EFFICIENT DESIGN OF SYSTEM TEST: A LAYERED ARCHITECTURE

Andrea BALDINI, Alfredo BENSO, Paolo PRINETTO

Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy
Phone +39-011-564-7007 – Fax +39-011-564-7099
Email: {baldini, benso, prinetto} @polito.it

Sergio MO, Andrea TADDEI

Magneti Marelli Electronic Systems
Research and Development Site
Viale Carlo Emanuele II n.118, I-10078, Venaria Reale (TO), Italy
Email: {Sergio.Mo, taddei} @venaria.marelli.it

Abstract

Starting from the idea of a general methodology to transform design specifications into system level functional test patterns for complex embedded systems, we propose a layered architecture as basis of such process. The architecture aims at strongly simplifying the test design, allowing the test engineer to concentrate on the high level parts of the system and wrapping all the complexity of the test environment.

The results are then verified on a complex case study of automotive applications.

1. INTRODUCTION

The core of our research is the correct definition of a methodology to obtain a generation of test scenarios, test cases and test patterns from high level and design descriptions. The large amount of information provided by the design of the system can be so entirely deployed, supposing that we can accomplish to lower the initial level of the representation of the behavior of the system into a description understandable in terms of black-box system level functional test. This research focuses on complex embedded systems such as complex and future top-of-the-line applications.

The research is backed by and done in collaboration with Magneti Marelli Electronic Systems of Venaria Reale, an international leader in automotive applications; such collaboration allows us to obtain a large amount of material, so that our case studies can be effective and industrially significant.

Since a methodology needs not only a process, but also a descriptive language, our choice has been from the beginning of our research [1] for the Unified Modeling Language (UML) [2][3], widely recognized and used in the industrial world.

UML is a modeling language, so it provides syntax and semantic to create models of the system at different levels. The basic idea of UML is to represent graphically all the aspects of the system, starting from broad and external views and including more details in a step by step process. The core of all the representations is so a set of diagrams for each level of abstraction.

A basic knowledge of UML is necessary to understand the ideas of this paper, and we emphasize the choice of such standard.

The basis of the presented architecture is an idea that is quite common in many fields of computer science, given the advantages in terms of complexity reduction and logical clarity: the use of a layered architecture. As a matter of fact, in many cases a good definition of layers can cope with the distance between different levels of description. Using actual case studies to refine the process, this part of the methodology is now mature, and complements and modifies the first intakes we had in this research [1].

0-7803-7542-4/02 $17.00

2. STATE-OF-THE-ART

UML is a standard of the Object Management Group (OMG) from 1997. Initially it was thought to describe software systems but the idea of modeling not only pieces of software, but entire and complex embedded systems in UML is quite recent but not new in literature [3][4][5][6]. Industrial tools (Rational / I-Logix) are also available to support the standard, and specific tools have been developed in academics, in particular for hardware support [7].

Since UML is only a notation, and does not include a process, it is not a complete methodology. General (RUP [2]) and specific (ROPES [3], BOOM [8]) methodologies have been so developed both for software and embedded real-time systems, and at least one (BOOM [8]) is specifically dedicated to behavioral description of the system.

These processes address partially test issues, but none has a real design-to-test approach; moreover, some other solutions are specifically addressed to different models, such as SDL (Autolink [9]) and pure statecharts models [10]. These last researches are alternative to our approach, even if they tend to lose the generality of notation of UML and are test-case oriented.

UML-and-test topics are so an open problem, and the market itself seems to feel the necessity of more refined models and methodologies both at theoretical and practical levels, in particular aiming at improving functional end-of-line tests, as individuated in a recent paper of our group [1], which represents the starting point of this entire research.

3. LAYERING

The starting point of the layered approach is the reliance to existing standards at very different levels. On the one hand we have high level description standards, such as UML, using which we can describe even the structure of the entire design; the interesting part from our perspective is the behavior of the system from an external point of view, which can be described in terms of sequences of messages. On the other hand we have low level signals as seen by test equipment, either a simulator, e.g. GSM, or a mechanical actuator to emulate the behavior of the user. We will now define the layers one by one, from the highest to the lowest level.

3.1. Design Layer

This is the interconnection level between the design and the test processes.

The basic idea is simple but effective: since we want to use all the information coming from the design of the system, we use the maximum flexibility for the high level layer; basically every sequence diagram made of high level messages is acceptable. In fact, sequence diagrams give a correct and easy view of the system's operative flow.

