Testbed Development for Communication Systems Using Formal Languages

Javier Poncela González, Rafael Sánchez Mejías, Pablo Tapia Moreno,
Ricardo Ferrer Álvarez, José Tomás Entrambasaguas Muñoz
Dpto. de Ingeniería de Comunicaciones University of Málaga
Campus de Teatinos, s/n, 29071 Málaga (Spain)
Telf: +34-95-2132759, Fax: +34-95-2132027
E-mail: poncela@ic.uma.es

Abstract

Conformance testing plays an important role in the development of telecommunication equipments. In this paper we introduce a methodology covering the whole development cycle for efficient implementation, both in time and cost, of testbeds for communication systems, having as input one or more TTCN test suites. The formal language SDL is used to implement the lower layers in the Test System. Validation is achieved using an IUT emulator. A protocol conformance tester for the GAP profile of DECT has been built using this methodology.

Keywords: Conformance testing, methodology, testbed, SDL, TTCN, DECT.

1. INTRODUCTION

The telecommunications industry is a tremendously dynamic area. Usually, network operators are supplied by a few different manufacturers, while user terminals are built by a much bigger group. So, it is more likely that compatibility problems arise due to errors in the design or due to ambiguities in the specifications. Test laboratories provide a third party certification of products but, in order to perform this role, development of complex testbeds is required.

Some authors have described implementations of testbeds for different communication systems and protocols such as ATM [1], INAP [2] and ISUP [3]. However, the use of SDL is usually limited to the automatic development of Abstract Test Suites. In this paper is presented a methodology oriented towards the efficient development of testbeds for communication systems using formal languages. It is based on the integrated use of several tools and working environments for the languages SDL [4], MSC [5] and TTCN [6], whose simulation possibilities as well as validation and verification properties allow us to efficiently carry out the development.

The methodology uses the existing standards and recommendations in the area of communication systems testing ([6], [7], [8]) and describes the steps to follow from the input documents up to the complete implementation. Additionally, the methodology makes use of some generic components and tools that have been designed to enhance development efficiency. The testbed structure, as well as those components and tools, are described in sections 2 and 3, respectively. Section 4 explains the proposed methodology. Section 5 shows its application to the development of a DECT test system.

2. GENERIC TESTBED STRUCTURE

Figure 1 shows the generic testbed structure with three main parts: Man-Machine Interface (MMI), Lower Tester (LT) and Lower Subsystem (LS). Adaptor and IF-Env modules perform the connection between LT and LS. Besides, hardware implements the physical layer and part of the medium access control.

Both MMI and Adaptor do not depend on a specific system; they are described below. However, the Lower Tester and the Lower Subsystem must be implemented anew for every testbed. Lower Tester contains the Abstract Test Suite (ATS); its Test Suite Operations (TSOs) will be written in C. Lower Subsystem models in SDL the lower layers needed by the tester, including some functionality required by the Lower Tester TSOs. The IF-Env module is partially generic and is further described in 4.6.

Different languages are used depending on the component: Java, TTCN, SDL, C.

2.1. Man-Machine Interface

The MMI controls the execution of the Abstract Test Cases (ATCs) via the GCI standard ([9], [10]). This interface allows to determine the test cases available in the Lower Tester, the test groups, the number and name of the PCOs, and also provides the mechanisms to start the execution and collect traces.

The main characteristics of the MMI are the following:

- PICS and PIXIT parameters' editor.
- ATCs selection: individual or in groups.
- Event logging.
- Report generation.

All test executions performed on an IUT are archived in one database together with a timestamp. This allows test houses to automatically generate conformance test reports with reference to any time baseline.

Traces can be displayed in textual format or in MSC diagram format. Both modes allow the operator to filter all the log messages either by default levels or by selecting specific types to be shown.

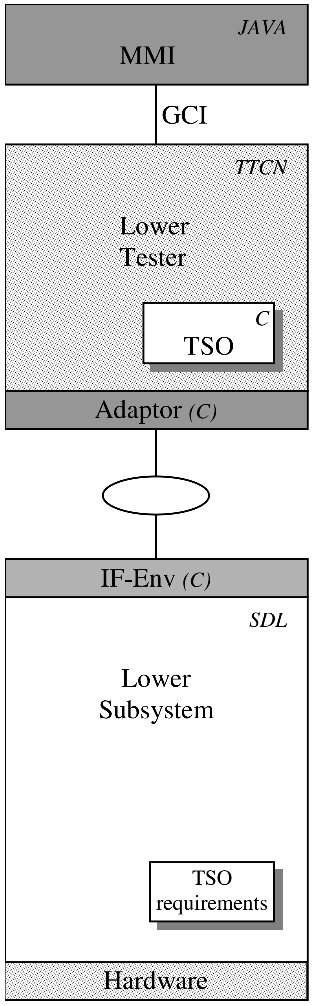

Fig. 1: General testbed structure.

The MMI can be implemented in a wide variety of languages. However, due to the fact that one of our requirements is the portability on different platforms, the language chosen has been Java using the Swing API. This API can provide the same appearance of the elements (buttons, scroll bars and so on) on every platform. Up to now, we have ported the application to Solaris 2.6/2.7, Linux RedHat 6.1 and Windows 95/98/NT. Examples of MMI windows are shown in Fig. 9 and Fig. 10.

2.2. TTCN platform adaptor

ATSs contain abstract descriptions of the conformance test cases. To obtain an Executable Test Suite (ETS) a platform dependent implementation of time management and input/output functions is needed. An adaptor including these functions has been built for Unix, Linux and Windows. Time accuracy may vary on different operating systems; this fact might pose some problems on test cases that require an accurate time measurement.

