Implementation of UMTS Protocol Layers for the Radio Access Interface

Javier Colás, J.M. Pérez, Javier Poncela, and J.T. Entrambasaguas

Dpto. Ingeniería de Comunicaciones, ETSI Telecomunicación, Universidad de Málaga
javier@ic.uma.es

Abstract. SDL has been promoted by ITU as the design language for communication systems. The technology and tools have improved and SDL has been adopted by most manufacturers and integrated in their prototype and product lines. In this paper we describe an implementation of UMTS elements using this formal language. First, basic UMTS concepts are introduced. Next, we present some SDL limitations that have affected design decisions and the available solutions provided by SDL-2000. Afterwards, the high level design is shown, together with a description of the main aspects of the implementation and tests performed on the system. Finally, conclusions are presented.

1 Introduction

The SDL language has been promoted by ITU in the last decade as the most adequate language for the specification and development of telecommunication system protocols. Accepting this recommendation, the main manufacturers have followed this trend, slowly integrating this language in their development process. At the same time, several new characteristics have been included from the first specifications of the language. In particular, the updates made in 1992 and 2000 mean important milestones, first including object oriented characteristics in the language, and then extending these features with the experience obtained in these years.

UMTS (*Universal Mobile Telecommunications System*) represents the effort of mobile communication system manufacturers to offer new and more powerful services to users. Based on GSM protocols, and augmenting its capabilities, UMTS will provide a higher data rate, up to 2 Mbps, and a better use of spectral resources. The air interface consists of two elements: the User Equipment (UE) and the front end of the transport network (UTRAN). The quality of an implementation will be an important issue regarding its success.

Modelling the new communication systems, with their increasing complexity, requires new methods and tools. The use and advantages of object orientated formal languages in the development of these communication systems is a question that stands forward. On one hand, the new concepts require that team engineers learn of their advantages and utility; on the other hand, commercial tools need to support such features, a task that is often not easy to achieve. In this paper, an SDL implementation of UE and UTRAN entities is presented, focusing on the capabilities of the language

E. Sherratt (Ed.): SAM 2002, LNCS 2599, pp. 74–89, 2003.

and its object-oriented characteristics for the development of mobile communication systems.
The first section introduces the context of this work, the UMTS mobile system. Next section analyses some limitations found during the design process, highlighting the design decisions affected by such limitations and presenting solutions provided by SDL-2000. Afterwards, a high level view of the system and the design decisions that have been made are provided. Finally, some results and conclusions are presented.

2 UMTS

Nowadays we are accustomed to the concept of digital mobile networks. However, we should recall that first generation networks were analogue in nature. They were based on several similar but incompatible technologies. Aside from mobility, they only offered voice services. Second generation systems introduced the digital channel technology; as a result, a more efficient use of the spectrum was achieved using TDMA (*Time Division Multiple Access*)/CDMA (*Code Division Multiple Access*) technologies. Many new services were implemented including the provision for data services and improved security. The next step of this evolution, third generation systems, are based on the open interfaces of GSM *Global System for Mobile Communications*). The driving force is the desired ability to provide global mobility and compatibility, while at the same time making available an ever-expanding array of services that include paging, text/voice messaging, video and broadband ISDN capability.

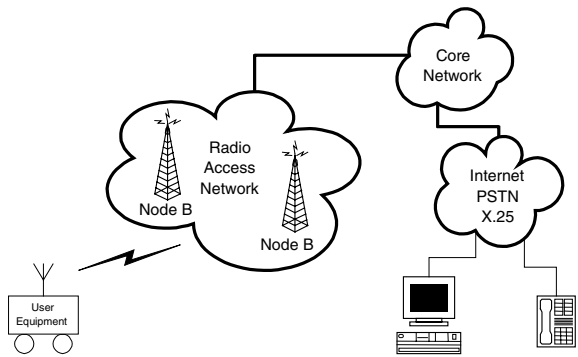

Fig. 1. UMTS architecture

The standardization work is done by 3GPP (*Third Generation Partnership Project*). 3GPP consists of multiple national standardization bodies and representatives of the telecommunication industry from Europe, Japan and USA. From the beginning it was clear that the 3GPP specifications would continuously be evolving; this was one of the major cornerstones on which the system design was based. Release 99 was functionally frozen at the beginning of 2000, and its biggest difference to GSM was the WCDMA (*Wideband Code Division Multiple Access*) radio access; new features and improvements, mainly on the core network, have been included in Release 4,

which was functionally frozen in March 2001. Other releases will appear in the near future.
UMTS (*Universal Mobile Telecommunications System*) is the name for the European, ETSI driven, 3G variant. It emphasises the interoperability and backward compatibility between 3G and GSM. The new radio access uses more efficiently the frequency spectrum. In synthesis, a 3G mobile communication system consists of three main components (see Figure 1):

- User Equipment (UE): is the device which is managed by the user and it is connected to the system via radio.
- Radio Access Network (RAN): provides the radio interface for the User Equipment, and controls and maintains its base stations (called Node B in UMTS).
- Core Network (CN): consists of multiple units, which transport data from source to destination.

