Ottawa University

Computer Science departement

DSM specification in LOTOS and Test Cases Generation

Nancy CARRERE

IG 3

INT

(French Telecommucication National Institute)

TABLE OF CONTENTS

INTRODUCTION

The purpose of this work is to write a LOTOS specification of the Mobile Application Part, which is made of a group of protocol used between GSM (Global System for Mobile Communication, described in Part 1) components.

The second step is to generate Tests Cases from this specification; that is to define sequences of events and the corresponding behaviour of the MAP, in order to be able to do some testing on a real implementation of the system.

The first 2 parts of this report are short presentations of GSM and of the Lotos specification language; then we explain in Part 3 what the MAP is and the different protocols it is made of. In part four, we will focus on the Dialogue State Machine which is the part of the MAP we chose to specify; and the last part will show and explain the building of the specification, and finally the result we obtained concerning Test Cases Generation.

ANNEX No 1 :

global LOTOS specification of the Dialogue State Machine

PART 1 : an introduction to GSM

1 Introduction

Throughout the last few years, formal description techniques (FDTs) have been recognized as useful tools to support the development of complex systems, such as telecommunications systems. FDTs aim at the production of unambiguous, precise, and complete (standard) specifications. Furthermore, they provide a formally well-defined basis for the verification of specifications prior to implementation and conformance testing after implementation. These means of system validation are indeed of high importance already at the earlier system development stages for managing the increasing complexity of each consecutive design phase, and for gaining stepwise confidence in the final product.

In this paper we focus on the use of the FDT LOTOS for specifying and validating GSM, a digital cellular communications system proposed by the European Telecommunication Standards Institute (ETSI) [REF]. LOTOS is an algebraic specification language standardized within ISO [ISO88]. Its applicability to the design and validation of telephone systems has already been shown quite extensively. In [FLS90, FLS91, SL93] LOTOS specifications of a POTS (Plain Old Telephone System) are given based on various specification styles. In [Bou91, BL93] these specifications are modified and enhanced with supplementary features such as call forwarding and conference calling. Also, in [EHM92] an experiment has been carried out towards the specification of an ISDN telephone system in LOTOS. The Global System for Mobile communications (GSM) offers the possibility for yet another application. It is a challenging one, still, particularly due to the complexity introduced by the mobility aspects of communication.

The area of mobile communications is not completely new to the FDT community. Work has been published on the specification of mobile networks [CV89, EFJ91, FO92, OP90]. Some of these papers involve GSM. For instance, Orava et al. describe a part of GSM, namely the handover procedure, in both LOTOS [FO92] and the π-calculus [OP90]. The complete mobile application part of GSM is specified in SDL within ETSI [REF].

2 GSM : a quick survey

In 1982, CEPT (Conférence Européenne des Administrations des Postes et des Télécommunications) formed the ‘Group Spécial Mobile’ (GSM!) with the task to make a recommendation for a Public Land Mobile Network (PLMN). The major characteristic of such a network is that some nodes are *mobile* and may be connected to different nodes at different points in time. The requirements on the PLMN were that it should be usable in all European countries, it should provide telephone services as well as supplementary services according to ISDN, it should be able to coexist - in the same radio band - with other mobile systems, and it should not require any major modifications to existing public networks. It was soon recognized that the PLMN would need a new set of communication protocols to manage routing to mobile nodes. The development of these protocols has now been finalized by ETSI.

In the following, the GSM PLMN system architecture is described from both an equipment viewpoint and a functional viewpoint. Furthermore, the GSM protocols demarcated within these two views are identified. This discussion is based on [MP92].

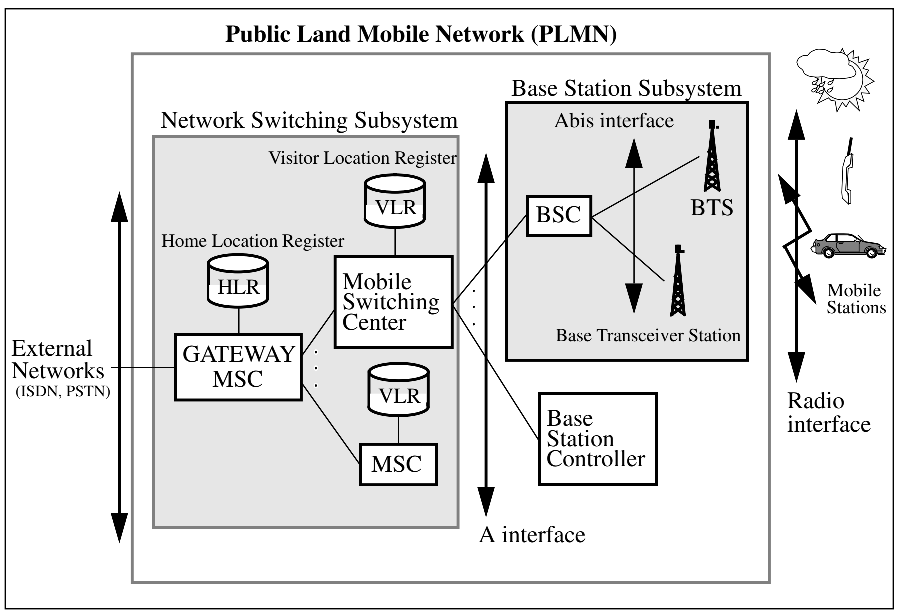
2.1 Physical machines

From a physical perspective the internal GSM domain is composed of 4 sub-systems:

1. the *mobile station* (MS);
- the *base station sub-system* (BSS);
- the *network and switching sub-system* (NSS);
- the *operation and maintenance sub-system* (OSS).

The MS, the BSS and the NSS form the operational part of GSM, handling the telecommunications traffic. The OSS constitutes (together with an operator) the control part which interacts with the traffic handling part by observing and modifying it so as to maintain or improve its functioning.

FIGURE 0. STRUCTURE OF A PLMN.



The operational structure of a GSM PLMN is depicted in Figure 1. A mobile station is a terminal equipment providing service to an end user. It can range from vehicle-mounted and portable equipment to normal hand-held stations. The BSS and the NSS are on the stationary side of the network and comprise the canonical GSM architecture. Basically, the BSS groups the infrastructure machines which are specific to the radio cellular aspects of GSM. These include the *base transceiver stations* (BTS) and the *base station controllers* (BSC). A BTS comprises devices for radio transmission and reception. Each BTS provides radio coverage within a designated geographical area, called a *cell*. As such, the BSS is in direct contact with mobile stations through the radio interface. A BSC is in charge of the management of the radio interface through the remote control of a set of BTSs via the so-called *Abis*-interface. This involves mainly the allocation and release of radio channels, and the handover management. The role of the BSS can be summarized as to connect the MS and the NSS, and hence the mobile station's end users with other telecommunication users. Since the BSS has to be controlled it is also in contact with the OSS.

The NSS includes the main switching functions of GSM, as well as the databases needed for subscriber data and mobility management. Its main role is to manage communications and connect mobile stations to the relevant external networks or other MSs. Similar as the BSS, the NSS is controlled by the OSS for the purpose of network operation and equipment maintenance. The main switching function in the NSS is performed by the *mobile switching centers* (MSCs), which coordinate the setting-up of calls to and from GSM users. Each MSC manages a set of BSCs through the so-called *A*-interface. As such, the NSS connects to the BSS, and hence to the GSM users. The databases within the NSS are called location registers and record the status of the PLMN. Each subscriber is associated with a *home location register* (HLR) which contains the information related to the subscriber's current location. This enables incoming calls towards mobile subscribers to be routed and established. The other location register is the *visitor location register* (VLR). A VLR is commonly integrated with each MSC, although it may be shared among several MSCs. The VLR is in charge of temporarily storing subscription data for those subscribers currently *roaming* in the service area of the corresponding MSC; this data is copied from the subscriber's HLR upon initial entry in the service area. Also, the VLR holds information on the subscribers' locations at a more precise level than the HLR (which only refers to MSCs). At last, the NSS contains the *gateway MSC* (GMSC). Upon an incoming call via an external network the GMSC is in charge of fetching the location information of the called GSM subscriber by interrogating the appropriate HLR, and subsequently routing the call towards the MSC currently serving the subscriber (the visited MSC).

2.2 Functional layers

The GSM structure also allows a decomposition from a strictly functional perspective. Analogous to the OSI reference model, functions are grouped in horizontal layers. Each layer offers a specific set of services to the next layer up by enhancing the services offered by the next layer below.

Five functional layers are distinguished within the GSM architecture, as shown in Figure 2:

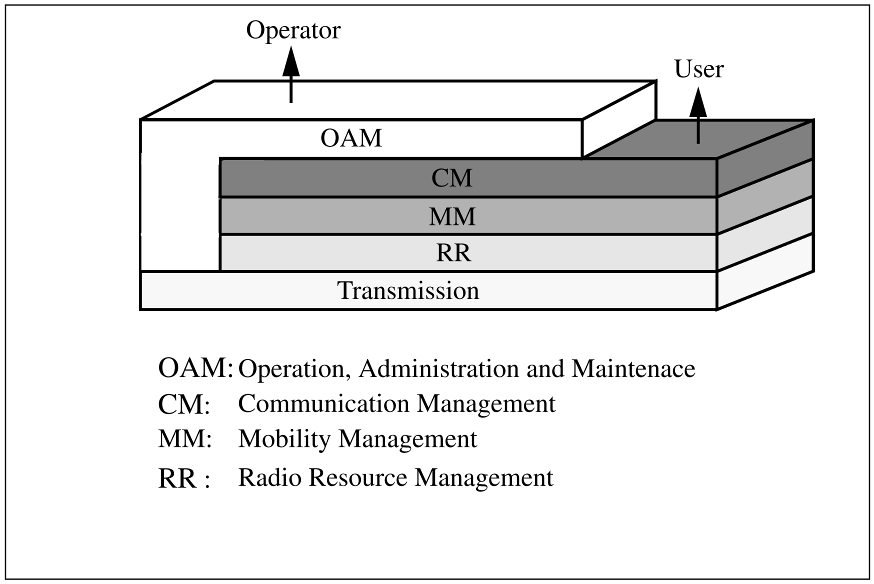
1. the *transmission* layer;
- the *radio resource management* layer (RR layer);
- the *mobility management* layer (MM layer);
- the *communication management* layer (CM layer);
- the *operation, administration and maintenance* layer (OAM layer).

The transmission layer lies at the bottom and provides transmission means for user information (speech or data), as well as for signalling messages between the cooperating machines. Being the basis of any telecommunications system, this layer is not GSM specific. Nevertheless, it involves recent developments in the telecommunications field, most notably the CCITT Signalling System No. 7 (SS7).

The next layer up is the radio resource management layer. The management of radio transmission resources is a complex issue in a cellular system such as GSM and warrants its own functional layer. The functions involved are the allocation and release of stable links between the mobile stations and the MSCs, and their maintenance despite user movements during calls (handovers). These functions are mainly implemented by the MS and the BSC, although the MSC implements part of the handover process.