The structure of a sequence diagram at design layer is based on two basic actors (see Figure 1). The first one represents the real world environment and the second one a generic user. The system communicates with those actors through two interfaces: the human machine interface and the environment system interface.

This strong simplification does not influence the subsequent parts of the process, since it is only a particular point of view of the flow of execution of possible behavioral scenarios.

Timing is optional.

Design layer messages have the form:

<msg> ::= <Signature> <ParameterList> [<Timing>]

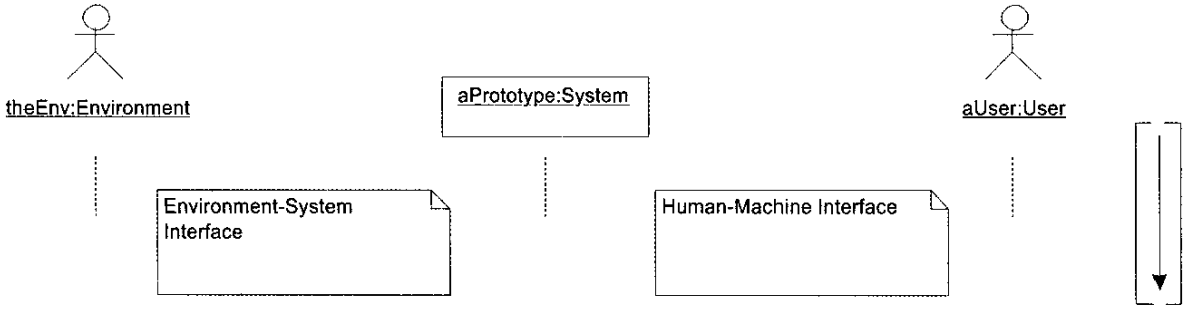

Figure 1: Sequence Diagram Structure – Design Layer

3.2. Test Layer

The test layer is the basis of the second step of the process, defining the first step as the translation of design information in design layer diagrams.

All the messages expressed at this layer are directed to single test equipments (either a high-level simulator or a low level driver) that interact with the system. A particular form of interaction with the system is the debug port, if provided. In this case it is possible to debug and animate the execution of the application software directly on the target system, being the real environment simulated by the test environment. When an application is under animation, it executes its normal flow but the state transitions and the exchanged messages are traced in real time through the test engine.

All design layer messages are translated in test layer messages, but each of them can be translated in different types of messages, in a polymorphic way. To represent this feature it is possible either to use the conditional branches in the sequence diagrams or to generate different distinct sequence diagrams, one for each message type. We can so cope with the typical complexity of a real system, which usually supports different ways to interact with the external world to obtain the same functionalities, e.g. input data as combination of keys, touch screen or vocal input.

The structure of the sequence diagram (see Figure 2) is focused on the test engine, since this is the only instance in the diagram and therefore the only part under direct control. Its function is twofold: it instructs the test equipment to simulate the wanted environment and verifies that the system reacts properly.

The term simulator in this context should be intended in a broader sense than usual. Test equipments currently used are high level simulators, which wrap all the functions of low level signals and provide higher level control interfaces. With this architecture, however, even low level drivers are viewed as simulators, provided that a high level description is available.

The timing is expressed in the standard UML notation and it is mandatory. The time can be either linear or sampled on an appropriate BTU (base time unit). If the test engineer is interested only in the chronology of the events a default time behavior can be used, i.e. typical time to accomplish a particular operation..

Test layer messages have the form:

<test_com> ::=
<TestInterface> <InterfaceParameterList>
<Signature> <ParameterList> <Timing>

The TestInterface is the interface to which the command is directed. It must contain a unique identifier of the simulator and its interface (port through which the test equipment is controlled). A little note is necessary for the parameters of the message. Valid parameters are for example couples of names/values representing constraints for the message and names/values used to evaluate the reply of the system itself, e.g. a message signaling an incoming GSM phone call can have also a parameter of timeout with the specific meaning: "If reply delay is more than 10 seconds the test is failed".

At Test Layer it is also necessary to properly configure the environment in which the system runs during the test process. This is done instructing the proper test equipment. In order to keep the messages always well catalogued and readable, the messages used to configure the test equipment that simulate the environment are named *env_com*, even if they are formally equivalent to *test_com*s.