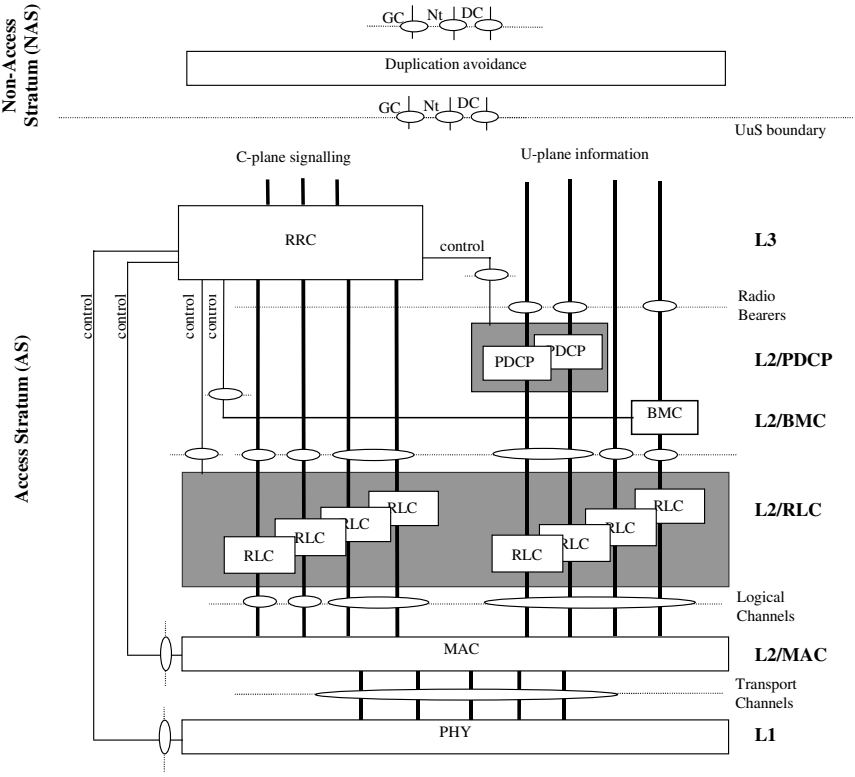

Fig. 2. General structure of UMTS

The architecture of UE and UTRAN (see Figure 2) is divided into *Non-Access Stratum* (NAS) and *Access Stratum* (AS). NAS layer contains the user applications and controls the *Access Stratum* functionality, which are the protocol layers that allow the interaction through the air interface (L3, L2 and L1).
The *Access Stratum* is subdivided in the following layers:

1. *Radio Resource Control* (RRC): It's responsible of managing all radio resources, from the establishment of connections up to the QoS. It consists of several entities that handle connections, broadcast data, paging, etc.
2. *Radio Link Control* (RLC): It offers three basic types of services: TM (*Transparent Mode*), UM (*Unacknowledged Mode*) and AM (*Acknowledged Mode*). Traffic modes are handled using the concept of radio bearer, which can be defined as a service with a given quality. In its interaction with MAC, RLC uses logical channels, while RRC interaction is performed on a transfer mode basis.
3. *Broadcast/Multicast Control* (BMC): Provides broadcast and multicast services.
4. *Packet Data Convergence Protocol* (PDCP): Manages the header compression for IP data streams, multiplexing user data into RLC radio bearers.
5. *Medium Access Control* (MAC): It is responsible for multiplexing/demultiplexing data among the different logical channels and the transport channels offered by the physical level. Functionally, its behaviour is divided in dedicated and common channels.
6. *Physical Layer* (PHY): Adapts the data to the physical medium. To achieve this, it performs several functions: synchronisation between UE and network, data modulation, control of orthogonalization codes, measurements in the air interface, etc.

3 SDL Limitations

This section describes the main constraints we have found in the use of an object-oriented approach, which have limited in some way our system implementation. Also, we present the solution that has been adopted and some new SDL-2000 [1] features that would have been useful for our design, if tools had supported them.

3.1 Interfaces

Usually both transmitter and receiver sides present the same structure; in other words, the same protocol layers exist in both sides, and they implement complementary parts of the protocol. Aside from its functionality, which is obviously different, the main difference between them, from a structural point of view, is the interface. In SDL-96 the interface is defined by means of gates. Signals defined in gates usually are the same, but have opposite directions at each entity. This constitutes a problem when using inheritance from a layer base type, because gates cannot be redefined in the inherited types, as it had been desirable in some cases (see Section 5.3).
If we use the latest SDL version, SDL-2000, there is a new feature that provides inheritance mechanism for the interfaces. This is a more suitable alternative for this issue. An interface is a type that contains the definition of a number of signals, remote variables and remote procedures and may be associated with channels, gates, connections or signal sets. These definitions are enclosed in the scope unit of the interface, so they will not be accessible from outside. Also, as this interface is a type, context parameters can be used. This feature could reduce the design time by increasing the code reuse.

3.2 Gate Used by a Signal

SDL allows to know the process identifier of the sender process, which is very useful in most cases. However, in other cases, such as when the sender process belongs to another block, which should be considered as a black box, or when the sender process sends the same signal through different gates, this identifier is not enough. These cases are not unusual in communication protocols, for example, situations where each protocol layer hides its implementation or the same signal is transmitted through different gates. An example is shown in Section 5.1, where the solution is inserting a new process for each gate so that the receiver process knows the gate used by means of the intermediate process identifier.