The mobility management layer has also strong GSM specificity. It includes aspects, not specific to the radio interface, but rather peculiar to cellular networks: mobility and security management. The management of mobility has rendered necessary by the movements of subscribers. The functions involved correspond to the network access procedure which determines how the mobile station chooses among cells, and the location update process by which the databases are updated when the MS moves across certain cell boundaries. The security management functions provide means to authenticate subscribers and to cipher transmission paths. The implementation of both the mobility and security management functions share the same machines (MS, MSC, VLR and HLR).

FIGURE 0. FUNCTIONAL LAYERS.



The next layer, the communication management layer, is again less specific to GSM. The functions consist in conventional call control procedures: setting up calls between users, as well as maintaining and releasing them. An important aspect herein is the routing function. The CM layer also includes the means for users to have some control over the management of calls through supplementary services such as call forwarding.

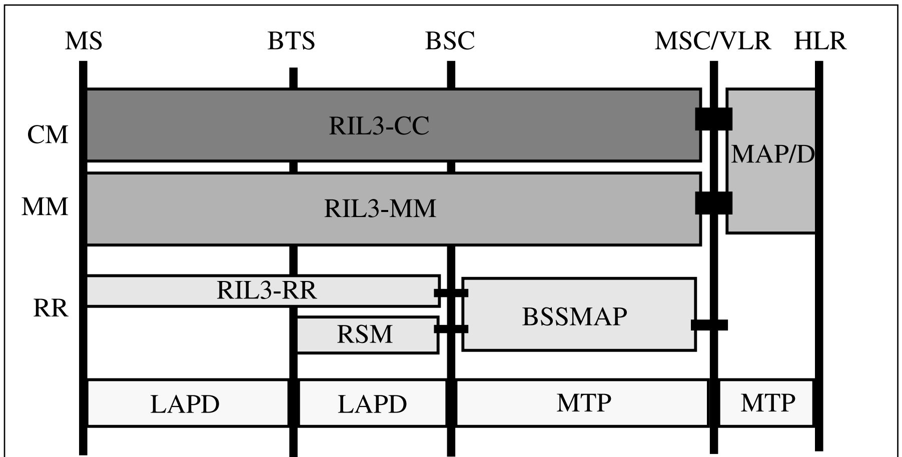
Finally, the operation, administration and maintenance layer is added to make the picture complete. It provides the means for operator actions, i.e. all functions which enable an operator to control and configure the PLMN. The OAM layer is not strictly above the other layers in terms of services, since it does not enhance directly the services offered by these layers. Its kinship with the operation and maintenance sub-system is obvious.

2.3 Protocols and interfaces

As part of the quick survey of GSM, this section introduces the main GSM protocols with a brief explanation about their respective functions. Figure 3 shows an overview of the signalling architecture in the machines of a GSM transmission chain, as far as the telecommunications functions are concerned. The horizontal axis corresponds to the distribution of the various physical machines and the vertical axis corresponds to the functional layers, as discussed in the previous subsections.

Due to its limited GSM specificity, the main transmission protocols included in the transmission layer are the customary LAPD and MTP (Message Transfer Part) protocols. The LAPD protocol operates on the radio interface as well as the Abis-interface. MTP refers in fact to a set of protocols used for signalling on an SS7 network. They appear at the A-interface and among the NSS entities.

FIGURE 0. GSM PROTOCOLS.



BSSMAP: BSS Management Application Part
RIL: Radio Interface Layer
RSM: Radio Subsystem Management
MAP: Mobile Application Part
LAP: Link Access Procedure
MTP: Message Transfer Part

The RIL3-RR protocol embodies the RR functional layer on the radio interface (RIL3 stands for Radio Interface Layer 3). It enables a mobile station and a BSC to cooperate for the management of radio resources. The protocol also appears on the Abis-interface where it works closely together with the RSM (Radio Subsystem Management) protocol. The latter specifically 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) protocol, which allows an MSC to manage several BSCs. The BSSMAP is also partially responsible for handover management (in collaboration with the MAP/E protocol, see below).

The upper layer protocols RIL3-MM and RIL3-CC define rules for signalling exchanges between the mobile station and NSS entities. Although these protocols also appear when looking at the Abis-interface and A-interface, both the BTS and BSC are “transparent” to these signalling exchanges, i.e. they just act as information carriers without being aware of the semantics of this information. Finally, non call-related signalling specific to GSM corresponds to many protocols which are grouped together as MAP/*x* (Mobile Application Part) protocols. In particular, MAP/D refers to the protocol between the MSC/VLR and the HLR.

PART 2 : An Overview of LOTOS : Language of Temporal Ordering Specification

3 LOTOS : an FDT from ISO

Formal Description Techniques are vital tools for communication system designers: they force the designer to communicate his design in precise and unambiguous terms, and they facilitate the sharing of design ideas between users, designers and managers.
LOTOS (Language Of Temporal Ordering Specification) is one of the two Formal Description Techniques developed within ISO (International Organization for Standardization) for the for specification of open distributed systems and in particular for those related to the Open System Interconnection (OSI) computer network Architecture. It was developed by eighties from the basic idea assuming that systems can be specified by defining the tem po ral relation among the interactions that constitute the externally observable behavior.

4 LOTOS basic elements : Process, Event, Synchronization

In LOTOS, a distributed, concurrent system is seen as a *process*, possibly consisting of several sub-processes. A sub-process is a process itself, so that generally a LOTOS specification describes a system via a hierarchy

of process definitions. LOTOS process definitions are named *Behavior Expressions*, similar to procedures in programming languages, whereas *Process Instantiations* are similar to procedure calls.
A *process* is an entity able to perform internal, unobservable actions, and to interact with other processes which form its environment. Complex interactions between processes are built up out of elementary units of synchronization called *events* or *atomic interactions* or simply *actions*.
Events imply *process synchronization* because the processes that interact in an event (they may be two or more) participate in its execution at the same moment in time and at the same interaction point called *gate*. *Events* are atomic in the sense that they occur instantaneously, without consuming time, and in case of synchronization without data exchange, the *event* name and the *gate* name coincide.
The semantics of process synchronization are quite different in LOTOS from those found in common programming languages: it is

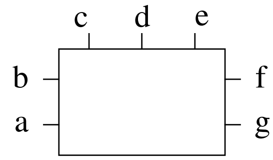
- *multi-way synchronization*: more than two processes can participate.

- *symmetric synchronization*: all processes that participate in a synchronization cooperate in it equally, in particular, there is no concept of a process initiating the synchronization and others responding. For instance, while it is often thought that in an output operation, a producer process transfers information to a passive external environment, in LOTOS, one says that both the producer and the environment participate in establishing the value being output. There is no directionality in this concept, although some processes may have more information on the values to be transferred than others.

- *anonymous synchronization*: a process that is ready for a synchronization proposes the synchronization to its environment without being able to direct its proposal to a specific process. It is the structure of the system where the process is embedded that decides which processes will have to participate for the synchronization to occur. Process identification can be specified as an exchange of appropriate values.

  - *nondeterministic synchronization*: often more than one synchronization is possible, but only one will be executed according to a nondeterministic choice.

The most abstract representation of a process P able to interact with its environment via gates *a* through *g* is a black box:



Process P

The process definition of P specifies its behavior by defining the sequences of observable actions that may occur at the seven gates of the process. Such a behavior is usually represented as a tree of actions where the root is the initial state of the system from which the behavior may execute one of the possible actions, in cooperation with its environment, and so move to another state. When an action is executed, the behavior expression of the process is transformed into another behavior expression. The inference rules of the dynamic semantics determine which actions can be offered and executed by a process and how behavior expressions are transformed by effect of the execution of actions. For instance, we shall see that the behavior expression a;B evolves into the behavior expression B after executing action a.

5 Basic LOTOS and full LOTOS

Basic LOTOS is the subset of LOTOS where processes interact with each other by pure synchronization, without exchanging values.
Full LOTOS allows processes to exchange values and to be parametrized by them.
A set of operators is used to combine actions and behavior expressions to form other behavior expressions. There are two predefined behavior expressions:

- the inaction behavior **stop**: it cannot offer anything to its environment, nor it can perform internal actions.

- the successful termination **exit**.

Other LOTOS operators are explained next.

6 LOTOS operators used in this work

6.1 Action Prefix Operator ‘;’

The *action prefix* operator written as a semi-colon ‘;’ expresses sequential composition of actions. This operator is used to sequentially order actions. For example, a;B denotes a behavior where action a must be executed before the behavior expression B.

6.2 Choice operator ‘[]’

The *choice* operator written ‘[]’ is used to express a choice between two alternatives. A selection between both alternatives is said to be nondeterministic if the environment cannot distinguish between them. Otherwise, the choice is said to be deterministic. For example, compare the behaviors of three vending machines:

(insert_quarter; get_coffee; **stop**) [] (insert_dime; get_milk; **stop**) (*M1*)
(insert_quarter; get_coffee; **stop**) [] (insert_quarter; get_milk; **stop**)(*M2*)
insert_quarter; ((get_coffee; **stop**) [] (get_milk; **stop**))(*M3*)

Machine M1 offers the client (the environment) a choice between inserting a quarter and getting a coffee or inserting a dime and getting a milk; the choice is deterministic.
Machine M2 accepts quarters only and after synchronization with the environment (i.e. once the client has inserted a quarter), the behavior of the machine can evolve to either get_coffee; **stop** or get_milk; **stop** depending on a nondeterministic choice (in other words, once the client has inserted a quarter, the choice to synchronize on either get_coffee or get_milk would no longer exist).
Machine M3 accepts quarters only like M2 but it is more democratic: once the client has inserted a quarter, the behavior of the machine evolves into (get_coffee; **stop**) [] (get_milk; **stop**), meaning that the client can still choose between coffee and milk.

6.3 enable operator ‘>>’

The *enable* operator ‘>>’ has a similar function as the *action prefix* operator, which expresses the sequential composition of an action with a behavior expression; the ‘>>’ is used to express the sequential composition of two behavior expressions. For example, if P1 and P2 are two processes and P1>>P2, P1 must terminate successfully in order for P2 to be enabled. This is the only condition under which process P2 is enabled. Execution of an **exit** in P1 results in an action on a special gate δ. The enable causes δ to become an **i**, and execution to continue with P2. For example,

((a; b; **exit**) >> (c; **stop**)) is equivalent to (a; b; **i**; c; **stop**) i.e. (a; b; c; **stop**)
((a; b; **stop**) >> (c; **stop**)) is equivalent to (a; b; **stop**) because the expression on the right hand side of the enable operator cannot be executed.