This component also contains functions to encode and decode the primitives exchanged between the Abstract Test Cases and the Lower Subsystem. The format used in this syntax transfer has been the one proposed in the template provided by Telelogic's Tau [19].

3. SUPPORTING TOOLS

Aside from those elements which are generic in the testbed architecture, we have developed a set of tools that will be integrated in the methodology in order to help in different development stages. These tools are grouped into:

- Naming rules.
- Automatic interface generators.

3.1. Naming rules

The naming rules describe how to assign names in an ordered and non-ambiguous form to every element which can appear in an SDL model, such as blocks, processes, data types, channels, primitives, etc. These rules are similar to the conventions used in several other programming languages.

Together with the naming rules, it is of use to keep an updated dictionary which includes the following components present in the SDL module:

- Channels
- Signals
- Messages

3.2. Automatic interface generators

Our methodology combines the use of modules written in SDL and TTCN. The interface to communicate these modules can be implemented either manually for each communication system testbed or automatically via generic applications. We have opted for the latter approach as it provides us with tools that reduce development time and the number of errors. Three automatic interface generators have been built: a TTCN to SDL data types extractor (*Datypex*), an octet-string encoder-decoder (*Oct2struct*) and an SDL environment interface module generator (*Envimod*).

a) *Datypex* tool: The interface between the upper layer modelled in SDL and the TTCN ATS is given by the latter. With this idea in mind, the upper interface of every block that connects to TTCN has been extracted from the standardized test suites. The tool takes as input the related .MP file and generates a set of files which contain the definition in SDL for data types, primitives and messages. Furthermore, the tool also generates the PICS/PIXIT parameters list; this will be used by the MMI to parameterize the test executions.

b) *Oct2struct* tool: In our SDL models the type definitions are built making use of the *struct* and *choice* constructions. However, at some point in the protocol stack these values must be translated into sequences of octets. This can be done in the lowest SDL layer or in upper layers, but the fact is that the IUT interface requires sequences of octets, not abstract representations. So, we implemented this tool which reads the type definitions included in one or more files and generates the appropriate encoder and decoder functions. The code generated is written in SDL.

c) *Envimod* tool: In the final stages of development, executables are generated for both TTCN and SDL modules. While during the earlier phases we can rely on the simulators' transfer syntax provided by Telelogic's Tau tool [19], in the end this interface must be implemented. This tool reuses parts developed for the *Oct2struct* tool, in particular, the SDL parsing module. *Envimod* automatically generates the IF-Env module (Fig. 1) using a transfer syntax defined for this purpose.

4. METHODOLOGY

The aim of the methodology presented further is to allow the implementation of testbeds for mobile communication systems using the Abstract Test Suites written in TTCN as the starting point. Initially, at our disposal will be the whole set of standardized ATSs as well as the specifications of the system, usually in natural language.

The methodology contains six stages that cover the whole development cycle. A brief overview is given here, while a more detailed explanation for each phase is included afterwards.

1. Definition: Specifies the functionality to be implemented in the testbed.
2. High-level design: Defines the high-level structure of the test system.
3. Lower Tester design: Checks and compiles the Abstract Test Suites. The TSOs are implemented in this step.
4. Lower Subsystem design: Creates the SDL models and verifies their behaviour.
5. Lower Subsystem integration: Integrates all the modules developed. This includes SDL modules, TTCN modules and hardware.
6. Construction and final tests: Builds and verifies the testbed.

4.1. Definition

The object of this step is to define the functionality to implement in the testbed. The beginning of development must be the gathering of all the documentation needed (Fig. 2). This documentation is composed of two groups: a) base and profile specifications; b) test specifications.

Test specifications include all documents that define the tests to perform on the System Under Test (SUT). These documents are: PICS & PIXIT questionnaires, Test Suite Structure and Test Purposes (TSS&TP), Abstract Test Methods (ATM) and Abstract Test Suites (ATS). The ATM specifies the testbed's architecture; usually there will be an ATM for each layer to test. In this step the ATM has to be selected according to the Abstract Test Cases chosen.

Related to the profiles, there are additional documents which, using the system test specifications as a base, comprise the issues to verify a device that implements such profile. These documents are the Profile Test Specification (PTS) and the Profile Specific

Test Specification (PSTS).

As an outcome, this first phase will provide a document including all tests that will be implemented in the testbed. If our target were designing a test system for some profile, this document would be equivalent to the PTS specification. Nevertheless, our methodology is valid for any list of test cases, although an arbitrary selection, as it doesn't represent complete subsets of functionality, could lack of usefulness. In Fig. 2 both options are depicted.

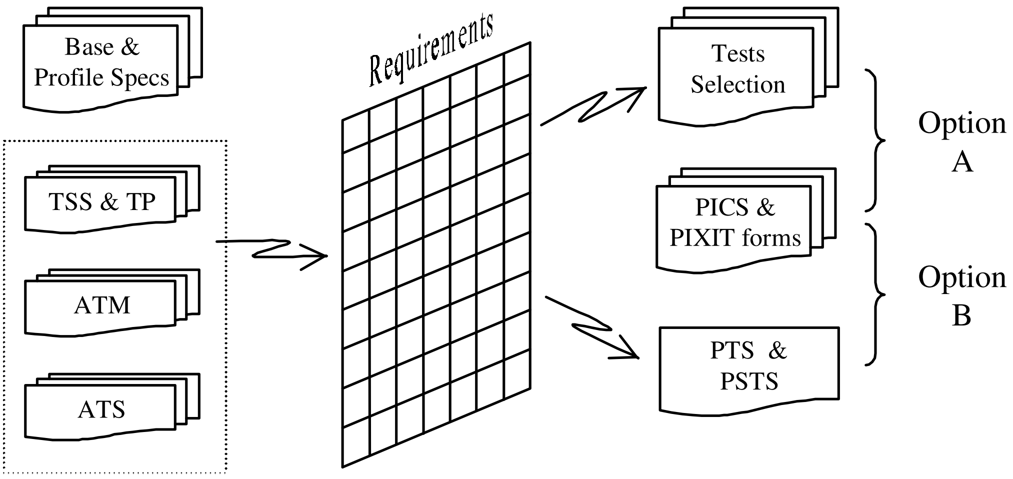

Fig. 2: Selection of testbed functionality based on requirements.