3.3 Redefinition Mechanism

SDL presents an important limitation in the redefinition mechanism: when using blocks inside another block, the designer is not allowed to use the redefinition mechanism at both block levels. For implementing a communication system between two entities, we can design the common behaviour of both entities as a base type, and specialise them via redefinitions. When the system complexity requires dividing a block into several, two redefinition alternatives can be used:

1. Using the redefinition mechanism at a high level. This means that a base type is defined, which groups the common characteristics, and we create new types which inherit from the base one. In this base type the system component blocks are defined as virtual, in such a way that blocks will be redefined when specialising the base type for adapting it to the particular characteristics. At the same time, all bidirectional channels are defined in the base type and the unidirectional ones will be defined in the subtype. This method has the advantage of clarity and results in an elegant code, as it is easiest to appreciate the differences between entities.
2. Using the redefinition mechanism at the internal block level. This alternative makes more flexible the implementation of the internal code. This flexibility consists in a quicker and easier way for adapting to standard changes, as these changes are usually modifications in the block behaviour instead of the addition of high level structural changes. No base type would exist for the complex block and no inheritance would be used for reusing high-level common structures.

In next Section we will refer to this problem when implementing the RRC layer.

3.4 Dynamic Block Creation

The concept of block is used to specify the static structure of a system. Blocks in SDL are used as containers of processes and other blocks. As blocks can be managed as objects, creating them dynamically would be a very interesting property. Unfortunately this is not possible in SDL-96, although this is one extension to the language that is incorporated in SDL-2000 [2]. In this new version, there are an extended and harmonised block/process concept (called agent), which covers all properties of SDL-96 blocks and processes, and a composite state concept with state partitions covering the properties of the SDL-96 service concept [3].

However, it is possible to model SDL systems where the perception of object (block) creation is achieved. This can be accomplished with process instances that exist at start-up time, whose sole purpose is to create other processes in that block when requested. This mechanism has been used in several places of our implementation An example can be seen in mobility management (see Sections 5.2 or 5.3).

3.5 Specialisation of Transitions

When specialising an SDL transition, it is possible to replace the whole transition. However, it is not feasible to include additional constructs keeping the already present statements. In [4] there is an attempt of overcome the mentioned constraints via a method called pattern approach. The intention of this approach is not to extend SDL, but to offer a description mechanism for SDL-based reuse artefacts. This solution is described by a syntactically incomplete SDL fragment together with accompanying rules that define how to reuse the pattern.
Replacing just some parts of one transition would be useful for reusing code and increasing the flexibility within the transition. SDL-2000 does not include this feature. However, despite it is not allowed to redefine only one part of a transition, it is possible to achieve a high quality code reutilization by dividing the tasks between transitions into several procedures. This way, the structure of the code is divided in smaller elements that can be called if required. This guideline has been followed in the design (see Section 5.1).

3.6 New Interesting SDL-2000 Features

The SDL-96 inheritance model for data was not consistent with other types in SDL. The new model brings SDL data more in line with the other object/type features of SDL (block type, process type, etc.). It makes data in SDL easier to understand for someone who is familiar with data in a (so-called) object oriented programming language such as C++ or Java. The SDL-2000 model introduces *object types* for data that can refer to *value types* and have polymorphic operators and methods. In our model, SDL data types inheritance has not been used, as most data types were defined in the 3GPP specification using ASN.1. However, internal data, which are not used directly in protocol primitives, is modelled as SDL data types. So, a data model like that can be useful to handle these internal data, for instance, encapsulating data and methods that access them.
Another interesting SDL-2000 feature is the possibility of writing textual algorithms within graphical diagrams. This is very useful because sometimes graphical diagrams turn out into a higher complexity and a lack of understandability of the algorithm [5].

4 Global Design

The two elements that communicate via the air interface, the UE and the UTRAN, have been implemented. The basic idea for the design has been to reutilize as much code as possible for both elements. Some studies have been published about the

benefits offered by object oriented approaches [6] versus classical ones. Studies such as [7] and [8] have shown that object-oriented approaches can reduce development time and the size of the resulting source code. At the same time, style rules for the development have been enumerated in [9].

We have roughly followed the phases indicated in [10] methodology: requirements analysis, system analysis, system design, implementation and testing. In this case, the requirements and system analysis phases recommended by SOMT are mostly performed by 3GPP [11], providing the standardized specification documents. These standards define the functional procedures over the air interface; the inner mechanisms, interfaces and structures are expected to be designed by the implementers.

The functionality of both UE and UTRAN is distributed along several layers, as it is shown in the previous UMTS section. Figure 3 depicts the designed architecture at the access network side (UTRAN). The block and channel structures are based on the structure suggested by the standards (see Figure 2). This block division allows the development of each block in parallel.