6.4 Parallel Composition

There are three parallel composition operators in LOTOS: a basic one, the *selective composition* operator, and two derived ones, the *interleaving* operator and the *full synchronization* operator.
The *interleaving* operator denoted by '|||' is used to express the concept of parallelism between behaviors when no synchronization is required:

((a; b; **exit**) ||| (c; d; **exit**)) is equivalent to
(
(a;
(b; c; d; **exit**
[]c; (b; d; **exit** [] d; b; **exit**)
)
)
[](c;
(d; a; b; **exit**
[]a; (b; d; **exit** [] d; b; **exit**)
)
))

When processes must synchronize on common actions, the *selective parallel* operator denoted by '|[L]|' is used, where L is a list of actions on which synchronization must occur. For example,

((a; b; **exit**) |[a]| (d; a; c; **exit**)) is equivalent to
(d; a; (b; **exit** [] c; **exit**)) which is equivalent to
(d;a; (b; c; **exit** [] c; b; **exit**))

This is because both behavior sub-expressions execute independently until one of them reaches a common action, in which case it must wait for the other sub-expression. Once the second sub-expression reaches

the same point, synchronization is possible (depending on the context in which the expression occurs, for example, participation of other processes may be necessary) and both sub-expressions proceed to offer their next actions. In this example, the first action of sub-expression 1 is a. Since action a is common to both sub-expressions, it must wait for sub-expression 2 to reach the action a, before offering action b. On the other hand, action d is not common, so no synchronization is required. After action a, they both continue independently, thus sub-expression 1 can offer action b and sub-expression 2 can offer action c, which can occur in either order.
Note that an action is not treated as common between processes composed in parallel unless it is shown in the list of gates in the parallel composition operator. Clearly, when L is the empty list, the selective parallel operator becomes identical to the interleaved operator.
A special case is provided by the action d which is produced by **exit**s. The action d is always considered as a common gate, for any parallel composition operator. Therefore, all behaviors composed in parallel must synchronize on theirs **exit**s. For example:

((a; **exit**) ||| (b; c; **exit**)) is equivalent to ((a; b; c; **exit**) [] b; (a; c; **exit** [] c; a; **exit**)) and both of these behaviors are ready to synchronize with the environment on any of these following sequences: a b c d, b a c d, b c a d.

The *full synchronization* operator denoted by '||' is used when the processes involved in synchronization must synchronize on every observable action. Clearly, when L is the list of all gates, the *selective composition* operator becomes identical to the *full synchronization* operator. For example:

(a; b; c; **exit**) can only synchronize with the behavior (a; b; c; **exit**).
Therefore, ((a; b; c; **exit**) || (a; b; c; **exit**)) is equivalent to (a; b; c; **exit**).
((a; b; **exit**) || (d; a; c; **exit**)) is equivalent to **stop** (deadlock!) because the left hand side offers action a while the right hand side offers action d.

The parallel composition operators '||' and '|||' are commutative and associative. |[L]| is commutative and may be associative depending on the gates in L

There are two more operators : the disable operator, and the hiding operator we are not going to describe because we won't make use of them.

7 Process Instanciation

An *Instantiation* is an invocation of an already defined process. Actual gates and values are used to instantiate the formal gates and parameters of the process definition. Recursivity can be achieved by having a process instantiate itself. For example:

Let P [fg1, fg2,..., fgn] (p1:s1, p2:s2,...,pm:sm).
If B= P [ag1, ag2,...,agn] (v1, v2,..., vn) then B is a process which behaves like P with the actual gates ag1, ag2,...,agn renaming the formal gates fg1, fg2,..., fgn and the actual values v1, v2,..., vn replacing the formal parameters p1:s1, p2:s2,...,pm:sm

PART 3: the MAP protocols

Among the several protocols that are used in GSM for components to exchange messages, 9 protocols are grouped together in the MAP, e.g the Mobile Application Part.

Each protocol corresponds to a specific interface between two components, at a specific GSM layer. The purpose of this part is to show the informations exchanged at those interfaces, and the means required to transfer the messages; that will lead us to understand the link between the 9 different protocols, and the reason why they can be grouped together in the MAP and specified at once.

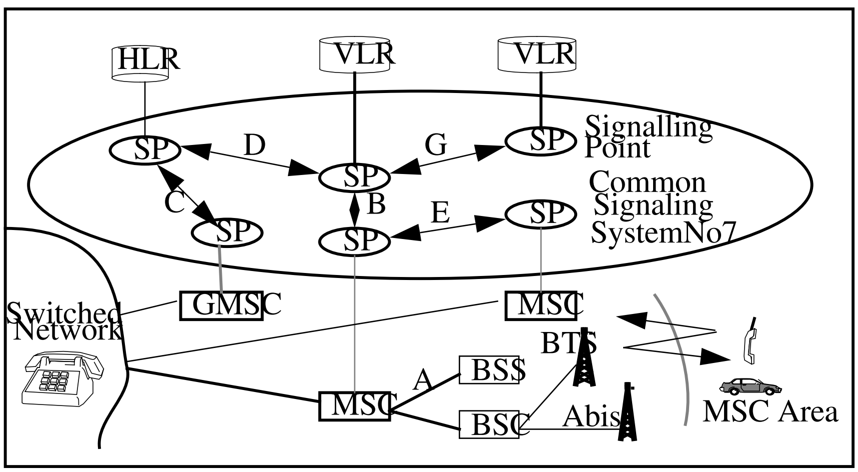
4 The components and GSM layers implied

The most important role of the MAP is to maintain consistency in the several databases that has to be frequently modified because of the mobility of the users, and also when a user decides to add new services, or to suppress some services,e.g. to modify his subscribing options. The MAP makes sure that all data are updated when they need to be; therefore, several components are implied, mostly the Mobile Switching Center (which is highly related to mobility management), the Visitor Location Register (the “local” database), and the Home Location Register (“general” database containing informations about all users roaming in a given PLMN).

Concerning the GSM layers, most of them are implied in so far as they all participate to handle the mobility of the user.

4.1 Components implied

FIGURE 1. Interfaces and control flows between components



The names given to the several interfaces correspond to the MAP protocols. This drawing shows how the information travels (using signalling points) through those interfaces: the signalling points are used when the components implied don’t have any physical connection, e.g when the communication protocol is part of the MAP. We can see that the protocol between a Mobile Switching Center and a Basis Station Controller (A interface) is not part of the MAP.

As we can see on the drawing, the components implied in the MAP/B, MAP/C, MAP/D, MAP/E, and MAP/G are the HLR, the VLRs and the MSCs. The Gateway MSC which is used to route Mobile Terminating calls, can be considered as some kind of MSC.

We notice that some of the MAP protocols are missing here:

- the MAP/F: this protocol is used for the network management, at the Operation and Administration Layer; it implies a database called Equipment Identity Register, that contains informations about the physical communication means (that is, the mobile stations). We won't consider this protocol in our MAP specification, in so far as we decided to focus on the protocols that only implies HLR, VLR or MSC.

- the MAP/H: this protocol is used between an MSC and a SMS_gateway (this component is used when short messages travel in the network). The corresponding layer is the Communication Layer.

- the MAP/I: also belonging to the CM layer, this protocol is used between a Mobile Station and the corresponding HLR.

We will now give a description of the three components (HLR, VLR and MSC) that are implied in the MAP protocols we decided to focus on:

- **HLR** is a database in charge of the management of mobile subscribers. One PLMN may contain one or several HLR; it depends on the number of mobile subscribers, on the capacity of the equipment and on the organization of the network.

All subscription data are stored there. The main information in a HLR concerns the location of each mobile station in order to be able to route calls to the mobile subscribers managed by the HLR.

All management interventions occur on this data base. It is important to underline that a HLR doesn't have any direct control on MSCs.

Concerning the data stored in the HLR

- two numbers attached to each mobile subscription: the IMSI, e.g. the International Mobile Subscriber Identifier, and the MSISDN that corresponds to the phone number one must dial to reach the subscriber.

- other informations, such as the VLR number (location information), the basic telecommunication services subscription information, the service restrictions (e.g. roaming limitation for a given user), and supplementary services chosen by the subscriber.

- **VLR** is in charge of a specific MSC area. An MS roaming in this area is controlled by the corresponding VLR.

When a mobile station appears in a location area, it starts a location updating procedure: the MSC in charge of this area notices this registration, and transfers to the VLR the identity of the location area where the MS is situated. One VLR may be in charge of one or several MSC areas.

The VLR also contains the information needed to handle the calls set up or received by the MSs registered in its database (in some cases, the VLR may have to obtain additional information from the HLR); the followings elements can be found:

- the IMSI of the subscriber

-the MSISDN

- the location area where the mobile station has been registered (this will be used to call the station)

- supplementary services parameters.

- MSC is an exchange which performs all the switching function for mobile stations located in a geographical area designated as the MSC area.

The main difference between an MSC and an exchange in a fixed network is that the MSC has to take into account the impact of the allocation of radio resources and the mobile nature of the subscribers and has to perform, for example, the following procedures:

- procedures required for the location registration

- procedures required for a handover (when a MS changes cell during a call)

The Gateway MSC is a switching center, and as a matter of fact it is exactly like an MSC. In the case of incoming calls to the PLMN, if the fixed network is unable to interrogate directly the right HLR, the call is routed to the GMSC, that is going to interrogate the appropriate HLR and then route the call to the MSC where the mobile station is located.

The choice of which MSCs can act as Gateway MSCs is a network operator matter: it can be decided that all MSCs can be GMSCs, or only some of them...

4.2 layers implied

The MAP deals with the subscriber's mobility: therefore, the GSM layers implied are the Mobility Management Layer, and the Communication Management layer (mobility sometimes needs to be taken into account to handle a communication between two users; for example if there is a location updating because there are two Mobile Switching Center implied in the communication).

The only exception is MAP/F which concerns the Operation and Administration Maintenance layer, and that won't be taken into account in this work.

Among the MAP protocols, the 5 interfaces we decided to focus on only imply MM and CM layer:

- **MAP/B**: between MSC and VLR. MM layer

- **MAP/C**: between HLR and GatewayMSC. CM layer

- **MAP/D**: between VLR and HLR. MM and CM layer

- **MAP/E**: between the relay MSC and the anchor MSC. CM layer

- **MAP/f**: between the old VLR and the VLR currently in charge of an MS. MM layer.

5 The MAP several protocols

5.1 presentation of the 5 interfaces

5.1.1 B interface

This is the interface between the VLR and its associated MSC(s). the MAP/B protocol is used when:

- the MSC needs data related to a given Mobile Station, currently located in its area.

- an MS initiates a location updating procedure: the new MSC informs its VLR which stores the relevant informations in its table.

- a subscriber activates a supplementary service or modifies some data attached to a service: the MSC transfers, via the VLR it is registered in, the request to the HLR which stores these modifications and updates the VLR if necessary.

5.1.2 C interface

this is between the HLR and an MSC, which is a gateway MSC: when a call is coming to a mobile subscriber, if the fixed network is not able to interrogate the HLR directly: in that case, the GMSC is a switching center that indicates which HLR to interrogate (the GMSC holds a table that relates an MSISDN with the corresponding HLR. Once the right HLR is found, the call can be routed thanks to the roaming number of the MS, which indicates its location.

This interface is also used between a HLR and a SMSC (Short Messages Switching Center), when a short message comes to an MS; but we decided to let apart short messages related services.