4.2. High-Level design

The second phase aims to define the high-level structure for the Test System. It will make use of the naming conventions described in 3.1.

Test System structure

The structure of the Test System is in a considerable degree given by the ATM used. When the purpose is to test an equipment, as in our case, the usual way is to choose the Remote ATM ([8], [11]). So, we can assume that the Test System will be composed of two parts: the Lower Tester (LT) and the Lower Subsystem (LS), which is the part of the service provider (ServPro) included in the Test System (Fig. 3). A third component, the Upper Tester (UT), present in other ATMs, will generally

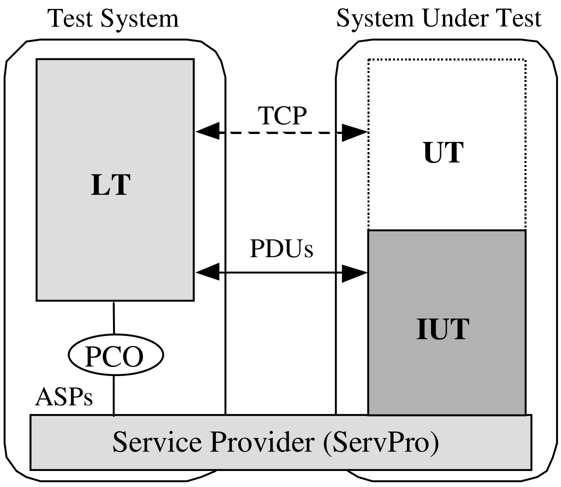

Fig. 3: Remote Abstract Test Method.

be a human operator that executes certain operations when requested by the LT; i.e., initiating a call from the IUT.

This structure must be rendered for each of the ATSs that will be included. That is, a communications equipment consists of several layers; for each of them it will be needed to design the corresponding architecture of the Test System.

The Lower Subsystem will be designed in SDL and its structure will be composed of an SDL block for each layer, excepting the Physical layer and those functions that are critical in time. Each of the SDL blocks will be connected to the lower layer through one or more channels. These channels are given by the PCOs defined in the corresponding LT.

Test plan

At this moment, the following tests to verify the Test System must be defined. There will be three groups of tests:

- Unit tests.
- Layer tests.
- System tests.

Although layers are built according to the standard, the tester usually requires additional functionalities. So, layer tests will comprise two parts: the ATSs standardized for the layer and extra tests that must be defined here in order to check the additional behaviour. The system tests will comprise the first set of layer tests, but they will also consider performance issues; these tests will be run in the last stage of the development cycle. Unit tests can not be defined until the detailed structure design is made (4.4).

4.3. Lower Tester design

This phase prepares the standardized TTCN Abstract Test Cases for their use in later stages. We can distinguish three substeps:

a) Syntactic and semantic checking: Although the standardized Abstract Test Suites are supposed to be error-free, the fact is that the earliest versions usually contain some mistakes.

b) Code generation: The TTCN code is compiled into C. This process is automatically done using any of the existing products in the market.

c) Test Suite Operations (TSOs): The TSOs are described either in natural language or in some pseudocode. For the ATCs to execute properly, the TSOs must be implemented. Those functions that request some operation from the Lower Subsystem will force changes in the SDL model.

The code obtained after this process will be linked to different libraries depending on the stage of development: simulation or application. The latter also needs to link with the generic adaptor module (2.2).

4.4. Lower Subsystem design

The Lower Subsystem will be mainly modelled in SDL. To this end, it is suitable to make use of any of the existing design methodologies for SDL, such as the one proposed by Cookson [12], Olsen [13] or SOMT [20]. This phase contains, for all the layers modelled in SDL, the following 3 substeps:

- Detailed structure design.
- Behaviour design.
- Layer tests

Detailed structure design

The structure defined in the second phase of the methodology will be detailed by defining processes and procedures as follows:

- Break down the blocks into processes, linking the channels already defined to the processes, and using routes between processes, indicating the signals that traverse each of these routes, together with their direction.
- Define procedures, including their functionality and the input and output parameters.

Behaviour design

The detailed design of each element will be carried out in this step. For this purpose, we will make use of the previous documents: architecture specification, blocks, processes and procedures. The needed behaviour will be implemented in these components.

To describe the most usual behaviours or the most complex ones, it is recommendable to employ MSC diagrams. Although these diagrams are somehow limited when representing all the possibilities that can happen during an exchange of messages, they provide us with enough information to clarify particular sequences.

The unit tests will be carried out making use of the simulation tools included in Tau. These tests involve subparts of the SDL models and provide some confidence on the correct implementation of the algorithms.