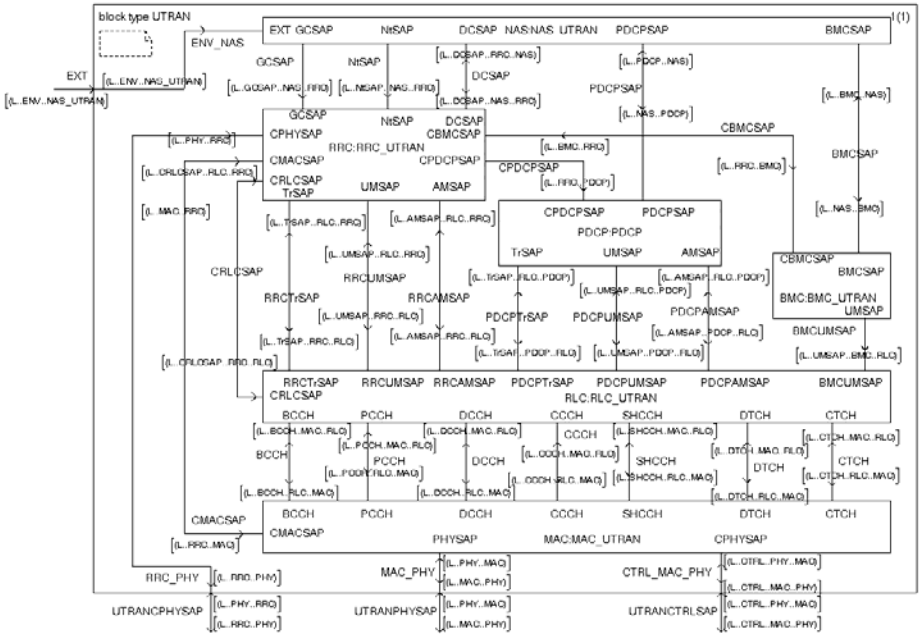

Fig. 3. High-level design for UTRAN

In the first stage the common functionality between UE and UTRAN has been identified. This subset has been implemented in block and process types; the UE and UTRAN systems have been built with blocks and processes derived from the base ones. Around 50% of MAC and RLC behaviour has been implemented in the base types; for RRC this percentage lowers down to a slim 10%. These figures are consistent with the expected differentiation between lower and upper layers of UE and UTRAN, as the management of radio resources is a more unidirectional function, while the data communication follows a more symmetric pattern.

In Figure 3, the interfaces between all UTRAN blocks can be seen. On the left side, three channels traverse vertically the architecture. They represent the interface used by RRC block to control the behaviour of the lower layers PHY, MAC and RLC. These interfaces will be used to establish and release sets of transport formats, as well as radio bearer services. The standard allows that this configuration happens quite frequently depending on parameters such as the radio conditions, the user bandwidth needs, etc. Between pairs of blocks, such as RRC-RLC or RLC-MAC, different channels have been modelled. For example, between RLC and MAC there is one SDL channel per type of logical channel. This decision has an impact on the design of those interfaces, as part of the multiplexing between those two layers is modelled with channels in the design, instead of having a software multiplexor. This solution has been chosen in order to simplify the burden of management (multiplex/demultiplex) of the signals inside each block, as usually each transport mode, logical or transport channel is handled in distinct ways.

The elements of the design have been organized in packages. These components have been arranged in a hierarchical structure, similarly to the way described in [12]. These definitions include data types, procedures and SDL entity types. In order to achieve this reutilization, definitions are located as globally in the hierarchy as needed and imported when necessary. Interface packages include definitions used by only two parts, allowing separated development of system components. The redefinition mechanism is used in our design to modify the layer behaviour according to the communication side. This problem has been addressed in the previous section. As it is impossible to use both of the alternatives presented, a decision must be made about which alternative better serves our purpose.

We have chosen to use the redefinition mechanism at the internal level, as we considered that its advantages are richer (flexibility for adapting independently both sides to changes). We think it is preferable to have a good capability to adapt new standard aspects than a more elegant design. Furthermore, communication protocols maintain many similarities along the evolution path of the systems (e.g. 2G, 2G+, 3G), which can be useful to reuse code by means of little modifications, and it is easier to reuse small components than large ones like entire blocks. The fact that specifications are continuously improving is not the only issue: a number of protocol versions exist with small differences due to regional technology and backward compatibility. So, this alternative offers a better approach to handle differences among specifications and among versions.

Data types used in the standardized interfaces are provided by 3GPP in ASN.1 notation. Since the use of ASN.1 in SDL code was standardized the support provided by commercial tools has improved significantly. Some deficiencies have been found in these modules; for example, the recursive call of types defined in two different modules. This cyclic reference is not properly handled, and a reorganisation of the module structure has been required to solve these problems. In SDL, packages have been used to organise these data type definitions. Other deficiency found in the conversion from ASN.1 to SDL is related to the `sequence of` construction. This structure is translated into `String` SDL type. If the type inside the sequence has no name, it is not possible to use operator `MkString()` to build one element of the sequence. To solve this problem, the type declaration of the element to be repeated must be provided in ASN.1.

5 Detailed Design

With the architecture shown in the previous section, the implementation of each layer still allows flexibility enough for the programmer. Work has been carried out independently for each of the layers. In this section, some issues of this implementation are briefly described.