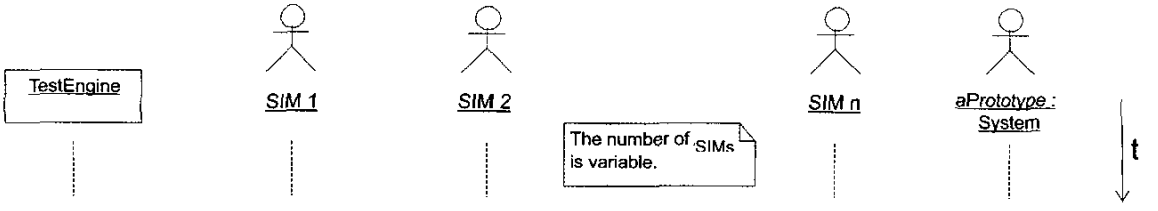

Figure 2: Sequence Diagram Structure – Test Layer

3.3. Layer 3

This is the layer that provides so-called user services. User services are related to standards (proprietary of public) supported by the system, e.g. RDS, GSM. Note that at layer 3 only the user-oriented aspects of the standard are visible, so strict timing aspects of a protocol are ignored, e.g. polling time of a GSM cell. The test engineer must describe in detail how these services must be specified in order to be compatible with the test-engine itself. This is a key point to support non-standard services, e.g. a proprietary bus. The concept of message is always present at layer 3, which is the lowest message level, since layer 2 and layer 1 are signal-oriented.

Layer 3 is made basically of messages and activity diagrams (equivalent to flow charts). The diagrams are needed to perform complex - algorithmic like – operations. This is useful to support two very different possibilities: if a high level simulator is present (standard simulator, e.g. GSM), layer 3 is simply the programming interface of the simulator; if we have low level drivers, the complex layer 3 messages must be mapped in layer 2 or 1 signals (see Figure 3).

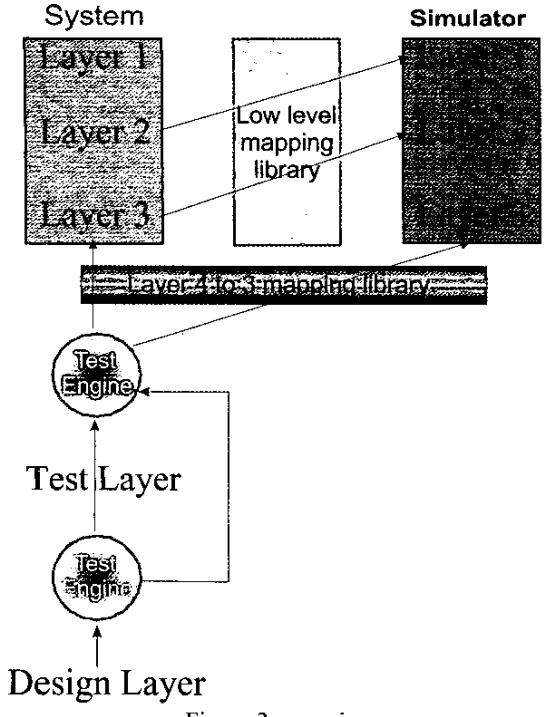

Figure 3: mapping

3.4. Layer 2

Layer 2 provides the so-called channel services; the basic concept is the port, i.e. a set of physical pin with cardinality equal or greater than one, linked by a logical relation.

Each port provides a set of services to the system and to the outside world. In particular the system can both take information about the outside environment and exchange data with other components (and with the test engine itself in the case of a debug port).

Note that in every system the interconnections from/to the outside of the system must follow a well defined protocol. Since usually communication protocols are layered, it is useful to understand that layer 2 corresponds to all the layers of a classic protocol except for the application layer (that is usually the higher abstraction level) and the physical layer, which corresponds to our layer 1

The applicable signal types at this level are, among the other standards, audio, video, buses.

Note that mechanical signals, i.e., instructions for a mechanical actuator, are also present including the protocol that drives the mechanical actuator that performs the wanted complex movements.

3.5. Layer 1

This is the physical layer. Two low level types of signal have been identified: mechanical and electrical.

Electrical signals are classified in terms of type of the electric signal (analogic or digital), typical electric values associated to the signal (voltages, currents, etc.) and the single pin to witch the signal is injected/detected to.

Mechanical signals are classified in terms of type of mechanical stimulus (single pressure or mono-versus rotation), point where the stimulus is applied.

The presence of a separate category representing mechanical signals is necessary to logically cope with the intrinsic complexity of actuators used during test, even if from another point of view each mechanical signal is strictly related to the electric signal able to activate the actuator.

