Application of a Formal Testing Methodology to Wireless Telephony Networks

Ana Cavalli[†], Amel Mederreg[†], Fatiha Zaidi[††]
[†]Laboratoire CNRS SAMOVAR, Groupe des Ecoles des Télécommunications/Institut National des Telecommunications, 9 rue Charles Fourier, F-91011 Evry Cedex France,
(email :{Ana.Cavalli,Amel.Mederreg}@int-evry.fr).
[††] Université Paris-sud 11, Laboratoire de Recherche en informatique
UMR CNRS 8623, Bât 490 Université Paris-Sud 91405 Orsay Cedex France, (email: fatiha.zaidi@lri.fr).

Abstract: This paper presents the application of a formal testing methodology to protocols and services for wireless telephony networks. The methodology provides a complete and integrated coverage of all phases of the testing procedure: specification, test generation, and test execution on a given architecture. It permits to perform conformance and interoperability testing detecting different kinds of implementation faults, as for instance output and transmission faults. The test execution is performed in the framework of a set of architectures capable to deal with different environments.
Telecommunication systems and mobility are the main focus of the application presented in this paper. Two case studies illustrates the application of the methodology to a wireless telephone network: conformance and interoperability testing of Wireless Application Protocol (WAP) protocols and services based on the subscriber location.

Keywords: component testing, conformance testing, interoperability testing, formal methods, location based services, mobile services, WAP, telephony networks.

1. INTRODUCTION

New trends in network technology lead to the design of new protocols and services. In most cases these networks, include heterogeneous elements that need to communicate among themselves. These elements need to be tested and experimented to guarantee their conformance to standards and their interoperability. Conformance and interoperability testing of these new products becomes a strategic activity in the telecom industry, both for the operators and for equipment vendors, as well as tool providers. Companies have to develop an important activity in order to guarantee the correctness of the behaviors of their software implementing these protocols and services. Due to this validation effort and experimentation on real platforms, trustable services will be produced and the time to market will be reduced. Methodologies integrating testing procedures in all phases of software life cycle will be one of the most profitable applications of formal testing techniques, and will address the companies concern of assessing applications in a systematic way.

This paper presents a conformance and interoperability testing methodology based on formal methods and its application to wireless telephony networks. The aim of this methodology is to cover all the steps of testing: from the formal system specification to test generation and test execution [16]. Even though a large number of testing methods have been already developed, only a few of them are able to automatically generate tests for real-life systems. Even in these cases, the results are not completely satisfactory. The main reason is the scalability problem associated with the existing algorithms. Indeed, until now research on automatic test generation for conformance testing was based on the *exhaustive simulation* of specifications, i.e. the exploration of the whole state-space (or *reachability graph*) [5,7]. The first drawback of these approaches is that, when testing real-world services or protocol components, the number of states is huge. The second one is that redundant parts may be explored uselessly. To avoid these problems, we have designed and implemented new algorithms, which are focused on components testing. They are based on the generation of partial reachability graphs, thus avoiding the combinatorial explosion of the number of states to be explored. They also avoid to perform redundant testing.

Another drawback of these methods [12, 11,15] is that they only consider the control part of the implementation under test. In our case, we deal with both control and data parts. The generated tests include parameter values in messages and consider variables in graph generation. Moreover, a set of test architectures capable to deal with a mobile phone environment have been defined. These architectures includes points of observation (to collect information from the implementation) and points of control and observation (to collect information but also to control the interactions with the implementation).

In order to illustrate the suitability of our methodology we have work on two real case studies: the Wireless Application Protocol (WAP) and several services based on the subscriber location. The WAP is an open global protocol that empowers mobile users with wireless devices to easily access and interact with Internet information and services instantly. To perform the experiments a software free protocol stack has been installed, namely Kannel. The other case study is on services related to users mobility and based on the subscriber location.

The work presented in this paper is part of a national French project, Platonis, that is carried out by a consortium composed of academic and industrial partners[8].

The original contributions of the paper are as follows. Our first contribution is methodological. We develop a testing methodology that covers all phases of software life cycle. The proposed methodology is original in that it combines automated test generation methods with test execution on different architectures. These architectures are generic and describe new aspects of the test of network communication protocols. They have been easily adapted to the mobile wireless environments presented in this paper. The proposed test generation methods and architectures facilitate the detection of output and transmission faults. Our second contribution is in terms of the application of our methodology to real case studies and the results of our experimentation. We present results on the test of relevant properties of WAP protocol and services. Our study is also unique in that it applies the methodology to real life applications, showing the scalability of the

proposed approach.

The paper is organized as follows. Section 2 introduces the proposed testing methodology. Section 3 presents the first case study: the application of the test methodology to the WAP protocol and the experimental results. Section 4 presents the second case study: the specification and test of location-based services and the experimental results. Finally, section 5 gives the conclusion and perspectives of this work.

2. TESTING METHODOLOGY

This section presents an outline of the testing methodology. This methodology has as objective to cover and to automate all the steps of the test production from the specification to its execution.