5.1 MAC Layer

Figure 4 shows the base design for MAC layer [13]. As half of the functionality is common to UE and UTRAN, the object-oriented approach has quite simplified the design. In contrast with RLC and RRC layers, MAC layer design has followed the structure suggested in the specifications. This layer has been divided in several entities:

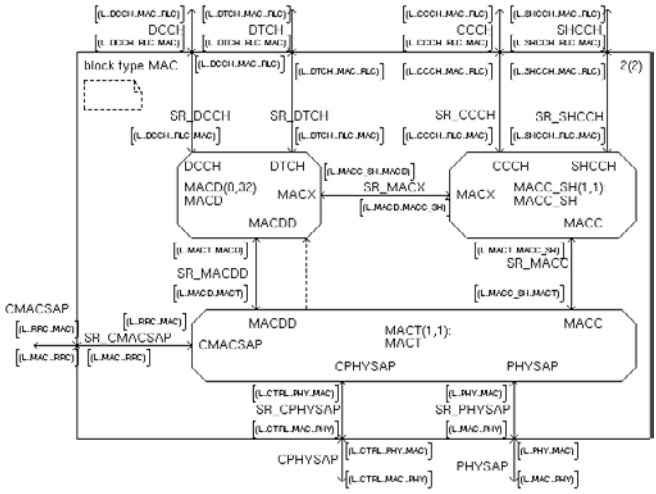

Fig. 4. MAC base type architecture

- MACC_SH: this process emulates entity MAC-c/sh, which controls access to logical common and shared channels. The base type contains the behaviour for channels CCCH and SHCCH, with identical processing in both sides, while channels CTCH and PCCH are included in the redefined types. It adds (transmission) and removes (reception) header parameters in PDU MAC-Data.
  The base type initialises most variables, empties reception variables and loads TCTF fields (indicate the logical channel type) with the correct values. It also allows reading the Buffer Occupancy for channel CCCH, as it is a common task for both sides. The necessary code for processing the headers for each channel is included when inheriting the behaviour in UE and UTRAN.
- MACD: implements entity MAC-d, which controls the dedicated logical channels DTCH and DCCH. It communicates with MAC-s/sh and MAC-control. Its behaviour is similar to MACC_SH process, but it handles the headers of the dedicated channels.

- MACT: contains the layer intelligence; it is also the process in charge of the communication with layers RRC, via control SAP, and physical. It also communicates with the processes that emulate the entities of the MAC layer.
  Its main functions are configuring MAC layer according to RRC queries, synchronising with physical and RLC layers (using timers and control signals), studying data flow measurements and transmitting them to RRC layer, and multiplexing/demultiplexing logical channels and transport channels. An example of the internal structure is shown in Figure 5, where a transition is redefined to modify the behaviour by means of procedure calls (see Section 3.5).

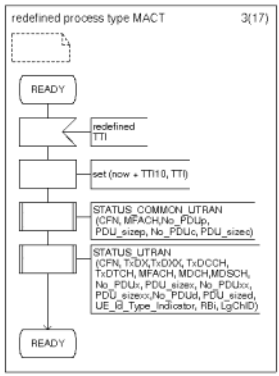

Fig. 5. MAC internal code

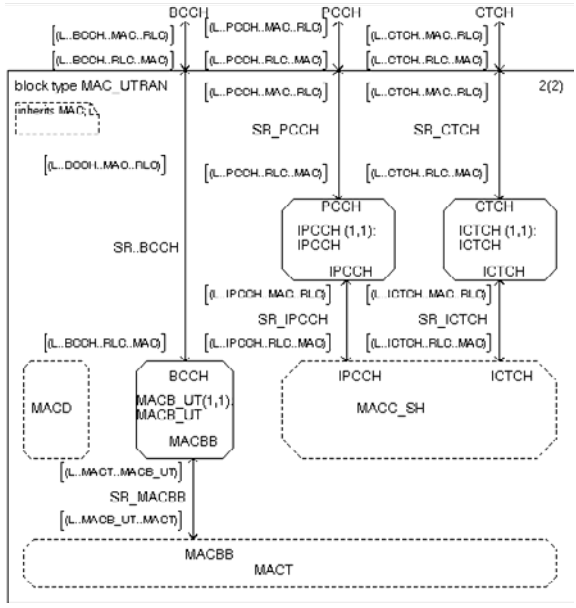

Fig. 6. MAC block type for UTRAN

The UE side redefines processes MACC_SH, MACD and MACT, and creates process MACB_UE which manages the broadcast channel in UE. This process will only listen to channel BCCH. After initialising variables with the broadcast channel values, this process will only wait for the data coming from MAC-control entity, delivering them to RLC via BCCH channel. Interfaces for PCCH, CTCH and BCCH are added in the inherited type.

Figure 6 shows the MAC block for UTRAN side. It redefines the same processes as in the UE side, and also creates a process for broadcast handling, the MACB_UT process, although in the reverse direction. In addition, two new processes named ICTCH and IPCCH must be created to distinguish the gate used by the signal (see Section 3.2). The sole function of these processes is forwarding received signals, allowing MACC_SH process to identify the sender process, which is different for each PCCH or CTCH channel. Also, the same unidirectional interfaces as in UE are created.

5.2 RLC Layer

The RLC layer [14] links RRC, PDCP and BMC with MAC. RLC uses logical channels as its interface with MAC and offers services in terms of transfer modes to the upper layers. The RLC layer structure has been designed following two basic principles:

- ✓ Reuse as much code as possible.
- ✓ Make the simplest code.

As the data flow is not symmetric, differences in this layer arise between the UE and UTRAN implementations. These discrepancies are basically the direction and transfer mode of the entities related with each of the channels. Given that for the dedicated logical channels any configuration is allowed, its structure is the same in both entities, however this is not the case for the common channels. As a result, the RLC base type includes those elements related to the dedicated logical channels, which are inherited in the derived types, RLC_UE and RLC_UTRAN.