5.1.3 D interface

Used between the HLR and a VLR, this interface is very important for the Location Updating procedure (when an MS reaches a new MSC area). The following events occur:

- the VLR inform the HLR about the registration of an MS and provides it with the relevant location information.

- the HLR sends to the VLR all the data it needs to support the service to the MS.

- the HLR then calls the previous VLR to inform it that it can cancel the location registration of this mobile station.

An exchange of data between an HLR and a VLR may also occur when an MS requires a particular service, or if the user wants to change some data linked to his subscription.

5.1.4 E interface

This interface is used between two MSCs, the ‘anchor MSC’ and the ‘relay MSC’: when a handover procedure occurs during a call -e.g. when an MS moves from an MSC area to another-, the old MSC and the new one have to exchange data, then they collaborate for the call to continue. The old MSC remains the switching center the call is routed to, this is the reason why it is called an anchor MSC. The new MSC is the relay MSC.

this interface is also used to forward short messages, which we decided not to consider.

5.1.5 G interface

this interface is used when there is a location updating using TMSI, which is an alias for IMSI: when an MS enters an MSC area, it has to give its identity to the network. But this identity musn’t be public: someone else musn’t be able to know where the subscriber is roaming at a given moment.

Therefore, a Temporal Mobile Subscriber Identifier must be used: the network and the MS agreed on it previously, during ciphered exchanges.

What happens is that the new VLR has to get the IMSI (knowing the TMSI only), and the authentication set (used each time the subscriber has to be authenticated) from the old VLR.

5.2 One protocol instead of five: the MAP protocol

In a PLMN, some of the components are physically connected, others are not; the link between them is logical. The GSM entities which are directly linked through a physical interface (contiguous entities) are:

- an MS and its BTS (through the radio channel allocated to the MS)

- a BTS and the corresponding BSC that controls the BTS

- a BSC and the corresponding MSC for cooperation in the area of communications establishment and handover

- an MSC and a point of entry into an external network, for the establishment of communications.

We can notice that the MAP interfaces we described previously take place between non physically linked components: this is the first common feature between MAP protocols; they all make use of the connectionless classes of the Signalling Connection Control Part (SCCP) of Signalling System 7 (we will introduce SS 7 later in this part).

Furthermore, all these protocols are supported by the same underlying architecture: they are at the top of TCAP (Transaction Capabilities Application Part) of SS7. Therefore, they share the same services provided by this TC layer.

Finally, as we will see later in the next part, they can all be melodized the same way, supporting a 'dialogue' between the two components involved, no matter what these components can be; there are some services provided by the MAP that are called Common services, used in order to handle the dialogue (the opening, continuation, closing...). The messages corresponding to the services have to travel from one component to the peer entity, making use of TC messages. This will be explained with more details; what is important to notice here is that this dialogue can be specified without taking into account the specific components involved in it.

For all these reasons, the 5 interfaces (B, C, D, E, G) can be grouped together in one protocol called the Mobile Application Part, because it deals with the behaviour of roaming mobile stations and the impact of their mobility on other MAP components.

6 the MAP architecture

6.1 A brief presentation of SS 7

SS No 7 is some kind of underlying network that provides the physical means to transport data between GSM components without physical connection between them.

Depending upon national regulation, a GSM operator may or may not be allowed to operate the full SS7 network between NSS (Network and Switching subsystem) machines. If the GSM operator has the full control of this signalling network, then Signalling Transfer Points (STPs) will probably be part of the NSS functions (see drawing page 2), and could be implemented either as stand-alone nodes or in the same machine as the MSCs, in order to optimize the cost of the signalling transport between NSS entities (MSCs, VLRs, GMSCs, HLRs).

The MAP makes use of the Signalling Connection Control Part (SCCP) of the SS No 7. It will use the connectionless classes (0 or 1) of the SCCP: within the GSM system thee will be a need to communicate between entities within the same PLMN and in different PLMNs. Using the MAP for this function implies the use of Transaction Capabilities (TC) and the Signalling Connection Control Part of Signalling System No 7.

To address entities (called Application Entities), SCCP makes use of a number called sub-system number (SSN) that refers to the kind of the component implied:

-0000 0110 for an HLR

- 0000 0111 for a VLR

- 0000 1000 for an MSC.

This number is the type of the SCCP address. Concerning the real and complete addressing, it becomes far more complicated, in so far as the address of a given components depends on what is known about this components: for example, when a mobile station registers for the first time in a VLR, location updating is initiated and the VLR has to communicate with the MS's HLR. In as much as the only data for addressing purposes that the VLR has is contained in the IMSI (International Mobile Subscriber Identifier), addressing information for SCCP must be derived from it. The result is an SPC (Signalling Point Code).

6.2 Use of Transaction Capabilities (TC)

The MAP makes use of the services offered by the Transaction Capabilities, and it uses all the services provided by TC, except the ones related to unstructured dialogue facility. We consider the MAP here as a single Application Service Element, like a black box providing services to a specific user (e.g. a component) and using TC services; further structuring of it is for further study.

6.2.1 presentation of TCAP

TCAP is structure as a component sub-layer above a transaction sublayer.

Between those two layers, messages are exchanged, Related Protocol Data Units. The Transaction Sub-layer provides an end-to-end connection over which several data units can be exchanged. The Component sub-layer, which is the upper layer, provides control of end-to-end dialogues, and remote operation handling; we are going to explain this more precisely.

<u>TCAP's transaction sub-layer</u> is not really interesting, although it is necessary: it simply provides a connection between the two components implied in a dialogue. The related protocol data units are built by the Component Sub-layer, and travel through the Transaction sub-layer to reach the other end of the connection.

<u>TCAP's component sub-layer</u> is far more useful and deserves to be described with more details:

- it provides the means to distinguish independent message flows because it adds a transaction indicator to each message that enables the destination to relate all the messages of this transition to a single context.

- TCAP component sub-layer links a request to the result of this request. This is very useful: MAP often does not specify a a message name for an answer, this answer will simply be contained in the return-result or returned-errorlinked by TCAP to the request. This service provided by TCAP is called Remote Operation Handling.

- TCAP provides also a grouping/degrouping function: several operations pertaining to the same dialogue can be grouped inside one TCAP message. We will explain that in the next paragraph.

6.2.2 How the MAP makes use of TCAP

- the MAP need means to relate a message to a specific dialogue (there can be several dialogues at the same time, each one interested only by the messages belonging to a single context).

- thanks to the transaction indicator, added to data units, some informations just have to be sent once, at the beginning of the dialogue: they remain true for all the dialogue. It is then important to delimitate a dialogue, its beginning corresponding to a message sent with TCAP primitive TC-Begin, and its end to a primitive TC-End.

- the MAP also makes use of the grouping/degrouping function: several MAP primitives are sent in the same TC message. We will give an overview of these messages in the next paragraph.

6.2.3 Mapping the MAP primitives onto TC messages

During a dialogue between two MAP users (let's call them User-1 and User-2), the MAP receives requests from the users, that it has to map onto a TC message so that the requests can reach the other user. We can not explain this mapping before learning what the different MAP services are; but let's give an example:

- if User-1 wants to open a dialogue with User-2, it will send to the MAP a 'MAP_OPEN request'. Then, it will send to the MAP the list of services that has to be performed by the MAP, which has to interrogate User-2. The most obvious idea would be to send TC requests for each one of these User-1 requests; but this is not the way it works.

There is no TC message sent before User-1 sends a MAP_DELIMITER request (indicating to the MAP that the list of services to be performed is over). Then, a TC_BEGIN request message is built, containing the whole list of services that User-1 requested, and this TC message reaches the other end of the connection at once.

We also have to consider what happens when a TC indication message is received on one side (User-1 or User-2 side): this TC message has to be mapped onto a MAP message, in order to be sent to the concerned user, as a MAP indication (this type of messages will be detailed later). Let's continue with our example:

- the MAP corresponding to User-2 receives a TC_BEGIN indication, containing some component handling primitives (if we assume that User-1 wanted some services to be performed). This TC message will become a MAP_OPEN indication, followed by user specific service primitives corresponding to the services requested by User-1, followed by a MAP_-DELIMITER indication, for User-2 to know that the list of services is over.

The detailed mapping is to be seen in Part 4.

7 the MAP services

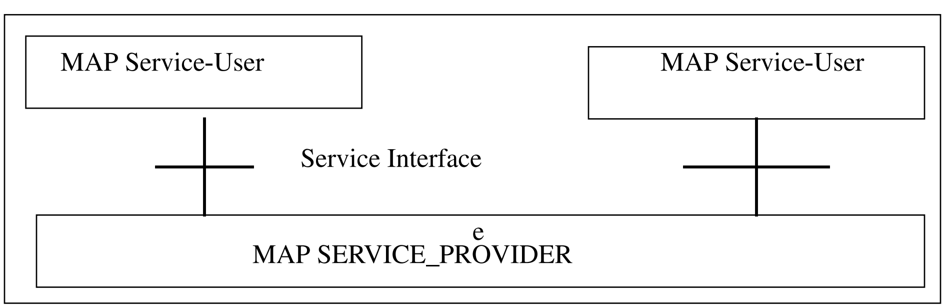
The MAP provides some services to its users, which are GSM components (MSCs, HLRs, VLRs). There is a great amount of various services, in so far as the three components that make use of the MAP are involved in a lot of GSM functions and procedures...

Among them, we can define two sorts of services: the MAP common services, which are related to dialogue handling, and don't correspond to a specific MAP user. On the contrary, the MAP user specific services can not be requested by any kind of user; they correspond to a GSM function, performed by a given component.

Among the second group of services, there are some which can be grouped because they correspond to one context (this context is not to be confused with the Application Contexts we will talk about later); for example, they all deal with Location Updating procedures... we will now go further into details, concerning these numerous services.

7.1 the basic modelling principles

From now on, we will make use of a model representing the MAP, which is defined in the Mobile Application Part Specification book (from the European Telecommunications Standards Institute), in order to make the MAP easier to understand, and above all, to be able to specify it. Basically, this model is supposed to lead to an SDL specification, but we will use it for the specification in Lotos.



MAP provides its users with a specified set of services and can be viewed by its users as a “black box” or abstract machine representing the MAP service-provider. Between the MAP service-provider and the MAP service-user, there is an interface called the service interface.

The MAP service-users interact with the MAP service-provider by issuing or receiving MAP service-primitives at the service interface. A MAP service-user may receive services from several instances of the MAP service-provider at the same time. In such cases the overall procedure is synchronized by the service-user.

The MAP service-primitives are named using the following denotation:

MAP-ServicePrimitiveName type

where type can be any of: request (req), indication (ind), response (resp), confirm (cnf). We will explain later what these types correspond to.

The services are further classified as unconfirmed-service, confirmed-service and provider-initiated service, where the first two categories refer to wether or not the service is confirmed by the service-provider.

As we already mentioned, MAP services are also classified as common MAP services which are available to all MAP service-users, and MAP service-user specific services which are services available to one or sometimes several, but not all, MAP service-users. The next paragraph is about the MAP common services.