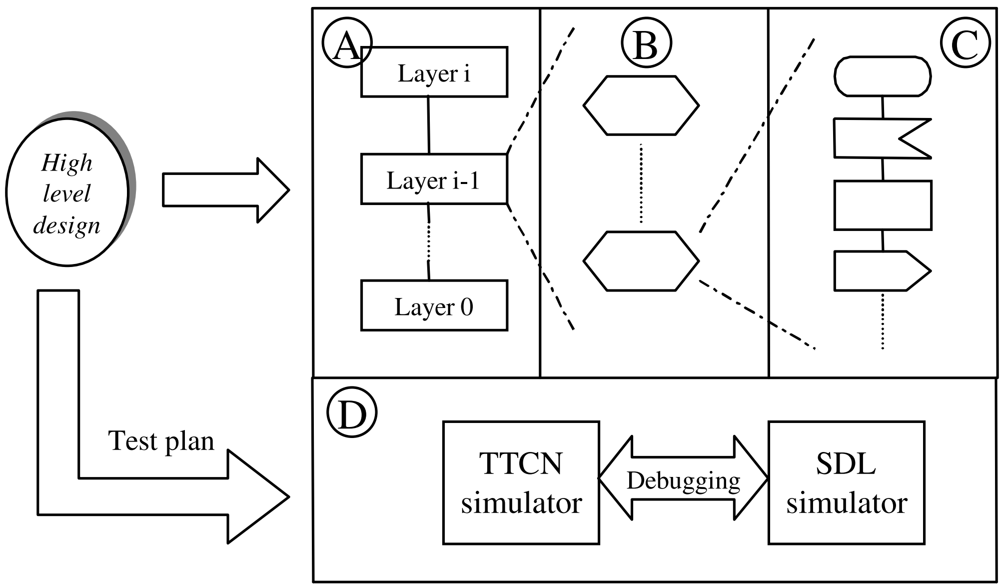

Fig. 4: Schematic overview of tasks performed in the Lower Subsystem design phase.

Layer tests

Layer tests are divided in two groups: standardized ATSs and extra tests.

The first group involves the construction of emulators for different service providers to carry out the tests defined in section 4.2 much more easily. They are built in SDL and integrated with the layers model. The behaviour of these emulators can be much simpler than the layers themselves. Fig. 4 depicts the case where the conformance ATSs are used to verify the SDL layer model. The steps are the following:

- ♦ TTCN simulator generation
- ♦ SDL simulator generation
- ♦ Debugging tasks

Simulators are constructed using the C code generators provided with Tau. At compilation time, the simulation kernel has to be chosen. The tool can connect one TTCN simulator with one SDL simulator.

Testing the extra functionality can be done in two ways: either describing the test in TTCN and using the above procedure; or making use of only the SDL simulator. This latter approach is easier to perform. The tests will then be described in scripts that can be automatically executed in the simulation tool.

The constraint of real-time execution can be eluded in the first approach, touching upon the behaviour more than on the performance. This issue will be taken into account in a later stage.

4.5. Lower Subsystem integration

In this phase, layers are put together incrementally, checking each new integration using the respective conformance layer tests. This can be done either using a top-down scheme, a bottom-up scheme or a combination of both. The former implies the use of service provider emulators. The latter approach requires, as a first step, integrating the hardware

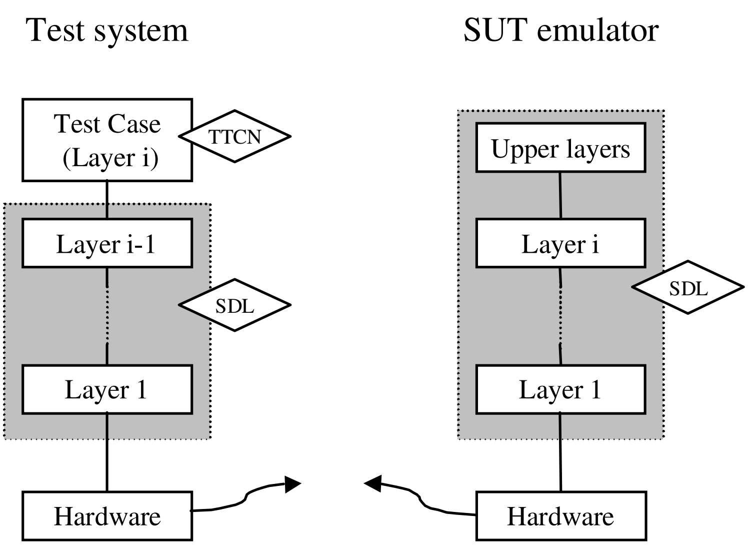

Fig. 5: Outcome of the Lower Subsystem integration phase.

with the lowest software layer. With Tau, this integration can be performed deeming the hardware as part of the environment of the SDL system or including C calls inside the SDL task symbols.

As a result of this phase, an integrated Test System is obtained, together with an emulator of the System Under Test that will be used to verify the tester behaviour. Their structures are depicted in Fig. 5. The Test System is composed of:

- a TTCN simulator which includes the Abstract Test Suites.
- an SDL simulator which represents the software part of the Lower Subsystem.
- the hardware implementing the Physical layer.

The structure of the System Under Test contains:

- the SDL system including all layers.
- the hardware implementing the Physical layer.

4.6. Construction and final tests

The Lower Tester (TTCN) and the Lower Subsystem (SDL) will execute as operating system processes. Three modules must be added (see Fig. 1): MMI, Platform Adaptor and IF-Env. The first two are generic components described in section 2; the latter must be generated.

The SDL to C code generator provides a template for the IF-Env module. The three main functions are: *init*, *receive* and *send*. The first one is a generic function that connects all the SDL-channels to and from the environment with their corresponding counterparts in the Lower Tester. The other two functions are generated automatically by *Envimod* [3.2].

Communication between the different processes is carried out via sockets. This is a portable mechanism that can be used in Unix, Linux and Windows. Using a control channel, the MMI automatically ties the PCOs of the Abstract Test Suite with the existing SDL-channels connected to the environment. The MMI also creates one internal channel, aside of the defined PCOs, that allows some of the operations requested by the TSOs to be performed.

Finally, the last validation tests will be performed on the testbed. Indeed, few errors should arise in this step, as nearly all the components, excepting the TTCN-SDL adapters, have already been tested.