However the case study has demonstrated that a separate approach is much more effective.

Obviously speaking of UML at this level has no real sense, so all the results are in terms of signals, and the sense of message is completely lost, wrapped by the higher levels.

4. LOW LEVEL DIAGRAMS

The choice of the diagrams to be used is of particular importance to conform to UML standards, so a few more words are necessary.

Layer 3 uses the activity diagrams instead of the sequence diagrams because an activity diagram can be used to describe the algorithm that performs the action wanted for a test layer's message. For example, if we decide to give the system an input string "TRY" after having chosen the keyboard as input device, at layer 3 it is possible to instruct the test engine to move a knob to position over and select each of the letters of T-R-Y.

Layer 2 and 1 do not need UML diagrams since the concept of message is no more present. In this case it is usually necessary to physically translate the algorithms at layer 3 into actual commands for low level drivers, e.g. mechanical actuators. This operation must be performed once. The lowest layers are entirely contained in the test engine.

5. TEST DESIGN INNOVATION

We want to emphasize that the major innovation of the proposed architecture is in terms of test design. As a matter of fact, the current industrial approach is low-level, starting directly from the command set of the simulators and manually building the test patterns.

Such approach becomes rapidly unfeasibly with the growing complexity of modern embedded systems.

Using a layered architecture, however, leads to the interesting results in test design, since the test engineer must not cope with the complexity of each single test equipment, either high level simulator or high level driver. In fact, high level simulators are formally at layer 3, whereas low level drivers are at layer 2 or 1, but they are accessed through a layer 3 interface.

Moreover the test designer actually can sketch the test scenario directly at design layer. The translation process between design layer message and test layer message and from test layer message to layer 3 messages is automatic, and customizable, so that a default working condition is created and custom test cases can be derived changing some message at a lower level, modifying a parameter or adding a new message.

The mapping process between layers is a complex argument, and it is out of the scope of this paper. However the claim is that such mapping can be made in an automated manner, relying on standard tables created once for all.

The industrial achievement from this point of view is quite high.

The most important fact to remember is the reliance of the entire architecture to two basic points: the UML standard on one side, and current and actual simulators on the market on the other. Also customized solutions are inserted in the same scheme quite easily, such as ad-hoc actuators and simulators.

The first consideration, the compliance with UML, is a very strong point, since other approaches are too low-level and UML is a de-facto standard in industry, even if it is used more frequently for design issues. However the same advantages it gives to software designers, in terms of clarity, simplicity and multi-level representation, are also valid for test designers.

The second consideration, the compliance with actual products on the market and the possibility of future extensions, is a basic need, but in this case the language each product uses is completely wrapped in the UML-based architecture.

6. CASE STUDY

Our collaboration with Magneti Marelli allowed us to obtain as a case study a top-of-the-line automotive application, i.e. a modern console which groups many controls of car-related devices. Such application has various orders of complexity: the interaction with the user is guaranteed by a set of input and output devices, such as a display, a keyboard, audio input (microphone) and output; moreover, a bunch of now common high-level-car components are integrated, such as positioning system (GPS) with navigation facilities, voice and data communications (GSM and WAP, or UMTS in the near future), vocal command, etc.

The system under test has two distinct but similar keyboards and there is a single robotic arm to handle them.

Such robotic arm is integrated into a complex test equipment, which is produced and used internally at Magneti Marelli, and it has actuation and observation capabilities. We present here a simplified version of the standard table with the main interfaces of the equipment.

Multipl.	*Description*	*InterfaceID*	*Layer*
22	System keys dedicated interfaces.	Key1-22	1
4	Knobs' movement interfaces. The interfaces are four because each one of the two knobs can be either rotated or pressed.	Knob1_Rotate Knob1_Press Knob2_Rotate Knob2_Press	1
1	System camera.	Camera	1
1	Image comparer. The algorithm that runs into the actuator able to compare two images.	IComp	3
1	Loud speaker with a line out plug.	Spkr	1
1	Microphone with a line in plug.	Mic	1
1	Sound comparer. The algorithm that runs into the actuator able to compare two sound waveforms.	SComp	3
1	Keyboard. This is a virtual interface and it is the wrapping of all the 22 keys present in the system.	Keyb	3
1	Test interface for the robotic arm of the actuator	Actuator.arm	4

Table 1: simplified standard table for actuator

We will now present a first example involving only the actuator.
We want to enter a number on the first of the two keyboards (see Table 2).
The actual mapping between one message to the lower layer corresponding messages is quite complex, and it is too burdensome to include here all the necessary tables.
It is sufficient to say that the translation is unique, provided the initial choice to resolve the polymorphism; in this case, the number is inserted using the keyboard and so the actuator.