7.2 the common services

7.2.1 An overview

All MAP service-users require access to services for performing basic application layer functions:

- for establishing and clearing MAP dialogues between peer MAP users

- for accessing functions supported by layers below the application layer (in the case of MAP, that means TCAP services)

- for reporting abnormal situations

- for testing wether or not a MAP dialogue is still active at each side.

For these purpose, the following Common Services are defined:

- MAP-OPEN service: for a user to open a dialogue

- MAP-CLOSE service: for a user to close a dialogue

- MAP-DELIMITER service: for a TCAP message to be sent (a MAP-DELIMITER message is sent by a MAP-user, indicating the end of a list)

- MAP-U-ABORT service: when a MAP user decides to abort a dialogue (it can be anyone of the two MAP users that are involved in the dialogue)

- MAP-P-ABORT service: when the MAP services provider (that appears as a black box in our metallization) decides to abort a dialogue

- MAP-NOTICE service: used to notify the MAP service-user about protocol problems related to a MAP dialogue, not affecting the state of the protocol machines: it's up to the MAP user to decide wether to abort the dialogue or not.

We won't go into details about those services; what interests us is only their effect on the Dialogue State Machine. Therefore, we will just mention a few features which are worth noticing:

- These services are rather simple, compared to the user-specific services, in so far as they have very few parameters and they are unconfirmed (no response from the peer entity). The only exception to this rule concerns the MAP-OPEN service, which is a confirm service: the parameter 'result' which inclusion is mandatory in the confirmation indicates if the dialogue is accepted by the peer or not.

- the MAP-DELIMITER service has no parameter, in so far as its role is only to explicitly request the transfer of the MAP protocol data units to the peer entities.

7.2.2 Sequencing of common services

The typical sequencing of services that manages a MAP dialogue is as follows:

- **Opening**: the MAP-OPEN service is invoked before any user specific service-primitive is accepted. The sequence may contain none, one or several user specific service primitives, that will be sent in the same TC message with the MAP-OPEN request. If no user specific service-primitive is contained between the MAP-OPEN and the MAP-DELIMITER primitives, then this will correspond to sending an empty Begin message in TCAP. If more than one user specific service-primitive is included, all are to be sent in the same TC-BEGIN message, as we said before (the sequence of user specific services ends with a MAP-DELIMITER request.)

- **continuing**: This sequence may not be present in some MAP dialogues. If it is present, it ends with a MAP-DELIMITER primitive. If more than one user-specific service-primitive is included, all are to be included in the same TC-CONTINUE message.

- **Closing**: The sequence can only appear after an opening sequence or a continuing sequence. The closing sequence may contain none, one or several user specific service-primitives, provided the MAP-CLOSE primitive specifies normal release (which means that there is no pre-arranged release: when the dialogue is closed by one of the 2 MAP users, the peer entity receives a corresponding indication.) If no user specific service-primitive is included, then this will correspond to sending an empty TC-END message in TCAP. If more than one user specific service-primitive is included, all are to be sent in the same TC-END message. If prearranged end is specified, there cannot be any user specific service-primitive. The MAP-CLOSE primitive must be sent after all user specific service-primitive have been delivered to the MAP service-provider.

- **Aborting**: a MAP user can issue a MAP-U-ABORT request at any time after the dialogue has been opened, or as a response to an attempt to open a MAP dialogue.

the MAP provider may issue at any time a MAP-P-ABORT primitive to a MAP service-user for which a MAP dialogue exists.

These two ABORT primitives terminate the MAP dialogue.

7.2.3 General rules for mapping common services onto TC messages

We talked a bit about mapping services onto TC; but we didn't give the rules.

With a table, we will see how each MAP common service becomes a TC primitive to be sent to the peer entity. The parameters used in the MAP services are mapped onto TC primitives corresponding parameters.

TABLE 1.

MAP service-primitive	TC service-primitive
MAP-OPEN request (+ any user specific primitive) + MAP-DELIMITER request	TC-BEGIN request (+ components handling primitives corresponding to user specific primitives)
MAP-OPEN response (+ any user specific primitive) + MAP-DELIMITER request	TC-CONTINUE request (+ components handling primitives)
(any user specific primitive) + MAP-DELIMITER request	TC-CONTINUE request (+ components handling primitives)
(any user specific primitive) + MAP-CLOSE request	TC-END request (+ components handling primitives)
MAP-U-ABORT request	TC-U-ABORT request

We also need a table for mapping TC primitives onto MAP service-primitives: when a TC indication occurs on one side, the MAP must be able to translate it in order to make use of it (process the requested services, or send some indication or confirmation to the corresponding user.)

TABLE 2.

TC service-primitive	MAP service-primitive
TC-BEGIN indication	First time:
(+ component handling primitives)	MAP-OPEN confirm
	(+user specific service primitives)
	+ MAP-DELIMITER indication
	Subsequent times:
	(user service primitives)
	+ MAP-DELIMITER indication
TC-END indication	MAP-OPEN confirm
(+ component handling primitives)	(+ user specific service primitives)
	+MAP-CLOSE indication
TC-U-ABORT indication	MAP-U-ABORT indication or
	MAP-P-ABORT indication (depending on wether the TC indication contains MAP-abort-PDU or a MAP-user-abort-PDU)
TC-P-ABORT indication	MAP-P-ABORT indication

7.3 the MAP user-specific services

These MAP user specific services are numerous, and we are not going to talk about them individually. Furthermore, each of them would have to be specified separately from the others, because it has its own parameters which depend on the specific service; the only common parameter shared by all MAP specific services is the Invoke Id (mandatory), a number that identifies uniquely the MAP-user currently requesting this service. If the service is confirmed, the confirmation will also carry the same service Id, so that the MAP identifies the right user.

7.3.1 The different sorts of services

First, we could divide the MAP services into 2 parts:

- the services that concern the Mobility Management Layer

- the services that concern the Communication Management layer.

We could also consider that all the services that can be requested during a dialogue between 2 specific components (HLR and MSC, for example) constitute one group of services. But the MAP standard chose an other way to divide the services into groups, putting together services used for procedures that are involved in similar processes. there are five types of services, but we will consider in this work four of them, which are:

- Mobility Services (Location registration services, handover services)

- Operation and Maintenance Services

- Call Handling Services.

The other types are related to Short Messages and Supplementary Services that involve components we decided to ignore (GatewayMSC and SMS_gateway).

To group the specific services in a more precise and useful way, let's introduce a new concept called an Application Context: it is very convenient to use it in so far as one application context correspond to a dialogue between 2 specific MAP users that was opened to perform a common task, which means that the service-primitives must belong to the group of services needed to perform the specific task: this group is an application context. Let's list them (only the ones that imply MSC, VLR or HLR):

-Access Location Updating: between MSC and VLR (location management procedures)

- **Location Updating**: between HLR and VLR (location updating procedures)

- **Location Cancellation**: between HLR and VLR (location cancellation procedures)

- **Roaming Number Inquiry**: between HLR and VLR (the roaming number is used to route calls)

- **Incoming Call Handling**: between MSC and VLR (incoming call set-up procedures)

- **Outgoing Call Setup**: between MSC and VLR (outgoing call set-up procedures)

- **Access Functional SS Handling**: between MSC and VLR (handling Functional like Supplementary Services procedures)

- **Access SS Invocation**: between MSC and VLR (handling Supplementary Services invocation procedures)

- **Access unstructured SS Handling**: between MSC and VLR (handling stimuli-like procedures)

- **Handover Control**: between MSC and MSC (for handover procedures)

- **Handover Number Allocation**: between MSC and VLR

- **Information Retrieval**: between HLR and VLR

- **Inter-VLR Information Retrieval**: between VLR and VLR

- **Stand Alone Subscriber Data Management**: between HLR and VLR

-**Network Functional SS Handling**: between VLR and HLR (for Functional like SS handling procedures)

- **Network Unstructured SS Handling**: between HLR and VLR (for stimuli like SS procedures).

As we said before, each Application Context (AC) groups the services that allow 2 components to perform a common task. Therefore, there are several services for one AC. To each dialogue established by a MAP-user is associated an AC, which is supplied in the MAP-OPEN req primitive by the MAP-user and transmitted to the peer under certain circumstances; if it isn't transmitted, the peer MAP machine will have to derive an AC from the operation code of the first service primitive of the TC-BEGIN message, as we will see later in our Lotos specification.

We're going to give an example of a MAP user specific service; this example is to be used later.

7.3.2 the MAP_SEND_END_SIGNAL service

This service is used at the E interface, that is, between 2 MSCs (the Anchor MSC and the Relay MSC). It indicates to the anchor MSC that the radio path between the MS and the relay MSC (the new MSC) has been established. This service is a confirmed service: the response is used for the anchor MSC to inform the relay MSC that the call is released, so all resources for the call can be released in the relay MSC.

This service belongs to an Application Context named Handover_Control. As we can see in the table, there are only two parameters, and the only mandatory one is the Invoke Id (a number that links the service-primitive to the user that requested it). The second one is optional, and can be used to indicate a protocol related type of error (such as: no response from the peer).

Parameter Name	Request	Indication	Response	Confirm
Invoke Id	Mandatory	Mandatory(=)	Mandatory(=)	Mandatory(=)
Provider error				optional

This particular service is very simple; the other MAP-services need generally more than two parameters, and many of them are mandatory. Anyway, the principles are the same, no matter the number and the sort of parameters.

In the following part, we're going to focus on MAP Dialogue State Machine, and for a moment we will forget user specific services, in so far as DSM deals with the MAP Common services.

PART 4: the Dialogue State Machine (DSM)

Partly because of the collaboration that was supposed to take place between Toma Macavei and myself, leading to a comparison between test cases generated from a SDL specification of the MAP and test cases generated from a Lotos specification, I decided to start specifying the DSM in as much as Toma Macavei was working on this part of the MAP. However, a five months training period wouldn't have given me enough time to specify all the MAP...

I also decided to follow the metallization presented in the ETSI book (leading to an SDL specification) for my specification to use the same concepts as an SDL specification (building my processes with Finite State Machines), for a comparison between the 2 specifications to be possible.

The Dialogue State Machine handles one specific dialogue and simulates the coordination between Common Map services and their sequencing (opening, continuing, closing). It manages message flows associated with User-Specific services, sending the appropriate messages to the peer entity (through TCAP); the DSM must also instantiate other machines that must be instantiate for each User-Specific service, as will be explained in the first paragraph.

8 The MAP model

The MAP protocol machine (MAP PM) can be modelled as a collection of service state machines, named SSMs. There is one SSM per MAP user specific service invoked. SSMs are coordinated by a MAP dialogue control function with its MAP Dialogue State Machine.

There are two types of Service State Machines: Requesting Service State Machine(RSM) and Performing Service State Machine (PSM).

A new invocation of a MAP PM is employed on the receipt of a MAP-OPEN request primitive or a TC-BEGIN indication primitive. Each invocation controls exactly one MAP dialogue. For each MAP specific service invoked during a dialogue, a MAP RSM is created at the requestor's side and a MAP PSM is created at the performer's side.