In the base type, there is one process type for each transfer mode; at the same time, two dedicated logical channels, DCCH and DTCH, are also considered in this base type. These logical channels can carry information either in transparent, acknowledged or unacknowledged modes. Transparent and unacknowledged entities are unidirectional because no information sharing is required between uplink and downlink directions; thus, one entity implements the receiver and another implements the transmitter. The acknowledged entity is bidirectional. This means that five entities will exist per dedicated channel; as there are two logical channels, it turns out that ten entities will be used.

In the layer functionality, it should be noted that the services are created when an RRC connection is established; so, no active process will exist until a configuration request is received from RRC. The core of this layer is contained in the processes that implement the radio bearers, while the rest of the elements are in fact auxiliary, helping mainly in the task of managing the interfaces of the layer. The processes that are included in the RLC layer are the following:

1. **Bearer**: Responsible for managing the transmission and the reception. The processes are distinguished depending on the logical channel, the transfer mode and the channel direction. As each radio bearer has been modelled as a separate process, when several radio bearers are active they are executed concurrently.
2. **Controller**. It is responsible for receiving and handling the configuration primitives sent by RRC. Also, it must update the multiplexor routing tables. When the establishment of a new radio bearer is requested, the PId of the new process is obtained. This value is inserted in the active process database that this component holds. This database is used when routing signals to processes and is accessed by processes `Controller` and `Transmitting_MUX`.
3. **Transmitting_MUX**. The behaviour of this process is basically the one that can be found in a router; it forwards the upper layers service requests, such as insertion and removal of rows in the routing table. The synchronization with `Controller` is achieved via the `Update_Dedicated_Table` signal.
4. **Receiving_MUX**. This process routes the lower layer signals to the appropriate RLC entities. This is necessary as several radio bearers, each running as a separate process, might be multiplexed in the same logical channel. This is only accounted for in the dedicated logical channels.

Including the multiplexors as separate processes in the RLC block has been a design decision. If there were no multiplexors, signals should be broadcast to all processes either transparent, unacknowledged or acknowledged, and they would be responsible for discarding or accepting the signals depending on whether it was intended for them or not. This is quite inefficient as many signals should be generated and only one of them would be finally processed; the use of multiplexors avoids this multiplication of signals.

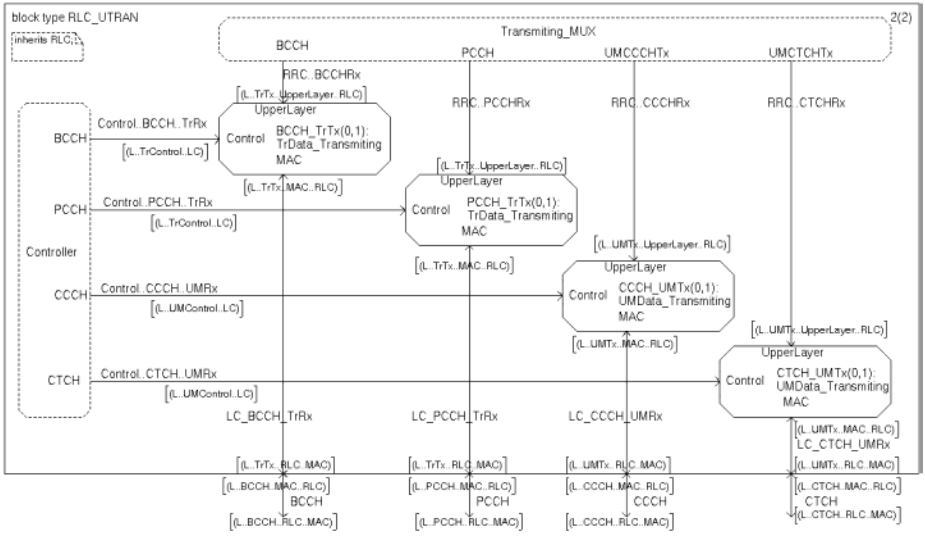

Fig. 7. RLC block type for UTRAN

The UTRAN RLC entity (Figure 7) is inherited redefining the process types that were declared virtual in the base class. The changed behaviour belongs to the areas of common logical channel management, broadcast and paging. Five logical channels are affected: BCCH, PCCH, SHCCH, CCCH and CTCH. As the data flow may be asymmetric, data services are separated into transmitter and receiver.
The main difference of UE implementation with block type RLC_UTRAN is the direction of the channels. For example, broadcast is a unidirectional downlink channel, and thus is transmitted by UTRAN and received by UE.

5.3 RRC Layer

Due to the complexity of the RRC layer [15], dividing the code among several blocks inside the layer is necessary. This involves certain limitations using SDL (see Section 3.3). This organisation has diverged from the guidelines suggested in the specification. The following entities have been modelled: Dedicated Control (DCFE), Mobility Management (MMFE), Broadcast Control (BCFE), Measurement (MFE) and Transfer Mode (TME). Each one has been implemented separately in different blocks inside RRC high level block.

