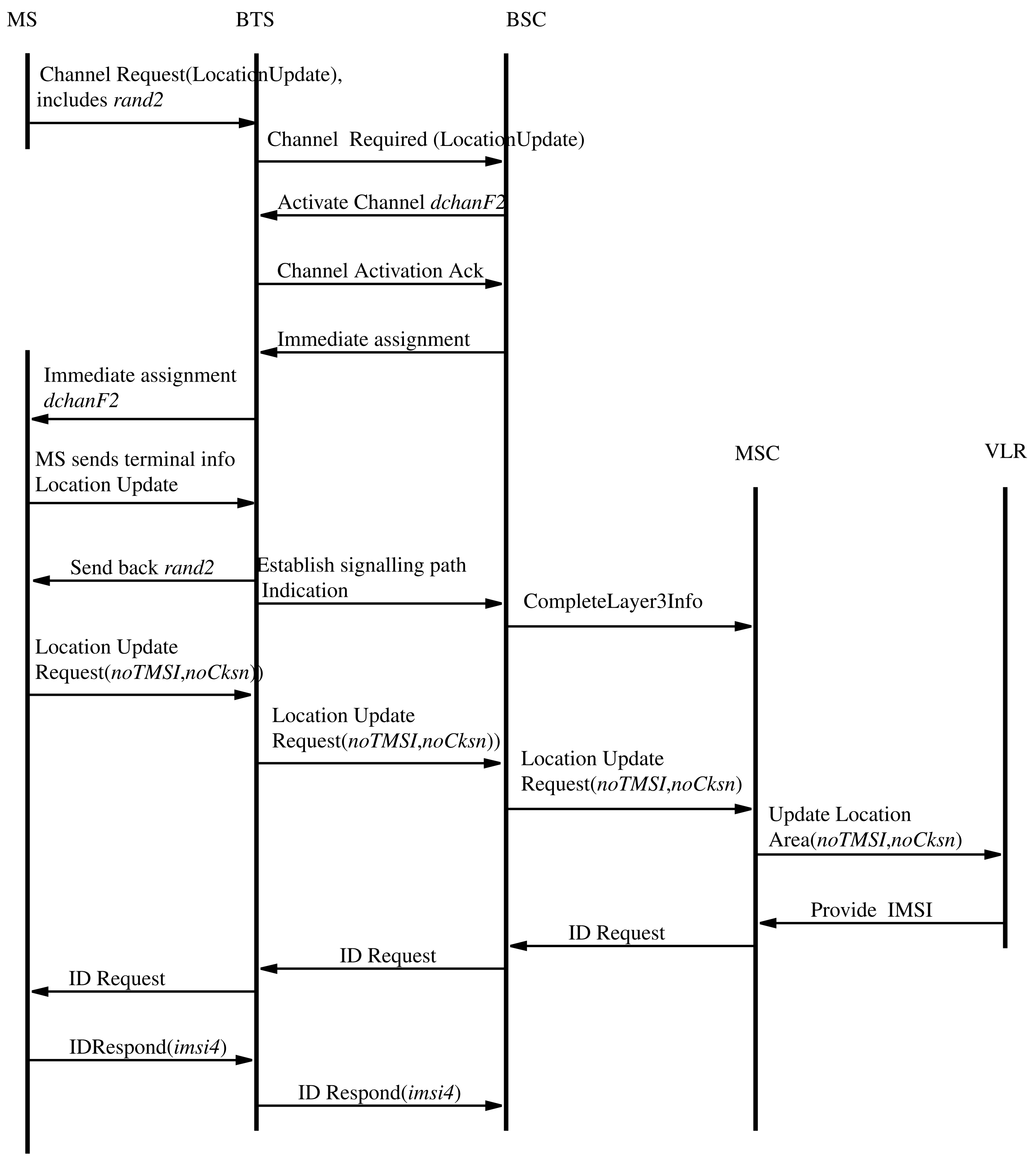


Figure 45 Location Update - TMSI Unknown

```
GSM19[eni,ext,mmi,oami]()
| ext !ISUP !IAM:PSTNAction ?dn = ms4;
| | hidden ci !MAPC !HLRof(ms4):HLRId !SendRoutingInfo !ms4;
| | | hidden ci !MAPC !HLRof(ms4):HLRId !SendRoutingInfoRes !msc(1);
| | | | hidden zi !msc(1) !IAM:PSTNAction !ms4;
| | | | | hidden bi !msc(1) !SendInfoForOutgoingCall !imsi4;
| | | | | | hidden intraVLR !VLR_QUERY !imsi4;
| | | | | | | hidden intraVLR !VLR_RESPONSE !tmsi121;
| | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !PageMS !tmsi121;
| | | | | | | | | hidden ai !RIL3CC !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !PageCmd !ms4;
| | | | | | | | | | hidden abisi !RIL3CC !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !PageCmd !ms4;
| | | | | | | | | | | hidden ri !RIL3CC !bts(bsc(msc(1),2)1) !ms4 !PAGCH:Channel !PageCmd;
| | | | | | | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !RACH:Channel ?rand,rnd = rand2 !ChanReq !PAGING_RESP, [rnd <> noRand];
| | | | | | | | | | | | | hidden abisi !RSM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !rand2 !ChanReqd !PAGING_RESP;
| | | | | | | | | | | | | | hidden abisi !RSM !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !rand2 !ChanActivation !noDChan;
| | | | | | | | | | | | | | | hidden abisi !RSM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !rand2 !ChanActivationAck;
| | | | | | | | | | | | | | | | hidden abisi !RIL3RR !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !rand2 !LateAssignment;
| | | | | | | | | | | | | | | | | internal exit(noDChan)
| | | | | | | | | | | | | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !PAGCH:Channel !rand2 !noDChan;
| | | | | | | | | | | | | | | | | | | internal exit(noDChan, bsc(msc(1),2))
| | | | | | | | | | | | | | | | | | | | hidden ri !RIL3RR !tmsi121 !bts(bsc(msc(1),2)1) !noDChan !PAGING_RESP ?rnd,rnd = rand2 !MS_INFO;
| | | | | | | | | | | | | | | | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !tmsi121 !noDChan !PAGING_RESP !rand2:;
| | | | | | | | | | | | | | | | | | | | | | hidden abisi !RSM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !tmsi121 !noDChan !PAGING_RESP !EstablishInd;
| | | | | | | | | | | | | | | | | | | | | | | internal exit
| | | | | | | | | | | | | | | | | | | | | | | | internal exit(PAGING_RESP, noDChan, bts(bsc(msc(1),2)1), noBts)
| | | | | | | | | | | | | | | | | | | | | | | | | internal exit(PAGING_RESP, noDChan, bts(bsc(msc(1),2)1))
| | | | | | | | | | | | | | | | | | | | | | | | | | hidden MS_SAP1 !dedicated:MSState !tmsi121 !PAGING_RESP !noDChan !bts(bsc(msc(1),2)1) !noCksn:CKSN;
| | | | | | | | | | | | | | | | | | | | | | | | | | | hidden MS_SAP2 !dedicated:MSState !tmsi121 !PAGING_RESP !noDChan !bts(bsc(msc(1),2)1) !noCksn:CKSN;
| | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden ri !RIL3CC !tmsi121 !bts(bsc(msc(1),2)1) !noDChan !Setup;
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden abisi !RIL3CC !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !tmsi121 !noDChan !Setup;
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden ai !BSSMAP !bsc(msc(1),2) !msc(1) !bts(bsc(msc(1),2)1) !tmsi121 !noDChan !PAGING_RESP !CompleteLayer3Info;
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !tmsi121 !ProcessAccessReq;
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | choice kcIsKnown:Bool;
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | internal exit at line(s) 424
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !tmsi121 !AccessReqAccepted;
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !tmsi121 !CompleteCall;
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden ai !RIL3CC !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !tmsi121 !AssignmentReq;
```

Figure 46 Trace Corresponding to the Mobile-Terminated Call-Late Channel Assignment scenario.