How the MAP user and the MAP refer to a MAP dialogue -i.e. a MAP PM invocation- is an implementation matter. The way TC Dialogue Identifiers (i.e. parameters in TCAP messages that refer to a specific dialogue) are assigned to a MAP PM invocation is also an implementation matter.

This modelling is used only to facilitate understanding and to organize SDL descriptions; it is not intended to suggest any implementation.

Apart from that, the MAP model we will choose to built the Lotos Specification will respect a few conventions, that we're not going to explain in details; let's just mention the fact that there are two kinds of Application Contexts:

- **AC belonging to the "Version One Context Set"** (V1). If the AC corresponding to a dialogue belongs to V1, its transfer by TC to the peer entity is not required: therefore, when receiving the Operation Code (corresponding to the service primitive), the peer DSM has to derive the Application Context from this operation code.

- **AC not belonging to this V1 context set**. Then, the MAP requests the transfer of the Application Context name by TCAP.

At last, let's talk a bit more about the Service State Machines and the problem they raise: for each specific service-primitive, one Requesting State Machine is instantiate. As we said in Part 3, each service has its specific parameters. The Requesting State Machine will have to send a TC_INVOKE message to the peer, containing of course all those specific parameters. Therefore, there must be one specific RSM per specific service. It is the same regarding PSMs. In the SDL specification of the Dialogue State Machine, this is not a problem: when the SSM corresponding to a specific service has to be instantiate, the one that needs to be instantiate is not specified: there is only a message sent to some kind of generic SSM (generic RSM or generic PSM).

To explain the reason why we use the word generic, let's take the example of RSMs: each one needs to send a specific TC_INVOKE request. However, in the SDL specification, we just see that a TC_INVOKE message is sent, without its parameters being known. That allows to specify all RSMs in one, but it remains very abstract as much as it doesn't take into account the differences between services parameters.

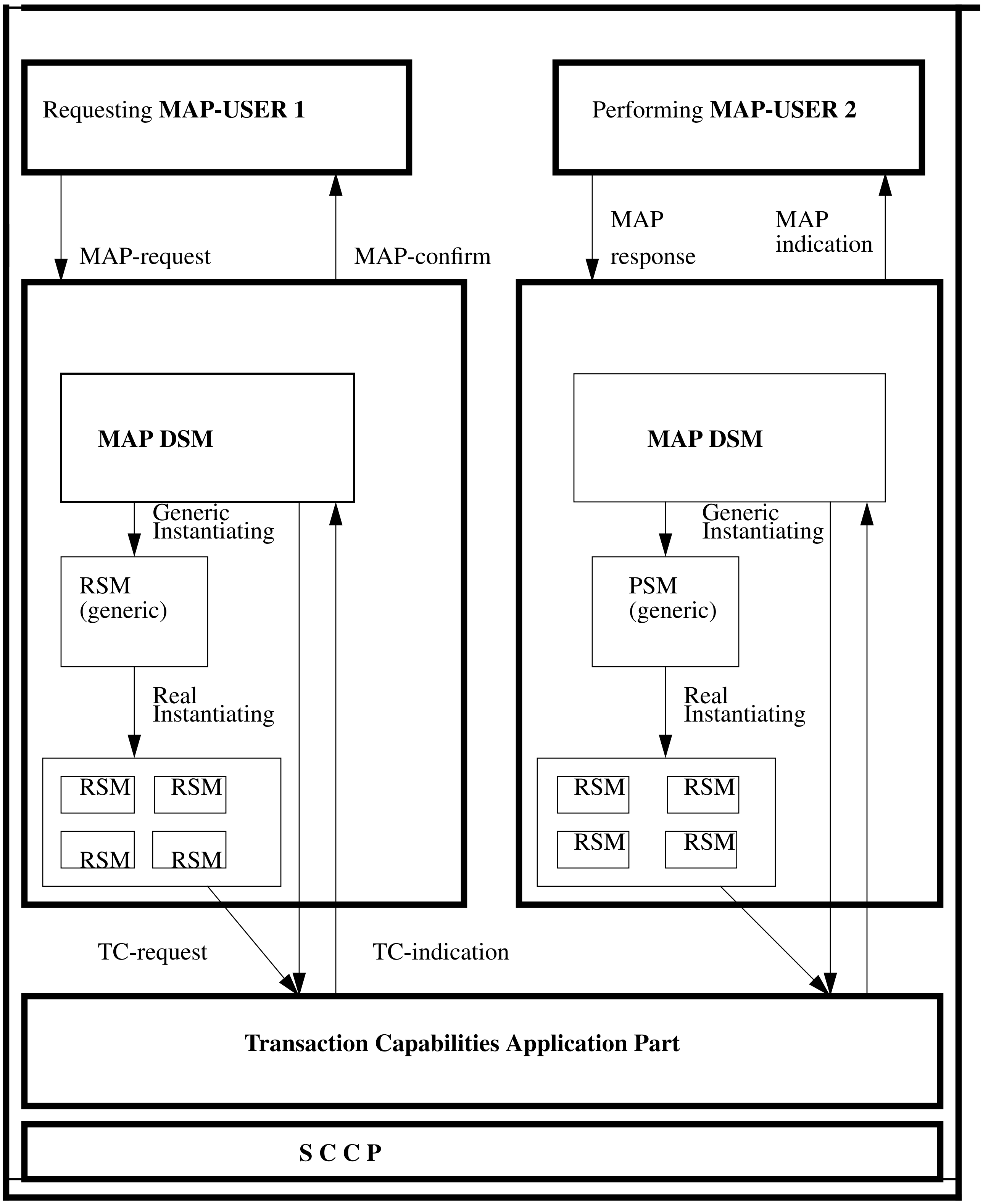
In our Lotos specification of the DSM, we could also remain at a very abstract level and send an Instantiating message to some generic SSMs... however, the problem is that this solution isn't appropriate to generate Test Cases (that will be explained later): it appears to be necessary to take into account the differences between TC-INVOKE messages that are sent, with their own parameters depending on the Operation Code (i.e. on the specific service).

In the next part, we will explain how to cope with this problem.

9 The Dialogue State Machine and its environment.

In the following figure, we assumed that MAP User 1 was the requesting user and MAP User 2 the performing one; however, after the opening of a dialogue, the situation becomes symmetric: both MAP Users can request services. Therefore, User 1 and User 2 can exchange their roles.

To remain as close as possible to the SDL specification of MAP DSM, and to the model proposed by the MAP book, we introduced a state machine named "generic RSM" (or PSM), which function is only to instantiate the right RSM. This state machine will also appear in our Lotos specification, named "RSM manager" (on the performing side, "PSM manager). It is an internal machine that doesn't exchange any message with the MAP-User neither with TCAP.


Requesting MAP-USER 1
Performing MAP-USER 2
MAP-request
MAP-confirm
MAP
response
MAP
indication
MAP DSM
MAP DSM
Generic
Instantiating
Generic
Instantiating
RSM
(generic)
PSM
(generic)
Real
Instantiating
Real
Instantiating
RSM
RSM
RSM
RSM
RSM
RSM
RSM
RSM
TC-request
TC-indication
Transaction Capabilities Application Part
S C C P

8 the SDL specification of the MAP

As we said, the SDL specification will be based on the MAP model we've just explained.

It is a graphic specification. The abstraction level remains high which allows not to take into account the differences (due to parameters) between TC-INVOKE requests sent by specific RSMs to TCAP.

8.1 The main blocs of the specification

The whole SDL specification is divided into 3 blocs:

- the **MAP-PROVIDER** bloc

- the **MAP-USER** bloc

- the **TCAP** bloc

Of course, only the MAP-PROVIDER bloc interests us. From now on, we will focus on the MAP-PROVIDER. It communicates with the MAP-USER bloc and with TCAP bloc through channels, as shown on the following figure.

8.2 Architecture of the MAP-PROVIDER bloc

Once again, there is a division into blocs inside the MAP-PROVIDER bloc:

- **MAP DSM** bloc: this type of process handles a dialogue. There exists one process instance per MAP dialogue.

- **PERFORMING MAP SSM** bloc: this type of process handle a MAP service performed during a dialogue. Instance of this process are created by an instance of the MAP_DSM process (on the performing side) for each MAP-service to be performed.

- **REQUESTING MAP RSM** bloc: this type of process handle a MAP service requested during a dialogue. Instance of this process are created by an instance of the MAP_DSM process (on the requesting side) for each requested MAP-service.

- **LOAD-CONTROL** bloc: this type of process is in charge of load control; there is one instance of this process in each system. For our work, we will assume that there is no overload problem, and this LOAD-CONTROL bloc won't be taken into account.

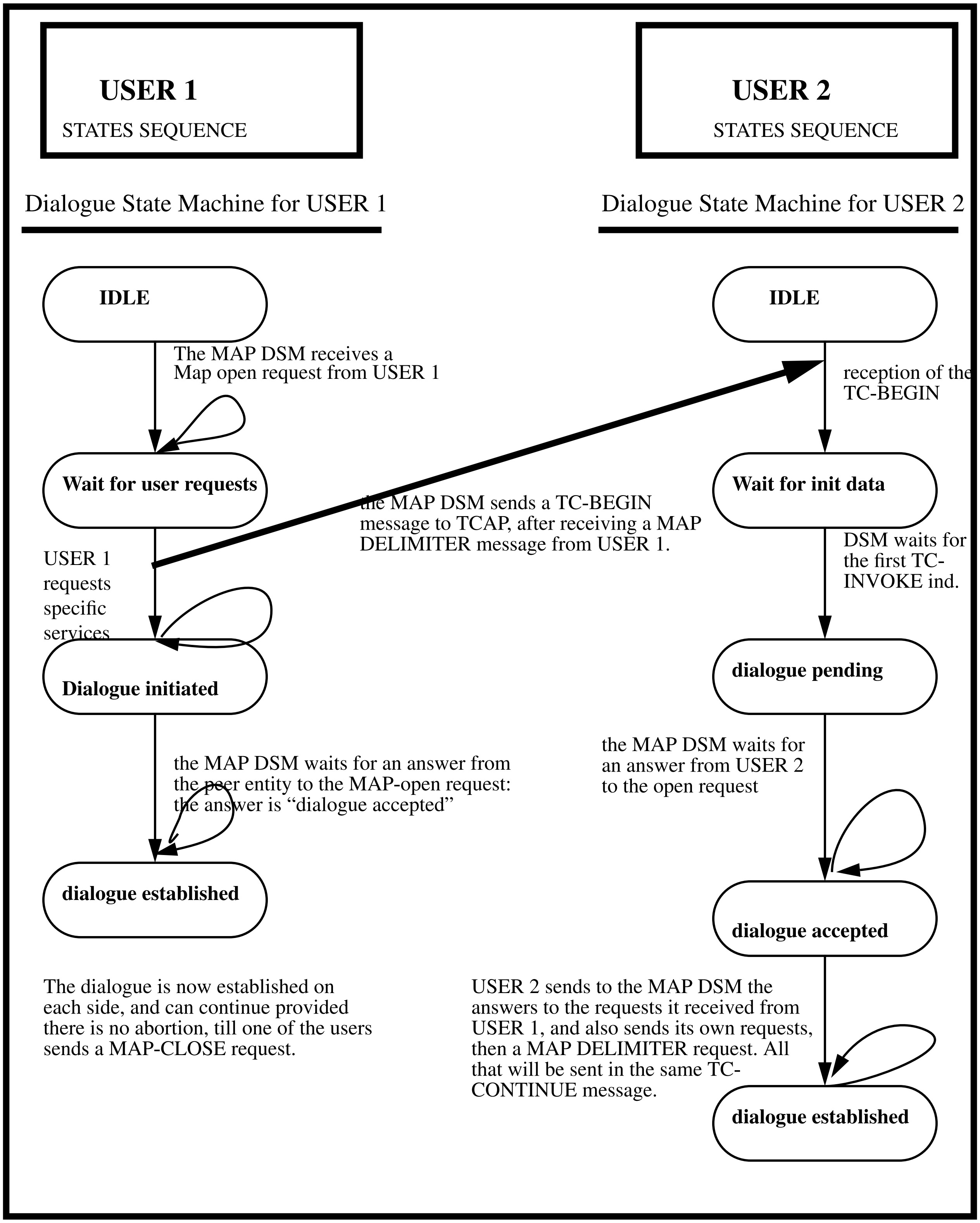
Concerning the MAP-DSM specification, the several state machines that build it will be detailed in the next paragraph, through examples.