5. APPLICATION EXAMPLE

The methodology shown in the above section has been applied to the development of a reduced testbed for the Generic Access Profile (GAP) of DECT (*Digital Enhanced Cordless Telecommunications*). In this section we will describe the main tasks performed to build such a testbed.

5.1. The DECT system

DECT is an ETSI standard for wireless private communications. The system contains fixed parts (FP) that communicate with mobile parts (PP). Each fixed part controls one cell. DECT supports uncoordinated system installation and includes the dynamic channel selection feature. In addition, provides rapid handover of calls in progress, both intracell and intercell. Up to 120 duplex connections can be simultaneously controlled by one FP.

DECT is specified in standard ETS 300-175 [14]. Parts 2 to 5 define, respectively, layers PHL, MAC, DLC and NWK. The Generic Access Profile (GAP), described in ETS 300-444 [15], identifies the minimum required functionality to convey voice at 3'1 KHz. It indicates how to establish, maintain and release a DECT connection with the appropriate access rights, irrespectively of whether the FP provides residential, business or public access services.

The protocol tests for DECT systems are described in ETS 300-497 [18], which includes the ATSs written in TTCN. For GAP, ETS 300-494 [17] describes the pertinent tests, referencing the general ones when required. Finally, ETS 300-474 [16] contains the PICS and PIXIT questionnaires.

5.2. DECT testbed implementation

First of all, from the available standards (5.1), we have defined the functionality that the testbed should cover. The conformance testing will include the NWK and DLC layers both for the Fixed Part (FP) and the Portable Part (PP). The number of test cases amounts to more than 100 for the NWK layer and around 30 for the DLC layer.

Then we proceeded on to the high-level design. The SDL system structure is defined using an SDL-block for each layer. Properly defining the interfaces between the different blocks is a key point for the success of the project. An appropriate definition reduces considerably the number of problems that can appear in the integration phase. These interfaces are forced by the TTCN Abstract Test Suites. Tool *Datypex* has been employed to build the signals and data types dictionaries. These dictionaries have undergone a continuous updating process during the development as new elements were defined.

At the same time, work on the tester design phase was started. The Lower Tester contains up to 35 Test Suite Operations, depending on the ATS. Using the Itex C code generator, the selected ATCs are automatically translated into C. The TSOs have been written in the same language.

Lower Subsystem design

The starting point is the high-level abstract specification, which clearly defines the blocks, the interfaces between them and the interface with the environment. Blocks for layers MAC, DLC and NWK, both for FP and PP, must be modelled. Here is a brief description of these blocks:

- Network layer (NWK): This layer is not required for the testbed, but it is built as part of the SUT emulator. It allows checking the NWK-TSOs, the IF-Env module and the additional behaviour included in the Lower Subsystem for the NWK Abstract Test Suite.
- Data Link Control layer (DLC): This layer is in charge of providing reliable links to transmit signalling information as well as carrying the broadcast messages. Tool *Oct2struct* (3.2) is used to automatically encode and decode the information contained in the PDUs and fill in the corresponding formal data structures employed in SDL. To verify DLC, emulators of the adjacent layers have been built. DLC is part of the NWK Test System. In addition, the ATM for DLC layer contains what the test specification calls the SubDLC block. This block is a subset of DLC and provides the ATS with functionality not described in the TTCN code, such as the segmentation and reassembly of fragments.

- Medium Access Control layer (MAC): This block has the responsibility to implement the Cluster Control Functions (CCF) described in ETS 300 175, part 3.

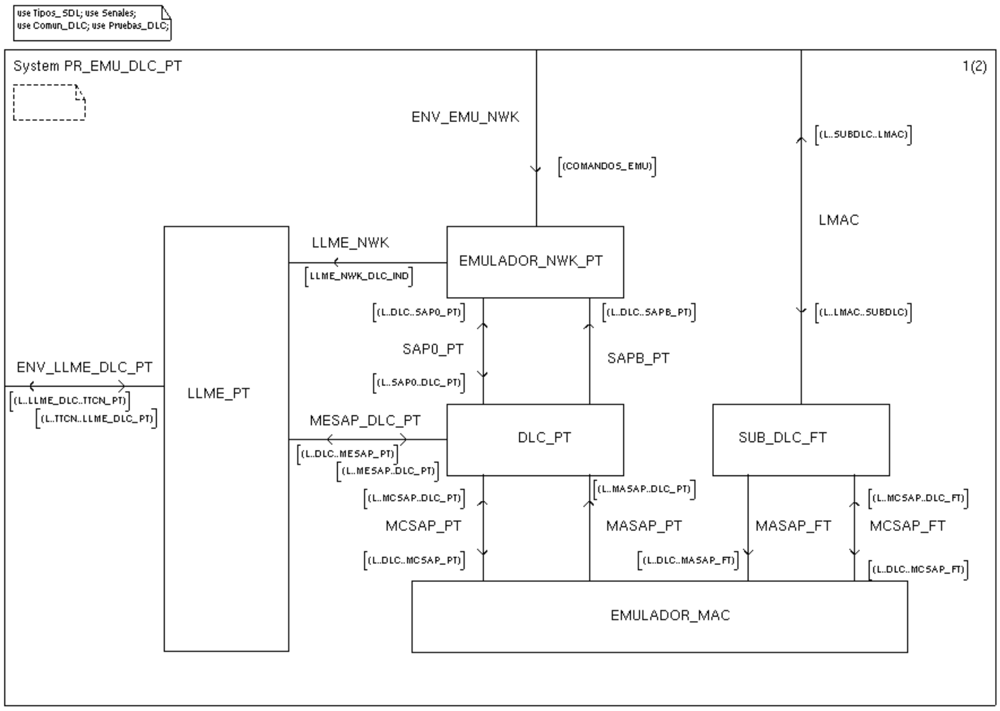

Fig. 6: SDL test architecture for layer DLC (PT).