LAYERING EXAMPLE OF NUMBER INPUT		
Layer	*Message*	*Notes*
Design	in_number("123")	The keyboard to be used to type the number is not specified. Therefore the message is polymorphic.
Test	actuator.arm.keyb0.in_number("123")	actuator.arm is the test interface to be used to inject the message. keyb0.in_number is the test message. The first keyboard now is indicated into the message and thus polymorphism has been removed.
III	actuator.arm.keyb[0][1].press actuator.arm.keyb[0][2].press actuator.arm.keyb[0][3].press	The keys 1, 2 and 3 are pressed on the first keyboard.
II	K0[1].press K0[2].press K0[3].press	The actuator is no more visible. At this layer the system ports are directly visible. In this case K0 is the port for the first keyboard of the system.
I	pression(key_12) pression(key_13) pression(key_14)	The physical pin is identified. In this case the key 1-2-3 of the first keyboard.

Table 2: layering example (UML diagram converted into explicit message)

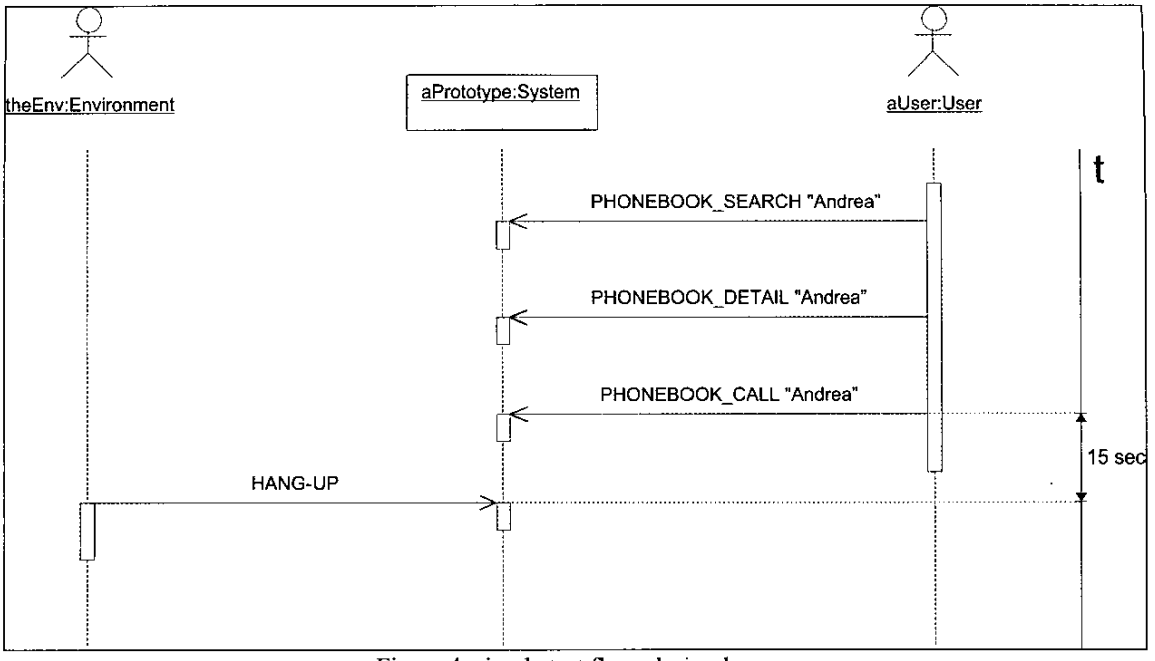

Figure 4: simple test flow: design layer

The second example (see Figure 4) is more complex, since it involves four high level messages. In this case the aim is quantitative, we want in other words to evaluate a figure of a typical ratio between the number of high level messages and the number of low level signals.

The test case consists of an automatic search in a phonebook and a subsequent call, hanged up by the remote user. The aim is testing the internal GSM phone of the device and all the related interface and algorithmic features. The test environment is made of a test equipment (A) with a robotic arm for mechanical actuation (A1) and a camera for screenshots observation (A2), and a GSM simulator of level 3 (B) (model Wawetek4200S); we simply omit layer 2 intermediate messages and present the data for the two most significant messages, PHONEBOOK_SEARCH and PHONEBOOK_CALL.