It is important to notice that there is only one DSM specification, whereas the Dialogue State Machine has different roles and functions, depending on wether it is the requesting side DSM, or the performing side; and wether it corresponds to the user that requests the opening of the dialogue or not. Therefore, the SDL specification of the DSM has to take into account all the different functions a DSM may have to perform, which makes it a bit complicated and uneasy to understand.

8.3 The State Machines

The MAP DSM bloc is a big State Machine; the actions that allow to go from one state to another are most of the time quite complicated: messages are received and sent, data are checked (for instance, the State Machine must make sure that the Application Context has not changed in one dialogue), etc...

A MAP dialogue is seen as a step by step process, each step corresponding to a specific state. Before giving a list of the different states of the DSM machine, but we will try to understand the way it works with an example: let's assume that a MAP user ("User 1") wants to open a dialogue with another MAP user ("User 2"). One of the possible states sequences is the following one (we assume that the opening is successful):


USER 1
STATES SEQUENCE
USER 2
STATES SEQUENCE
Dialogue State Machine for USER 1
Dialogue State Machine for USER 2
IDLE
The MAP DSM receives a Map open request from USER 1
Wait for user requests
USER 1 requests specific services
Dialogue initiated
the MAP DSM waits for an answer from the peer entity to the MAP-open request: the answer is “dialogue accepted”
dialogue established
The dialogue is now established on each side, and can continue provided there is no abortion, till one of the users sends a MAP-CLOSE request.
the MAP DSM sends a TC-BEGIN message to TCAP, after receiving a MAP DELIMITER message from USER 1.
IDLE
reception of the TC-BEGIN
Wait for init data
DSM waits for the first TC-INVOKE ind.
dialogue pending
the MAP DSM waits for an answer from USER 2 to the open request
dialogue accepted
USER 2 sends to the MAP DSM the answers to the requests it received from USER 1, and also sends its own requests, then a MAP DELIMITER request. All that will be sent in the same TC-CONTINUE message.
dialogue established

We are now going to explain a few things about this figure, and try to define the different states that appear on it (which are the only possible states of the DSM State Machine).

- **IDLE** state: it's the initial state (when DSM has not been instantiate) and also the final state (the DSM Instantiating is killed).

- **Wait For User Request** state: this state is reached by the user who wants to open the dialogue, after he sent a MAP-OPEN request which can be followed by a list of user-specific requests (this list may be empty), till the user sends a MAP-DELIMITER request: a TC-BEGIN message is built from the MAP-OPEN request and the following list of services, and sent to the peer through TCAP layer. The DSM corresponding to this user (User 1) reaches the next state:

- **Dialogue Initiated**: this state is reached by the user who wants to open a dialogue once the TC-BEGIN message has been sent.

- **Dialogue Established**:

- Concerning User 1: this state is reached from the Dialogue Initiated state, when the corresponding MAP DSM receives the first TC-CONTINUE message from the peer, indicating that the dialogue has been accepted, and sending some user-specific services to be performed. these services are handled by their specific PSMs, and a TC-CONTINUE request can be sent by User 1 DSM, containing responses and a new list of services requested by User 1, etc...

- Concerning User 2: the dialogue is established after User 2 has given its own list of services to be performed, and sent a MAP-DELIMITER request to close this list. The first TC_CONTINUE request is then sent to the peer, and the User 2 DSM changes state and goes from "Dialogue Accepted" to "Dialogue Established".

- **Wait For Init Data**: this state is reached by User 2 DSM when it receives the TC-BEGIN indication, if the Application Context is not included in this message (which is generally the case). The DSM has to derive the right Application Context from the first TC-INVOKE indication (corresponding to the first service to be performed): thanks to the Operation Code, it will find the AC.

- **Dialogue Pending**: once the AC has been found, a MAP-OPEN indication is sent to User 2.

- **Dialogue Accepted**: this state is reached once the MAP-OPEN response has been issued by User 2 (and says that the dialogue is accepted). The following state is “Dialogue Established”.

In the next part, we will explain how the Lotos specification was built from this concepts of State Machines, in order to keep with the SDL specification. Then, we will explore the problem raised by Test Cases Generation using Random Walk, and finally give the test sequences that we found.

PART 5: LOTOS specification of the Dialogue State Machine and Test Cases generation with Random Walk

In this part, we attend to present our work on the MAP, and the results it leaded to.

First we will talk a bit about the first specification that was written following the SDL specification, and we will explain why it seems quite impossible to use concerning Test Cases Generation; then we will explain how the second specification was written in order to be able to use Random Walk and obtain interesting Test Cases, in the particular case of one MAP user-specific service.

9 A global specification of the DSM

The first specification of the DSM was written at the same abstraction level as the SDL specification, because it was the only way to make a global specification: there are too many different User-Specific services to reach a lower abstraction level, because that would lead to consider each particular case (i.e. each particular MAP service). Therefore, this specification remains very “generic” and does not allow to do Test Cases generation.

9.1 The Dialogue State Machine and its environment

Before starting to build a specification, the first thing to do is to define the limits between what will be specified, and what will be considered as part of the environment.

In this specification, we decided to consider only the DSM. As a consequence, it is up to the specification user to simulate the TCAP layer and the Service State Machines, which implies that the user has to be careful about the parameters contained in a message: for instance if a TC-BEGIN request is sent with a list of services, the TC-BEGIN indication must contain the same list.

The reason why it appeared to be much too complicated to specify TCAP functions and SSMs functions is the number of the MAP specific services. We are now going to explain with more details the problem we had to cope with.

Each MAP User specific service has its own parameters (which can be numerous), that are to be included in the TC_INVOKE message sent by the corresponding RSM. That makes a lot of different TC_INVOKE messages possible, all the more as some parameters are optionals, which increases this number. Therefore, it appeared to be too long to specify all RSMs and the different TC_INVOKE messages that each RSM could send. Thus, in our global specification, it's up to a process called "RSM manager" (which does not belong to the specification but communicates with it through a gate) to instantiate the right RSM when a service is requested. It is the same concerning PSMs. <u>All Service State Machines are considered as part of the environment</u>, Furthermore the specification can not access them directly.

The same reasons made it very difficult to include a simulation of TCAP layer in the specification, in as much as it is impossible to link a TC_INVOKE indication received on the performing side, to the corresponding TC_INVOKE request issued at the requesting side, because we decided not to specify TC_INVOKE requests. Concerning TC_INVOKE indications, the problem is that we could not afford not to mention them because the Operation Code they contain is needed for DSM functions. We decided to create a TC message that doesn't actually exist: a TC_IN-

VOKE indication with only one parameter (the Operation Code). So, TCAP is also completely part of the environment.

As a consequence, the specification user who wants to execute it has to simulate the synchronization between the two involved entities (the Requesting side and the Performing side). He has to do it for every TC message, in order to link the TC_request with the corresponding TC_indication: TC_BEGIN, TC_CONTINUE and TC_END requests may contain a list of services to be performed that the specification user will have to memorize and repeat in the corresponding TC indications.

Obviously, such a specification is not very interesting because of this lack of synchronization; it doesn't really help understanding and simulating DSM's behaviour.

9.2 Global architecture of the specification

9.2.1 the specification ‘s style

A specification can be written in 3 different styles, depending on the purpose of this specification:

Constraint-oriented style:

In this style, the temporal ordering of externally observable events is a composition of a certain number of constraints [FLS91]. The specifier only focuses on the external behavior of the system and does not try to represent the internal architecture of the system. This style may be recommended if the specification is written for analysis purpose, since the analysis step must be independent of the architecture of the system.In this style, the temporal ordering of externally observable events is a composition of a certain number of constraints [FLS91]. The specifier only focuses on the external behavior of the system and does not try to represent the internal architecture of the system. This style may be recommended if the specification is written for analysis purpose, since the analysis step must be independent of the architecture of the system.

Resource-oriented style:

In the resource-oriented style, the system is designed as a composition of resources whose functionalities are well-defined. Each resource can again be specified in another style. This style can be recommended for a designer whose purpose is to express an architecture of the system. The GSM informal specifications, for instance, specifies not only the behavior of the GSM but the architecture of the system as well. In this sense, the resource-oriented style is recommended if readability is a concern.

Monolithic and state-oriented style:

The structure is based on state and event sequencing. Specifications in this style are close to the structure of protocol implementations.

In a way, the MAP DSM specification we wrote can be described as State-Oriented because it is actually based on state and event sequencing; but those states and events are not observable by a MAP-user: they are internal and concern internal resources. Therefore, this specification is rather <u>resource-oriented</u> than state-oriented.

9.2.2 the processes and their composition

- The main process: this process is called “DIALOGUE”; one Instantiating of this process corresponds to one Instantiating of the MAP DSM. To specify a dialogue between 2 MAP Users (which is the purpose of the specification), we used the **interleave operator** (|||) between 2 DIALOGUE processes, which means that there is no synchronization (this is due to what we explained previously.):

behavior

DIALOGUE [a1, b1, c1, d1...]

|||

DIALOGUE [a2, b2, c2, d2...]

- The DIALOGUE process is composed of several processes following SDL State Machine model: one process corresponds to the transition between 2 states that follow each other. Therefore DIALOGUE is built as follows (the enable operator corresponds to the DSM entering a new state):

process DIALOGUE [....]**: noexit:=**

P1 **>>** P2 **>>**P3 **>>** P4......(* USER 1*)

[]

P’1 **>>** P’2 **>>** P’3...... (*USER 2 *)

We have the **choice** operator which is needed because DIALOGUE will be instantiated by the user that wants to open the dialogue, and by the user that receives this opening request.