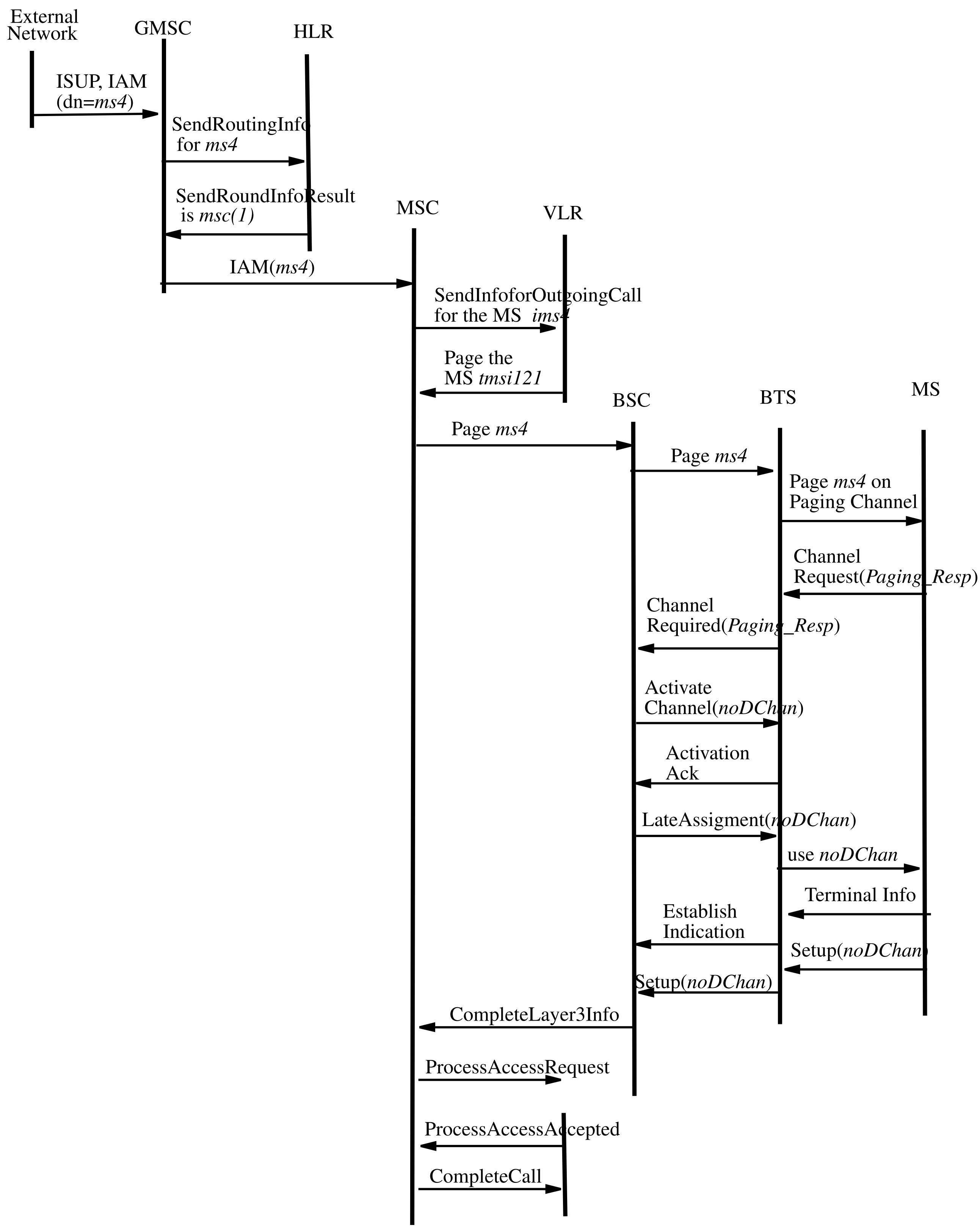


Figure 47 Mobile-Terminated Call- Late Channel Assignment

```
1  GSM19[eni,ext,mmi,oami]()
2  | hidden MS_SAP1 !dedicated !noTMSI !LOCUPD !dchanH1 !bts(bsc(msc(1),2)1) !noCksn [LOCUPD == LOCUPD or LOCUPD == IMSIDetach];
3  | | hidden ri !RIL3MM !dchanH1 !bts(bsc(msc(1),2)1) !LUReq(imsi4) [noTMSI == noTMSI];
4  | | | hidden abisi !RIL3MM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !dchanH1 !LUReq(imsi4);
5  | | | | hidden ai !RIL3MM !bsc(msc(1),2) !msc(1) !bts(bsc(msc(1),2)1) !dchanH1 !LUReq(imsi4) [not(IsPrimitive(AuthRespx, LUReq(imsi4)))];
6  | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !ULArea !imsi4 !noCksn;
7  | | | | | | hidden di !MAPD !msc(1) !hlr1 !imsi4 !UpdLoc;
8  | | | | | | | i (specified explicitly)
9  | | | | | | | | hidden di !MAPD !hlr1 !msc(1) !UpdLocRes;
10 | | | | | | | | | hidden di !MAPD !hlr1 !msc(1) !imsi4 !InsSubscData;
11 | | | | | | | | | | hidden di !MAPD !msc(1) !HLRof(imsi4) !InsSubscDataRes;
12 | | | | | | | | | | | internal exit
13 | | | | | | | | | | | | choice ?r = rand1;
14 | | | | | | | | | | | | | choice ?c = cksn2;
15 | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !Auth !rand1 !cksn2;
16 | | | | | | | | | | | | | | | hidden ai !RIL3MM !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !AuthReq(rand1, cksn2);
17 | | | | | | | | | | | | | | | | hidden abisi !RIL3MM !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !AuthReq(rand1, cksn2);
18 | | | | | | | | | | | | | | | | | hidden ri !RIL3MM !bts(bsc(msc(1),2)1) !dchanH1 !AuthReq(rand1, cksn2) [IsPrimitive(AuthReqx, AuthReq(rand1, cksn2))];
19 | | | | | | | | | | | | | | | | | | hidden sim !A8:SecMan !A3:SecMan !Of(AuthReq(rand1, cksn2));
20 | | | | | | | | | | | | | | | | | | | choice ?MSanswer = triplet1:AuthTriplet;
21 | | | | | | | | | | | | | | | | | | | | hidden sim !triplet1:AuthTriplet;