# Msgs/signal	PHONEBOOK_SEARCH		PHONEBOOK_CALL	
Design layer	1		1	
Test layer A	1	Total:	1	Total:
Test layer B	1	2	10	11
L3 A (A1+A2)	3	Total:	2	Total:
L3 B	0	3	10	12
L1 A.A1	59	Total:	1	Total:
L1 A.A2	51	110	1	2

Table 3: accumulated results

In this case is significant to present the results in terms of diagrams, to understand the overall complexity of the problem. From the design layer diagram, the test engine maps all the messages into a test layer diagram, so a sequence of layer 4 messages (see Figure 5, next page).

A complete analysis of the mapping is again too burdensome, but it is evident that the automatic insertion of the environment simulation and of the observation messages is a key point in the simplification of test design. In fact, this scheme can be modified to create derived test cases, but can also be used as it is.

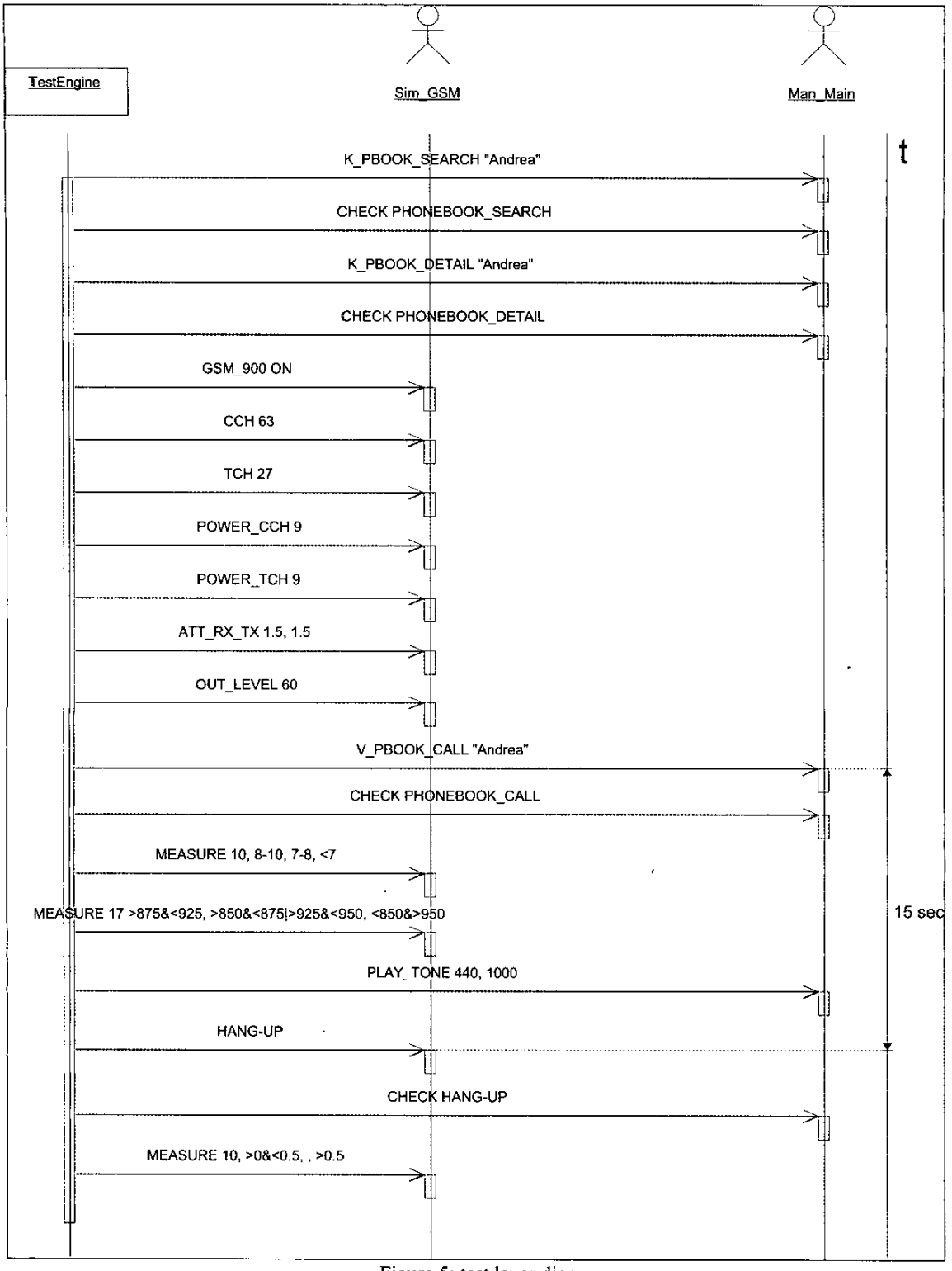

Figure 5: test layer diagram

As presented in the table with the accumulate results (Table 3), from the test layer diagram a layer 3 diagram is extracted. If the test designer modifies the test layer diagram at some extent, such modifications reflects on lower layers.
The layer 3 diagram is similar to the test layer, since this is basically a wrapping layer for the specific test environment. We present here only the last messages of the diagram (Figure 6).
We omit layer 2 and 1, not significant to present from a qualitative point of view, provided that in that case we have already provided significant quantitative data for specific commands (Table 3)..

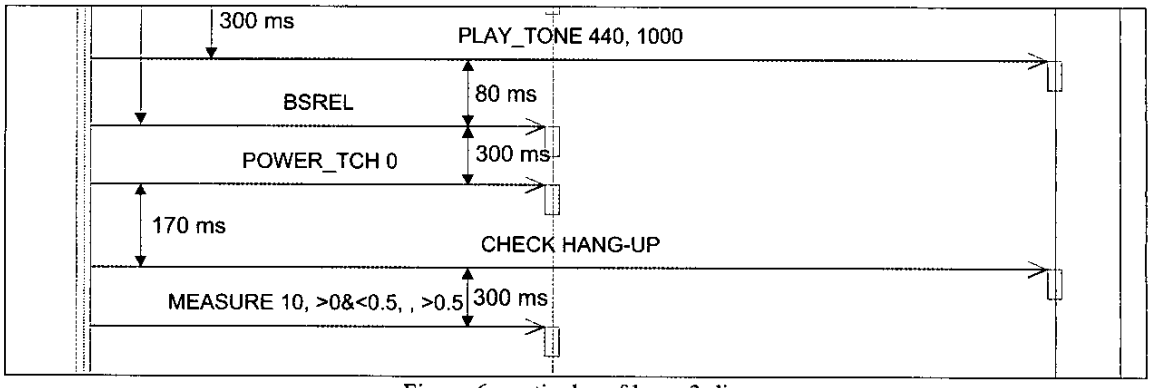

Figure 6: particular of layer 3 diagram

7. RESEARCH DIRECTIONS

This research extends the insights obtained in collaboration with the industry [1] but many aspects remain interesting and worth exploring. Orthogonal researches are concluded or in progress on the features of a test engine to support the design-to-test process and on the formal definition of the process itself; those two arguments are in some measure independent from the architecture presented in this paper, since such layered architecture is an evolution of out first approach. On the other side, the layers clearly fit into the general structure of the process.