We have used prototype boards from a DECT chip manufacturer to implement the radio access (Physical layer) and part of the MAC layer. These boards also process the user information (voice) transmitted.

The layer tests defined in the high-level design phase have been carried out on the DLC and NWK layers. In both cases we have implemented emulators of the corresponding service providers and some needed functionality of the upper layers. Fig. 6 shows the test architecture that was set up to check the DLC implementation used in a PP. In this case, channel `ENV_EMU_NWK` allows a limited control of the upper layer emulator, while channel `LMAC` provides the communication between the SDL system and the Abstract Test Suites.

We found out some discrepancies between the data structures used in our SDL models and the data types defined in the Abstract Test Suites. These discrepancies appeared mainly with the optional fields of some primitives. So, we had to change our structures, although the standard agreed with our model. One example is the `old_mcei` field in the `MAC_CON_IND` primitive. While this parameter is not included in the primitive [14], the ATS deemed it as always present.

Integration and Construction phases

The work on the integration phase can be carried out in two parallel tasks. The first task takes into account the hardware; the second task is related to the integration of the different SDL blocks.

For the former task there are two possibilities, as we explained in 4.5. We have preferred to include C calls into SDL task symbols as this method simplifies the module that provides the interface with the environment. One drawback present on this approach is that we must poll the radio boards driver to check whether a new packet has arrived. However, this polling does not degrade the performance of the system.

The SDL modules have been integrated in an incremental way. Depending on the specific ATM structure for each test suite, the Lower Subsystem requires different components. We chose to build four SDL systems, one for each test suite to be implemented: layers NWK and DLC, both for FP and PP.

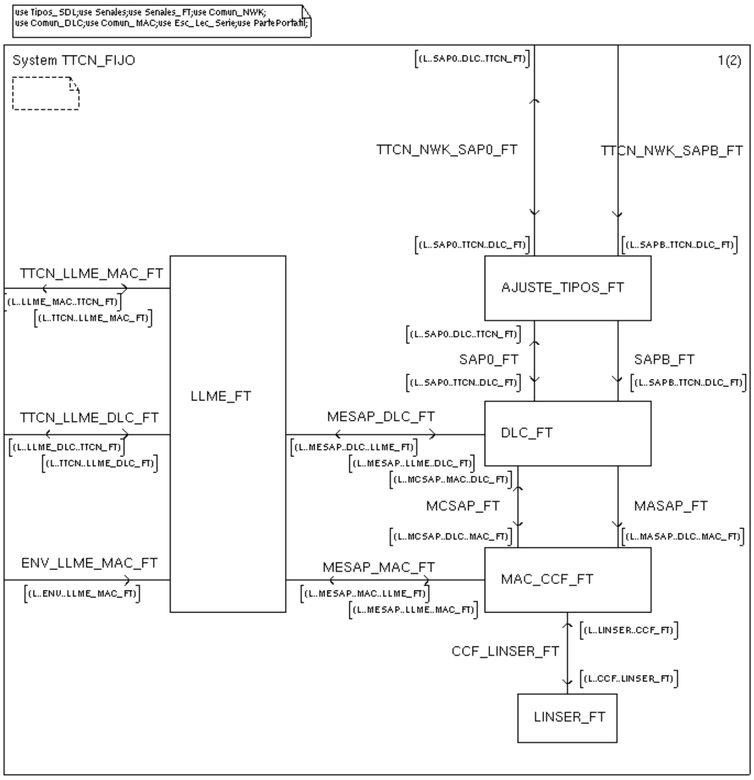

Fig. 7: Structure of SDL part for NWK (PT) layer tester.

As an example of the SDL systems designed, Fig. 7 shows the structure of the NWK layer conformance testbed when the IUT is the Portable Part. We have also implemented emulators of the possible IUTs, one for the PP (Fig. 8) and another for the FP.

During the construction phase, the IF-Env module was generated. All the other modules added are part of the generic testbed structure and need no changes for their integration. To setup the executable testbed we must provide the MMI with appropriate configurations which include the processes to connect for each architecture and the list of PICS/PIXIT parameters required. In Fig. 9 and Fig. 10 we can see the appearance of the user interface.

Performance of the system has been taken into account by modifying the libraries provided with Tau so that real-time is used. As a final step, out testbeds have been checked against a commercial DECT system.

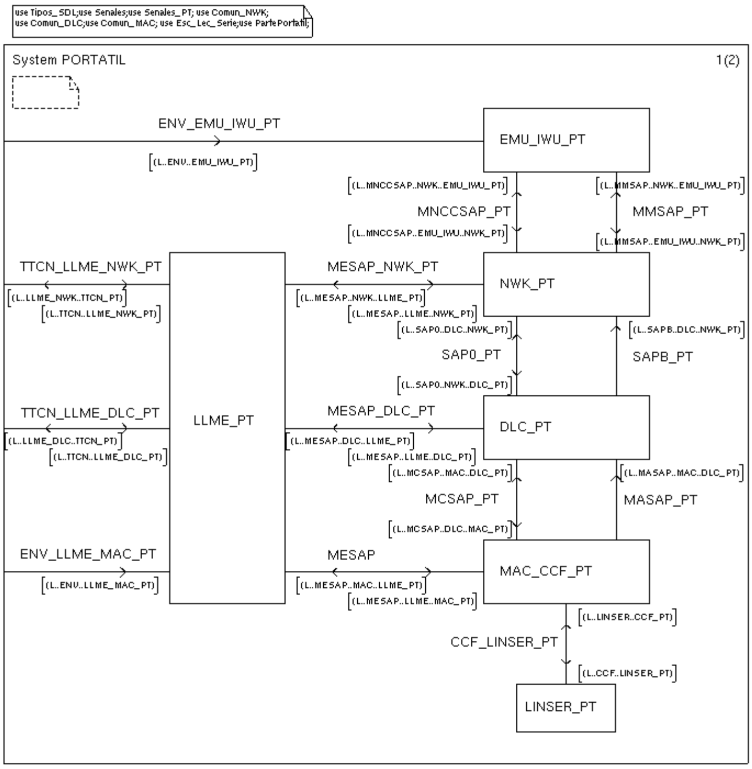

Fig. 8: Structure of SDL part for SUT emulator (PT)