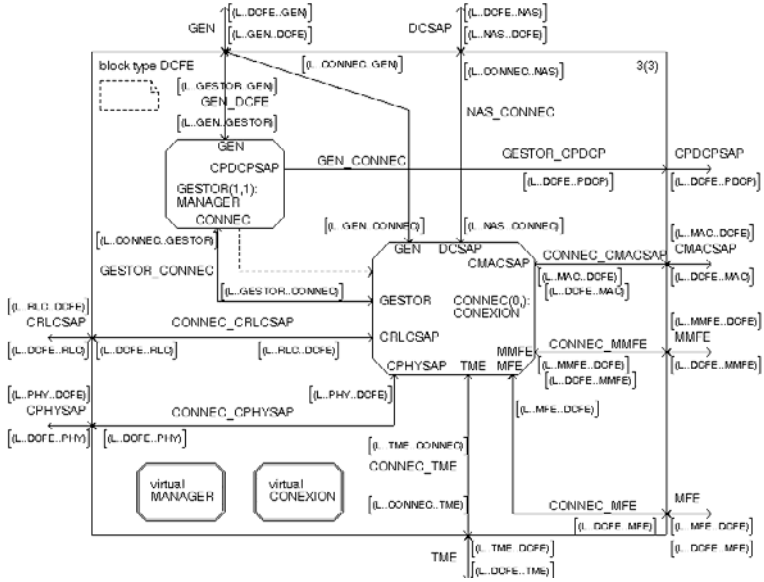

Fig. 8. Block structure for DCFE

DCFE block must manage the connection establishment and release as well as the configuration of resources provided by lower layers. In the base type, there is one controlling process type (`MANAGER`) that handles the creation of connections. Each connection runs as a separate process (`CONEXION`) since the establishment till its release. The model is shown in Figure 8. The corresponding entities for UE and UTRAN are inherited from this base type via redefinition. The `MANAGER` process type also holds a database with information about all established connections; in UE only one connection is allowed, while UTRAN can establish several of them simultaneously. However, the connection attributes, such as radio resources and configurations in use, are kept in the `CONEXION` process type. When upper layers request a new connection, a new instance of process type `CONEXION` is created and the connections database is updated in `MANAGER` process. A similar flow of actions is performed when releasing a connection: first, radio resources are released and afterwards the related `CONEXION` process instance is removed.

Block BCFE handles the broadcast information. In UTRAN, this information is received from NAS layer and sent to UEs in the appropriate frames; the scheduling is determined inside the block. This block exports part of this information so that other blocks in this layer can access it. This task is implemented in both sides because broadcast information is necessary to control the behaviour of the other blocks inside RRC layer. This information is also necessary in other layers, but is communicated via signals instead of exported variables, which cannot be shared among blocks at system level. As there is some common functionality, using inheritance was considered. However, signal directions are just opposite at each entity, so no gates could be included in the base type (see Section 3.1).

The specification states that *Unaligned PER* algorithm must be used for ASN.1 coding inside RRC layer. As RRC layer is very complex and consists of several

blocks, a design decision is where this function should be placed. Due to the nature of this functionality, which makes sense in RRC-RLC interface, it should be located as near to this frontier as possible. Block TME already exists inside RRC layer for routing messages from RLC to the appropriate RRC block. This could be a suitable solution, with the additional advantage that this coding mechanism will be transparent to all other RRC blocks that will handle the information in a completely abstract way.

6 Tests

Several types of tests have been performed along the development process. Besides internal behaviour and external interaction tests, which have been done for each layer, other tests carried out are:

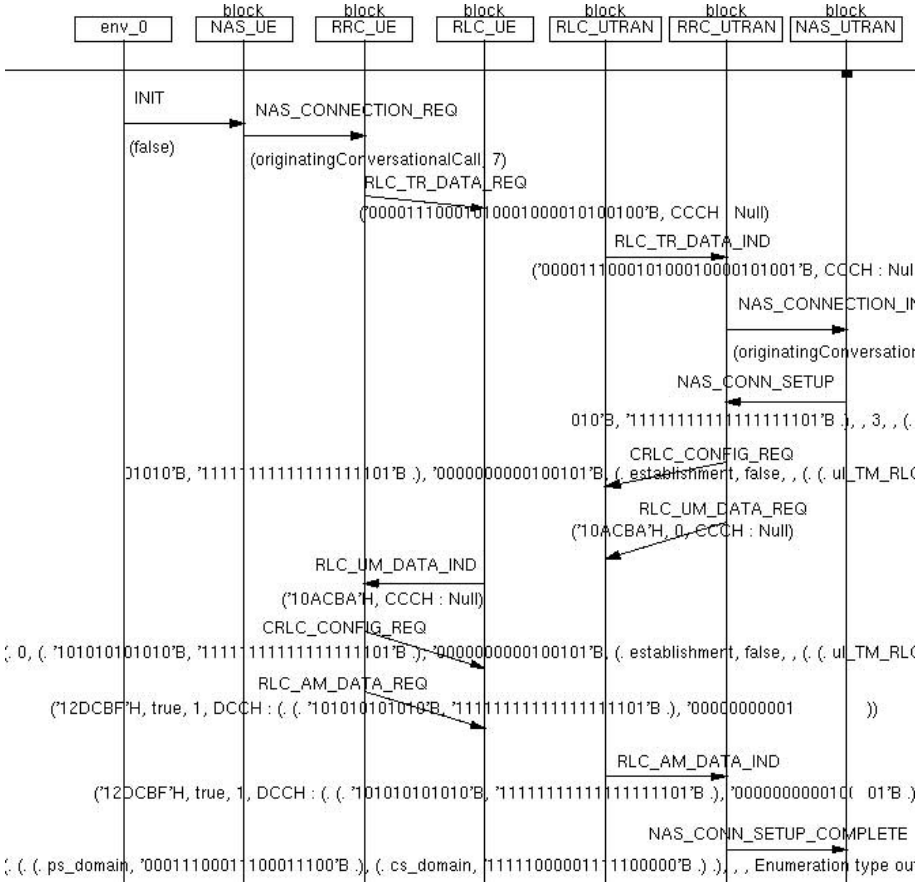

Fig. 9. MSC for connection establishment

1. Global tests: As an example, Figure 9 shows an MSC diagram with the messages that help to achieve the connection establishment. First, the UE requests the connection establishment by sending the INIT signal to NAS_UE. The adequate