The process itself is under development and it is changing from the first researches [1], provided that our current interest is in covering the gap between the design level and the test commands, and layers are very effective in this direction.
However, viewing the process from a general perspective, we can individuate two big phases: a test-case generation at a high level and the translation of such test cases into low level commands. Our current researches on the test engine and on the translation process address in fact the second phase.
The definition of the coverage for the test relies, on the other hand, on the first phase, since in system level testing we are interested in the functionalities of the system from a behavioral point of view, so the test-case generation is the main factor.
As a consequence, our research is now focusing heavily on the first phase, using many concepts inherited from software engineering and software testing. As a matter of fact, the idea of generating test cases from a high-level model [1] is quite accepted in modern researches, and different methods are available to generate test cases, such Use Case Based Testing [11], or particular criteria for statechart models [12] and collaboration diagrams [13]. Such criteria contain also a metric to evaluate the coverage of generated test cases. Our research will address the problem of evaluating the effectiveness of such metrics when applied in conjunction with our translation process, so that the coverage figure is meaningful also at the lowest level.
On the other hand a research work is also necessary to integrate our test engine with such methods, so that the test designer can effectively work directly with the high level model, customizing the low levels only when necessary.
This last accomplishment would give our process a tremendous advantage with respect to other techniques.
Last evolution is the application of such concepts to entire families of systems, to reuse as much as possible the testing of the core functionalities and the test engine programming; such evolution relies on domain analysis.