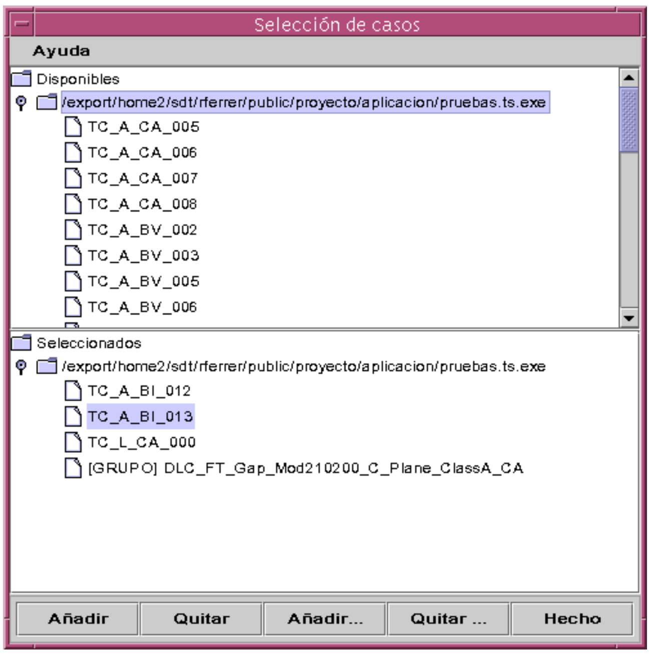

Fig. 9: Window for test case selection.

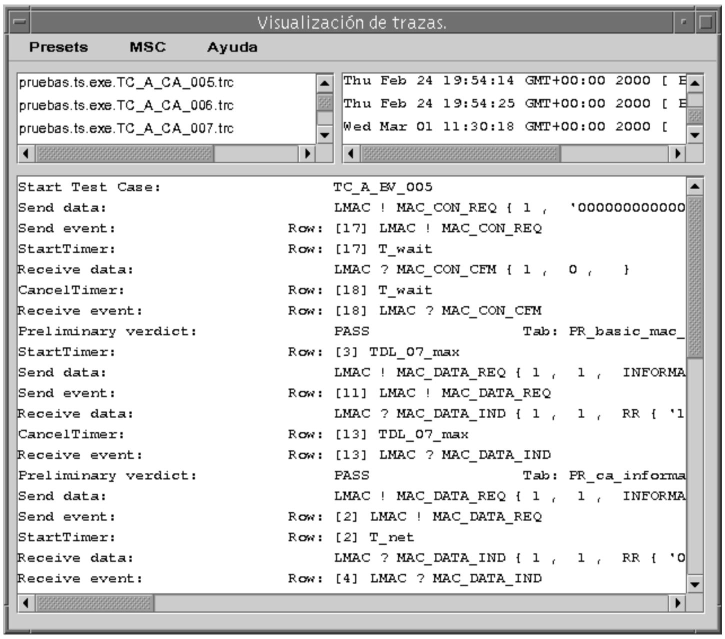

Fig. 10: Test case execution traces

Porting to other platforms

Our development platform has been Solaris, but the testbeds have also been ported to Linux. The main porting activities performed on each module included in Fig. 1 were the following:

- MMI: installing the JDK environment and the Swing API. The MMI only required some minor adjustments due to the different implementations of the Java run time environment.
- Lower Tester (TTCN): only needed rebuilding the C code to include the Linux libraries.
- Adaptor: when this module was written, we already took into account the portability issue; so, no changes were required.
- IF-Env: no changes needed.
- Lower Subsystem (SDL): some changes were needed in the C code to include the appropriate platform libraries. Also, the kernel libraries provided by Tau had to be compiled in Linux. This compilation was the main difficulty.
- Hardware driver: no changes needed.

Moving our development to Linux virtually offered no errors. Our measurements have shown that this platform provides quite good performance characteristics as a running environment. We are now porting to Windows, but the final checks on this platform are not ready yet.

Project schedule

The development teamwork has been composed of three people. Fig. 11 shows the main activities of the project. As we can see, the most time consuming task is the Lower Subsystem design. This is specific for every testbed, so this is the task where more resources must be introduced if we want to shorten the development.

We can also notice that the unit and layer tests require a big effort. This is partly due to the fact that this task is the most prone to discover errors in the SDL models. These bugs require changes in the designs and, because of this, the tests must be performed again. Verifying the correct behaviour of these modules is very important as it will dramatically reduce the effort required for later phases.