22 | | | | | | | | | | | | | | | | | | | | | hidden ri !RIL3MM !dchanH1 !bts(bsc(msc(1),2)1) !AuthResp(sres1);
23 | | | | | | | | | | | | | | | | | | | | | | hidden abisi !RIL3MM !bts(bsc(msc(1),2)1) !bsc(msc(1),2) !dchanH1 !AuthResp(sres1);
24 | | | | | | | | | | | | | | | | | | | | | | | internal exit(CKSNof(AuthReq(rand1, cksn2)), KCof(triplet1):KC)
25 | | | | | | | | | | | | | | | | | | | | | | | | hidden ai !RIL3MM !bsc(msc(1),2) !msc(1) !bts(bsc(msc(1),2)1) !dchanH1 !AuthResp(sres1) [IsPrimitive(AuthRespx, AuthResp(sres1))];
26 | | | | | | | | | | | | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !AuthResp !sres1:SRES;
27 | | | | | | | | | | | | | | | | | | | | | | | | | | internal exit(KCof(AuthTripletOf(rand1)):KC)
28 | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !SetCipherMode !kc1:KC;
29 | | | | | | | | | | | | | | | | | | | | | | | | | | | | internal exit
30 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden bi !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !FwdTMSI ?newtm,tm = tmsi112, [MSCof(newtm) == msc(1)];
31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden ai !BSSMAP !msc(1) !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !CipherModeCmd(kc1) [IsPrimitive(CipherModeCmdx,
CipherModeCmd(kc1))];
32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden abisi !RSM !bsc(msc(1),2) !bts(bsc(msc(1),2)1) !dchanH1 !EncryptCmd(CipheringMCmd, kc1);
33 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !dchanH1 !CipheringMCmd [IsPrimitive(CipheringMCmdx, CipheringMCmd)];
34 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | i (specified explicitly)
35 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | hidden ri !RIL3RR !bts(bsc(msc(1),2)1) !dchanH1 !CipheringMComp; line(s)
```

Figure 48 Trace corresponding to the concatenation of the Location Update with IMSI, Authentication and Set Cipher Mode scenarios.