We are now leaving this global specification for a very partial DSM specification that was finally the only one we could use to generate Test Cases using the “Random Walk” tool.

10 A partial specification for Test Cases generation

In order to generate Test Cases with the “Random Walk” tool, we had to write this new specification that focuses on a particular MAP service, and describes DSM functions when considering the case of a dialogue in which this service is the first one to be requested by the opening user. As we will see in the next paragraph, Random Walk can not work properly on the global specification we first wrote.

10.1 Random Walk and Test Cases generation

Random Walk is a tool that gives possible scenarios by walking randomly through the specification, till a deadlock is reached. The word “Random” means that when several actions are offered in a behavior expression, one of them will be randomly chosen and the other actions are not possible anymore. The fact that all actions have the same probability to be chosen is a very important point, that makes this tool very difficult to use: for instance, if 3 actions are offered and only one corresponds to a “normal” behavior (the 2 others corresponding to errors or very particular cases), there is only one chance out of 3 that the “interesting” action is going to be chosen: therefore, there will only be one scenario out of 3 generated by Random Walk that will correspond to the “normal” behavior... thus, the Test Cases obtained this way don’t match the real situation.

Concerning our global specification, the first problem and the most important one that prevented us from using Random Walk is the absence of synchronization during a dialogue. As long as there was a specification user executing the specification step by step and filling TC-indications messages with the parameters corresponding to TC-requests, it worked all right. The problem is that this user had a very important function, simulating the whole environment (such as TCAP layer); it was up to him to synchronize the 2 DSM machines (one per user). Without him, our specification can not be executed. However, Random Walk doesn't allow anyone to interfere when running the specification, and chose the right parameters: when a message needs parameters, it just chooses them randomly among a list of values (a "pool"). Therefore, our specification doesn't make sense anymore because the services received on the performing side are not the ones sent at the requesting side!

The lack of synchronization raises a lot of other very big problems: because of the interleave (|||) operator between the 2 Instantiations of the DIALOGUE process, User 1 can try to open a dialogue with User 2, while User 2 tries to open a dialogue with User 1, which leads to terribly confusing situations that soon become completely meaningless!

For all those reasons (and others), it is obviously impossible to run Random Walk on this specification.

10.2 the MAP SEND_END_SIGNAL service

This MAP user-specific service is the one we chose for our partial specification. We already described it, as an example of MAP service, in Part 3. Let's just recall that it is a confirmed service between an Anchor MSC and a Relay MSC, with only one parameter involved which is the Invoke-Id.

It belongs to the "Handover Control" Application Context, that contains 6 more services: PerformHandover, ProcessAccessSignalling, ForwardAccessSignalling, PerformSubsequentHandover, AllocateHandoverNumber, SendHandoverReport; their operation code belongs to [28..34] and thanks to it, we can check wether a service belongs to this A.C. or not.

We are not going to describe the six other services, this is not our purpose. The particular case of a dialogue in the Handover Control Application Context, starting with a SendEndSignal user-specific service request, was specified in order to see if it was possible to generate Test Cases, considering a very specific part of the global specification. We could have chosen another Application Context and another MAP User specific service.

10.3 Several assumptions

The partial specification we wrote has a lot in common with the global specification: data types are the same, and the general architecture is also the same (except for the synchronization processes we defined: that will be explained later); but there are some more assumptions we had to set for Random Walk to generate interesting sequences:

- the dialogue can only occur between an MSC and another MSC (in the specification, they are called User 1 and User 2).

- the Application context must be “Handover Control”.

- there is only one dialogue at a time: therefore, there are only 2 instantiations of the DIALOGUE process at a time.

- there are 2 users only (User 1 and User 2), so there can be no ambiguity concerning the destination of a TC message.

10.4 the synchronization processes

10.4.1 the “SYNCHRO” process

To introduce the synchronization we need between DSM-1 (i.e User 1 Dialogue State Machine) and DSM 2, which in fact implies a simulation of TCAP function, we decided to create a new process named SYNCHRO: SYNCRO [TC_I, TC_R]: noexit.

This process synchronizes on the gates TC_R_1, TC_R_2, TC_I_1, TC_I_2, with the behaviour previously defined in the global specification. That results in the following behavior:

(

DIALOGUE [REQ_1, CONF_1, RESP_1, IND_1, RSM_1, PSM_1, TC_I_1, TC_R_1]

|||

DIALOGUE [REQ_2, CONF_2, RESP_2, IND_2, RSM_2, PSM_2, TC_I_2, TC_R_2]

)

|[TC_I_1, TC_R_1, TC_I_2, TC_R_2]|

(

SYNCHRO [TC_I_1, TC_R_2] (* executed when User 1 is the requesting side*)

|||

SYNCHRO [TC_I_2, TC_R_1] (* executed when User 2 is the performing side*)

)

What happens is that when a TC_Request is issued on the requesting side, the SYNCRO process synchronizes with the DSM of the requesting user, and offers its next action which is a TC_Indication (corresponding to the request) on the performing user's TC_I gate. To know the precise definition of this process, see the specification.

10.4.2 the "OPEN_SYNCHRO" process

The synchronization process we've just described is not enough to ensure that the opening of the dialogue will work properly: we could have 2 simultaneous dialogues, and this is a situation that we want to avoid. Let's consider the following sequence:

Possible actions: User 1 sends a MAP_OPEN_req to DSM_1, or User-2 sends a MAP_OPEN_req to DSM_2

-User-1 sends a MAP_OPEN-request to DSM_1

Possible actions: the next action offered by the DIALOGUE process Instantiating corresponding to User-1, or User-2 sends a MAP_OPEN_-req to DSM_1.

- User-2 sends a MAP_OPEN_req to DSM_2

That results in all kinds of problems raised by theses simultaneous opening attempts.

Thus, we had to define another synchronization process:

OPEN_SYNCHRO [REQ, TC_I]

This process will force User-2 to enter a state named "Initial State" from which there is no way for the user to send a MAP_OPEN_req. (See the specification). The previously defined behaviour is changed into:

behavior

(

DIALOGUE [.............]

|||

DIALOGUE [...............]

)

|[REQ_1, REQ_2, TC_I_1, TC_R_1, TC_I_2, TC_R_2]|

(

SYNCHRO [TC_I_1, TC_R_2]

|||

SYNCHRO [TC_I_2, TC_R_1]

|||

(

OPEN_SYNCHRO [REQ_1, TC_I_2]

[](*we used the choice operator to ensure that there would be only one opening attempt: once a user has sent a Map-Open request, a choice is done and it is no more possible for the other user to open a dialogue.*)

OPEN_SYNCHRO [REQ_2, TC_I_1]

)

)

Once a MAP user (User 1 or User 2) has sent a MAP_OPEN request to its own DSM, the other user has to synchronize on the action TC_I!Initial-State, and this action forces the corresponding DIALOGUE process to one of its possible behavior expressions (at the beginning, there are 3 possible behavior expressions composed with the choice operator), and that eliminates the possibility to send a MAP_OPEN request.

11 Test Cases generation: a few results

Running Random Walk on the second specification, we obtained some interesting Test Sequences, even though one problem still remained. This problem is raised by Random Walk rather than by our specification: among the possible actions offered during a dialogue, the ones that lead to an abortion are numerous, whereas there may not be so frequent dialogue abortion in reality; but because of the Random principle, the Test Sequences we obtained are most of the time rather short. There is always a high probability to pick an action that leads to an abortion, and a low one to continue the dialogue...

Let's give one example among the sequences we got:

Sequence No 1:

- REQ_2!MAP_OPEN_req:Common-Services!Handover-Control:AC!-MSCA1:Origin

(* User-2 = MSCA1 opens the dialogue*)

- REQ_2!SendEndSignal:op_code

(* We assumed that all dialogue started with this request from the opening user*)

- i/exit (Handover_Control:AC, MSCA1:Origin)

(* internal exit with parameters that enable the process “Dialogue -Initiation” *)

- TC_I_1!Initial-State: message

(* Because of the synchronization that occurred on REQ_2 with OPEN_-SYNCHRO, DSM_1 can only execute this action *)

- REQ_2!MAP_DELIMITER_req:Common_Services

(* User_2 sends no User-specific service *)

- TC_R_2!TC_BEGIN_req: Common-Services

(* TC_BEGIN_req that contain an empty list of User specific services is sent to the peer *)

- TC_I_1!TC_BEGIN_ind: Common-Services

(* the corresponding TC_indication is received by the peer thanks to the synchronization that occurred on TC_R_2 with the SYNCHRO process *)

- i/exit

(* internal exit in process OPEN_SYNCHRO *)

- i/exit

(* internal exit in process SYNCHRO*)

- i/exit ({}:list)

(* internal exit in process DSM_BEGIN_2 that enables the WAIT_FOR_AC process *)

- TC_R_2!TC_INVOKE_req (SendEndSignal, Service_Id(MSCA1, SendEndSignal)): P_Services

(* the corresponding RSM sends a TC_INVOKE message to the peer *)

-i/exit (HandoverControl:AC)

(* internal exit in the DIALOGUE_INITIATION process which enables DIALOGUE_CONFIRMATION *)

- REQ_2!MAP_U_ABORT_req: Common-Services

(* the dialogue will be aborted...*)

- TC_R_2!TC_U_ABORT-req (user-Abort-PDU, User-Specific): P_Services

(* abortion message sent to the peer through TCAP *)

- RSM!Terminated

- PSM!Terminated

(* Indication sent to all instantiate RSM and PSM in order to inform them that the dialogue is over *)

- i/exit

(* internal exit in the SYNCHRO process *)

-DEADLOCK

We notice that this sequence is rather short because of an abortion coming from one of the 2 MAP users. The problem is that not many sequences will be very much longer...

CONCLUSION ON THIS WORK

Despite the numerous and unexpected difficulties that appeared as I went along in my work, preventing me from going as far as I wanted in the problem of Test Cases generation, we got some interesting results:

First, concerning the MAP and its specification, it proved to be very difficult to reach a low abstraction level, because of the diversity and the high number of the MAP User-specific services primitives which makes them very uneasy to introduce in a specification; unless it is a very long one; and it turned out that it was not possible to generate Tests sequences walking this kind of specification randomly.

As regards Test Cases generation, Random Walk proved to be not very efficient because of the random way it picks one action among the possible ones: this tool would be more efficient if there was a way to put a “weight” on actions, so that some of them would have a higher probability to be chosen than others; unfortunately this seems to be impossible - according to Antoine Bonavita who built Random Walk.

However, the sequences that were finally generated from the Dialogue State Machine specification on the particular case we considered were realistic and interesting sequences, even if they often were a bit short because of too frequents dialogue abortions. A good idea to go on with this subject would be to use the Test-Gen tool, and that would be easy, in so far as the specification was initially written in order to fit with CAESAR. I think it could be very interesting to compare Test Cases generated this way to the ones that were generated from the SDL specification of the DSM.