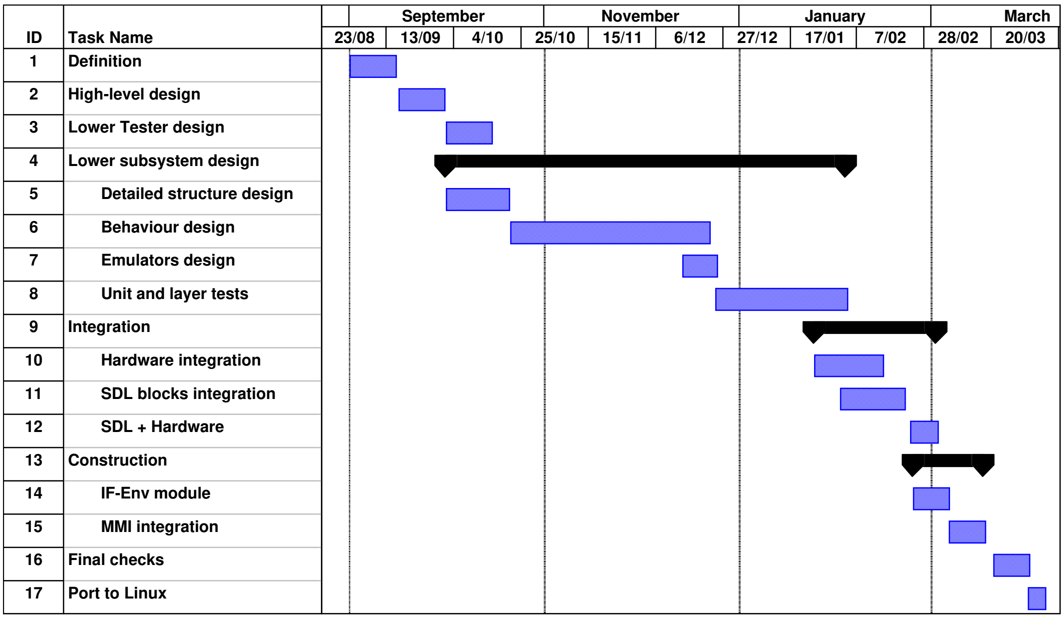

Fig. 11: Time scheduling.

6. CONCLUSIONS

We have presented a methodology useful for developing testbeds using formal languages. Use of SDL considerably improves development because of the capability, present in commercial tools, to integrate the development of SDL and TTCN modules. In this way, the modelled blocks can be simulated and the standardized test cases executed on them, allowing to verify models very early in the development cycle. Besides, SDL requires a smaller maintenance effort.

The methodology makes use of generic components, such as MMI and Platform Adaptor, that have been implemented. Also, several automatic interface generators (*Datypex*, *Oct2struct*, *Envimod*) have been developed. Together, these components and tools, simplify the construction of any specific testbed.

Usefulness of the methodology has been demonstrated by its application to a testbed for DECT wireless systems showing how small teamworks can carry out the development of this kind of projects in a short time. Quality of results was verified against already certified devices. Finally, the application has been easily ported from Solaris to Linux, used as running platform for commercial products.

Acknowledgements

This work has been partially supported by the Comisión Interministerial de Ciencia y Tecnología under Grant 1FD97-0650.

References

[1] S. Kang, Y. Seo, M. Hong, "*Development and application of ATM protocol conformance test system*", Journal of KISS(C), vol.5, Aug. 1999.

[2] S. Bae, H. Bae, S. Cho, "*INAP conformance test system development and verification using IUT simulator*", IEEE GLOBECOM, vol.6, Nov. 1998.

[3] J. Yuehui, C. Shiduan, "*The protocol conformance testing of CCSS7 ISUP*", High-Technology-Letters, vol. 4, Dec. 1998.

[4] ITU-T Recommendation Z.100, "*Specification and Description Language (SDL)*", 1993.

[5] ITU-T Recommendation Z.120, "*Message Sequence Charts (MSC)*", 1996.

[6] ISO/IEC 9646, "*Information Technology; Open Systems Interconnection; Conformance testing methodology and framework*", 1996.

[7] ETSI, ETS 300 406, "*Methods for Testing and Specification (MTS); Protocol and profile conformance testing specifications; Standardization methodology*", 1995.

[8] ETSI, ETR 021, "*Advanced Testing Methods (ATM); Tutorial on protocol conformance testing*", 1991.

[9] F. Brady, R.M. Barker, "*GCI Interface Specification*", v.2.2, NPL Inc. Dec. 1996.

[10] E. Desecures, L. Boullier, B. Pequignot, "*The INTOOL/CATG European Project: Development of an Industrial Tool in the Field of Computer Aided Test Generation*", IFIP TC6 10th International Workshop on Testing of Communicating Systems, vol. 10, Cheju Island, South Korea, Sep. 1997.

[11] B. Baumgarten, A. Giessler, "*OSI Conformance Testing Methodology and TTCN*", Elsevier Science B. V., 1994.

[12] M.D. Cookson, S.G. Woodsford, "*Design methodology using SDL*", BT Tech J. Vol. 11, No 4, pp.16-24, Oct. 1993.

[13] A. Olsen, O. Færgemand, B. Møller-Pedersen, R. Reed, J.R.W. Smith, "*Systems Engineering Using SDL-92*", North-Holland, 1994.

[14] ETSI, ETS 300 175, "*Radio Equipment and Systems (RES); Digital Enhanced Cordless Telecommunications (DECT); Common Interface (CI)*", 1996.

[15] ETSI, ETS 300 444, "*RES. DECT. Generic Access Profile (GAP)*", 1995.

[16] ETSI, ETS 300 474, "*RES. DECT. GAP. Profile requirement list and profile specific Implementation Conformance Statement (ICS) proforma*", 1996.

[17] ETSI, ETS 300 494, "*RES. DECT. GAP. Profile Test Specification (PTS)*", 1996.

[18] ETSI, ETS 300 497, "*RES. DECT. Common Interface (CI); Test Case Library (TCL)*", 1996.

[19] Telelogic Tau 3.4, Telelogic AB, Sweden, Aug. 1998.

[20] Telelogic SDT 3.4, "*Methodology Guidelines. Part 1: The SOMT Method.*", 1998.