primitives are interchanged (MAC and physical layers are not shown for simplicity), and the NAS_CONNECTION_IND primitive is originated in the UTRAN side. This causes UTRAN to send the NAS_CONN_SETUP primitive, which starts the configuration of lower layers. After processing the response, the User Equipment establishes one dedicated channel. When this channel is established, the User Equipment sends data via this channel in acknowledged mode (AM) to finish the establishment procedure.

2. Performance: The maximum transfer data rate has been measured between adjacent layers to obtain the performance of our implementation. Test configuration parameters are the packet size and the transmission time interval. The test architecture counts received packets at RLC layer, which are transmitted by the peer RLC. We have measured a transfer rate of almost 2 Mbps. Our measures show that, for packets of size 9920 bits, MAC layer takes around 2 ms to carry out all required processing. These results have been obtained on a Pentium III platform with Windows 2000.

7 Conclusions

This paper has shown an implementation of new mobile communication systems using SDL. The implementation has partly followed the structure division suggested by the specifications, but this scheme has been discarded in those places where more appropriate solutions could be found, mainly in the link control and higher layers.

The two entities at the air interface, User Equipment and Radio Access Network, have been modelled. The design has been carried out using an object-oriented approach that has allowed us to maximise the code reuse. It should be noted that the percentage of code that can be reused decreases as we proceed up in the protocol stack. The high level design has described how mechanisms such as inheritance and redefinition can be used in SDL models. Also, a feasible structure for each layer has been presented.

Some limitations have been found in the language. Some of them have been addressed in the latest version of SDL, although the standardized mechanisms are not included in the design tools yet. These new features are aimed to bridge the gap between SDL and traditional object-oriented languages; in particular, the interface feature is expected to be very useful to the communication system modelling.

Finally, some test results of the implemented system have been shown. These results reveal that, for systems without size or consumption constraints, the use of SDL is adequate as timing constraints are easily met.

Acknowledgements. This work has been partially supported by spanish Comisión Interministerial de Ciencia y Tecnología and European Commission under grant 1FD97-0650.

The authors would like to thank Leopoldo Alarcón, Isaac Blanco, Miguel Angel Cobalea, Alejandro Contreras and Alberto Peinado for their participation in this project.

Abbreviations

BCCH	Broadcast Control Channel
CTCH	Common Traffic Channel
DCCH	Dedicated Control Channel
DCFE	Dedicated Channel Functional Entity
DTCH	Dedicated Traffic Channel
MAC	Medium Access Control
MMFE	Mobility Management Functional Entity
PCCH	Paging Control Channel
RAN	Radio Access Network
RLC	Radio Link Control
RRC	Radio Resource Control
SHCCH	Shared Channel Control Channel
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
UTRAN	Universal Terrestrial Radio Access Network

References

1. ITU-T Recommendation Z.100, *Languages for Telecommunication Applications – Specification and Description Language*, Geneva, November 1999.
2. Sinnott, R.; Kolberg, M. *Creating Telecommunication services based on Object-Oriented Frameworks and SDL*, Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), pp 93–102, 1999.
3. Fischer, J.; Holz, E.; Moller-Pedersen, B. *Structural and behavioral decomposition in object oriented models*, Proceedings of the 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), pp 368–375, 2000.
4. Geppert, B. Rößler, F. *The SDL pattern approach – A reuse-driven SDL design methodology*, Computer Networks 35, pp 627–645.
5. Reed, R. *Notes on SDL-2000 for the new millennium*. Computer Networks 35, pp 709–720.
6. Booch, G. *Object-oriented analysis and design with applications*, Addison-Wesley Publishing Company, 1994.
7. Taylor, D. A. *Object-oriented information systems: planning and implementation*. New York, New York John Wiley and Sons. 1992.
8. Pinson, L. J; Wiener, R. S. *Applications of object-oriented programming*, Reading, Massachusetts: Addison-Wesley Publishing Company, 1990.
9. Derr, K. W. *Applying OMT*, Cambridge University Press, 1998
10. Telelogic Tau 4.0, *SOMT Methodology Guidelines*, February 2000.
11. 3rd Generation Partnership Project, http://www.3gpp.org.
12. Sipilä, J.; Luukkala, V. *An SDL Implementation Framework for Third Generation Mobile Communications System*, 10th International SDL Forum Copenhagen, Denmark, June 27–29, 2001, Proceedings
13. 3GPP TS 25.321, *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; MAC protocol specification*; Release 99.
14. 3GPP TS 25.322, *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; RLC protocol specification*; Release 99.
15. 3GPP TS 25.331, *3rd Generation Partnership Project; Technical Specification Group Radio Access Network; RRC protocol specification*; Release 99.