8. INDUSTRIAL EXPERIENCE AND LIMITATIONS OF THE APPROACH

From the industrial perspective the presented research was a partial success. It was a success because we actually manage to cope with the big

abstraction gap between the two levels, design description and test commands. It was partial because the case study is a real world example but the test command generation has been performed only for a subset of all the functionalities of the system.
Moreover the approach itself has some limitations and the research needs must address the big problem of evaluating the effectiveness of the generated functional system level test.
The basic idea of taking message sequence diagrams, a small part of UML, from specifications and translating them into test sequences (including timing considerations) can be criticized: such charts alone should not be regarded as an adequate specification of a system.
In other words one can have no confidence that they are complete or consistent. The test cases will just be those that the designers considered, which are the ones the system is most likely to execute correctly.
It is necessary to provide criteria and methods to address such problems, but on the other hand this approach is able to accept any sequence diagram as input, both coming from the actual design and specifically designed and created for test purposes. It is up to the test designer to apply other techniques, such as formal methods, or specific generation methods for test cases starting from the complete model. In other words many diagrams and design parts, such as state-charts, can be analyzed and used to generate message sequences that represent additional test cases and can be successfully used in our approach. Apart from these considerations, the test coverage evaluation remains an open problem, and a difficult one.

9. CONCLUSIONS

The use of layered architectures is quite common in many fields of computer science. Also in this research a good definition of layers has proven the key point for a correct solution to a simplification of test design, following the general idea of a well defined designed-to-test process. Case studies demonstrate the power of our approach, indicating a good general structure and a high ratio low-to-high level.
Open problems and directions of the research have been also indicated, inserting this research in a more complex general framework.
Industrial importance of the approach, accomplishments and limitations have been analyzed.

10. REFERENCES

[1] Baldini, A.; Benso, A.; Mo, S.; Taddei, A.; Prinetto, P. – *Towards a Unified Test Process: from UML to End-of-Line Functional Test* – IEEE International Test Conference 2001 (ITC'01) - Proceedings - page(s) 600-608 - Oct. 2001

[2] Fowler, M.; Scott, K. – *UML Distilled* – Addison Wesley Pub. – Sep. 1999

[3] Douglass, B.P. – *Real Time UML* – Addison Wesley Pub. – Oct. 1999

[4] Selic., B. – *Using UML for Modeling Complex Real-Time Systems* – Rational Inc. White Paper – http://www.rational.com/products/whitepapers/

[5] Fernandes, J.M.; Machado, R.J.; Santos, H.D. – *Modeling industrial embedded systems with UML* – Hardware/Software Codesign, 2000. CODES 2000. Proceedings of the Eighth International Workshop on – page(s): 18 – 22 - May 2000

[6] Selic, B. – *Using the object paradigm for distributed real-time systems* – Object-Oriented Real-time Distributed Computing, 1998. (ISORC 98) Proceedings. 1998 First IEEE International Symposium on – page(s): 478 – 480 - April 1998

[7] Sinha, V.; Doucet, F.; Siska, C.; Gupta, R.; Liao, S.; Ghosh, A. – *YAML: a tool for hardware design visualization and capture* – System Synthesis, 2000. Proceedings. The 13th International Symposium on – page(s): 9 – 14 - Sept. 2000

[8] Mendelbaum, B.H.G.; Gallant, R.; Brette, J.-F.; Ducateau, Ch.F. – *Java-prototyping of hardware/software CBS using a behavioral OO model* – Eng. of Computer Based Systems, 2000. (ECBS 2000) Procs. 7th Intl. Conference and Workshop on the – page(s): 73 – 81 - Apr. 2000

[9] Koch, B.; Grabowski, J.; Hogrefe, D.; Schmitt, M. – *Autolink-a tool for automatic test generation from SDL specifications* – Industrial Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE Workshop on – page(s): 114 – 125 - Oct. 1998

[10] Kim, Y.G.; Hong, H.S.; Bae, D.H.; Cha, S.D. – *Test cases generation from UML state diagrams* – Software, IEE Procs – page(s): 187–192 Aug. 1999

[11] *System Test via Use Case Based Testing (UCBT)* – IBM Research – Center for Software Engineering - http://www.research.ibm.com/softeng/TESTING/ucbt.htm

[12] Offutt, J.; Abdurazik, A – *Generating tests from UML specifications* – Second International Conference on UML. UML'99. Proceedings, pages 416-429 – Oct.99

[13] Abdurazik, A.; Offutt, J. – *Using UML collaboration diagrams for static checking and test generation* - Third International Conference on UML. UML'00. Proceedings – Oct.00