2.1. Steps of the testing methodology

The steps of the testing methodology are as follows:

1. To design a precise and concise specification [10], we use the SDL language, which is a standard of ITU-TS, as a Formal Description Technique (FDT). SDL allows to describe the architecture and the behavior of the system. The semantic model is based on Extended Finite State Machines (EFSMs) [13]. An EFSM is a state machine in which the states are parameterized with some free variables. Of course, the transitions possible on those states also depend on the values of those free variables, and may update the free variables in the resulting state. The behavior of a system is described in SDL as a set of cooperating processes, communicating through the exchange of signals. A check on local variable values imposes a condition (predicate) on moving to the next state. The actions in a transition include: the execution of tasks (assignment or informal text), procedure calls, dynamic creation of processes (SDL contains the concepts of "type" and "instance of type"), arming and disarming of timers. Each combination of a state and variable values consists of a configuration. The initial state with the initial variable values form an initial configuration. Data are defined as abstract data types. SDL 96 supports objects which allows to define generic types that could be validated and used in different contexts. It also supports ASN.1 [14], a standard defined for data transfer.

2. Once we have the specification, we can start the test generation process. First, we need to characterize the tests to be performed (i.e. to define test purposes according to predefined criteria in terms of fault coverage). For test generation, we use a test generation algorithm developed in our group. It is described in the following section. This algorithm has been conceived to perform component testing in context (i.e. testing a component which is embedded in a complex system). It has also been adapted to perform interoperability testing[6].

3. The generated tests are executed in a given test architecture. We actually perform execution

by using several test architectures which are described in section 2.3.

2.2. Outline of the algorithm

The algorithm allows to cover all the interactions of the component (protocol layer or service component) in its context. The interactions that we want to test are described by test purposes that guide the test generation. The algorithm works as follows : We consider a partial graph generated from the SDL specification. This partial graph is obtained by simulation. To consider only partial graphs allows to overcome the problem of state space explosion that occurs often when we try to produce the complete reachability graph from a model with data. The stepwise of the algorithm is explained in the following. We try to find a path that leads to our test purpose. If such purpose is found (which we call a *Hit*), we keep it for the final test sequence and continue the search process from there. Otherwise, we move randomly to the frontier of the partial graph (which we call a *Jump*), and resume the process from there. Thus, we avoid looping on the same state, as far as we choose uniformly and randomly in the spanning tree. We do not build a complete system accessibility graph. The number of steps of the search process is an execution parameter. This parameter can be determined by the user as a depth limit or a maximum number of states. As pointed out in [12], random walk may get "*trapped*" in a certain part of the component under test. Our algorithm allows us to *jump out of the trap* and pursue the exploration further. Also, since it builds a partial accessibility graph, it avoids the well-known state explosion problem as mentioned previously. As a result, our algorithm produces a test sequence for the test purpose. The algorithm terminates when all the interactions (test purposes) are found.

In [4] we showed how the algorithm has been adapted to perform interoperability testing. Indeed, to perform interoperability testing, we use as test purpose the interactions between distant entities (e.g. between a client and a server). This algorithm has been implemented in our tool called *TESTGEN-SDL* [7]. Figure.1 gives a description of the algorithm.

2.3. Test architecture

The proposed methodology is based on test architectures that integrate *Points of Observation* (or *POs*) and *Points of Control and Observation* (or *PCOs*). These architectures are illustrated by the WAP system. They can also be extended to other systems such as GPRS (General Packet Radio Service)[19] and UMTS (Universal Mobile Telecommunication System)[20].

WAP protocols are asymmetrical. In general, mobile terminals are clients that initiate a service. A WAP gateway, which is a server, receives messages from the client and performs proper operations such as making an HTTP request and sending it to the WAP application server. The proposed test architectures use several distributed access points with a WAP gateway. The behavior of each entity is observed through a PO and controlled through a PCO. Five levels of conformance and interoperability tests are performed:

1. The first level uses a PCO to control and observe the terminal exchanges, and two PO in the heart of the network. These two are placed between components (Figure 2 (a)), to detect

transmission errors and to perform traffic analysis. This architecture can also be used for network performance evaluation.

```
   Inputs : the system to test, a set of test purposes (a suite of
   inputs/outputs), the depth limit and the mode of search
   Output : Test sequence
   Condition := false
   while not condition do
    while a test purpose not found or limit not reached do
     From the current node of the system, we launch an exhaustive
     simulation.
    end while
    if one test purpose is reached then
          Keep track from the current node to the reached edge (add the
          path from the current node to the reached edge in the test
          sequence under construction)
          Throw away the partial reachability graph produced
          Reach a new current state
          Remove from the set the reached edge
          if the set of test purposes is empty then
           Condition := true
           Return the test sequence
    end if
    else  /* we reach a limit without hitting any edge of the test
          purpose */
          Consider the partial reachability graph produced
          Consider one spanning tree of this graph
          Examine all the leaf nodes of the tree and select one
          uniformly at random
          /* choice of one outgoing edge randomly and uniformly from
          each node)
          Include the path from the root to the selected leaf in the
          test sequence under construction
          Arrive at the selected leaf
    end if
end while
```

Fig. 1. Test generation algorithm

2. In the second level we observe and analyze log files. A PCO is on the terminal side, and a PCO and a PO are located in the gateway (Figure (b)). The gateway becomes an active tester (i.e. we have a remote test architecture). Notice that any mobile phone can be tested under this architecture.

3. In the WAP architecture, a number of services are possible such as web-based content services. To test these services, the following test architecture can be used (Figure (c)). In this architecture, we have one PCO in a mobile terminal and one PO in the web server. The tests that are executed on this architecture are those generated from the SDL description of the service. Moreover, we are able to check the behavior of the server and by this way we can test the interoperability between the terminal and the HTTP server.

4. The fourth level considers the network as a black box that is observed and controlled

through the PCO of the terminal (Figure (d)). This test architecture will be adapted to test the WAP protocols on the server side. This test is more difficult to perform. Since we only have one PCO and no PO, testing in context technique is needed to test WAP protocols layers.

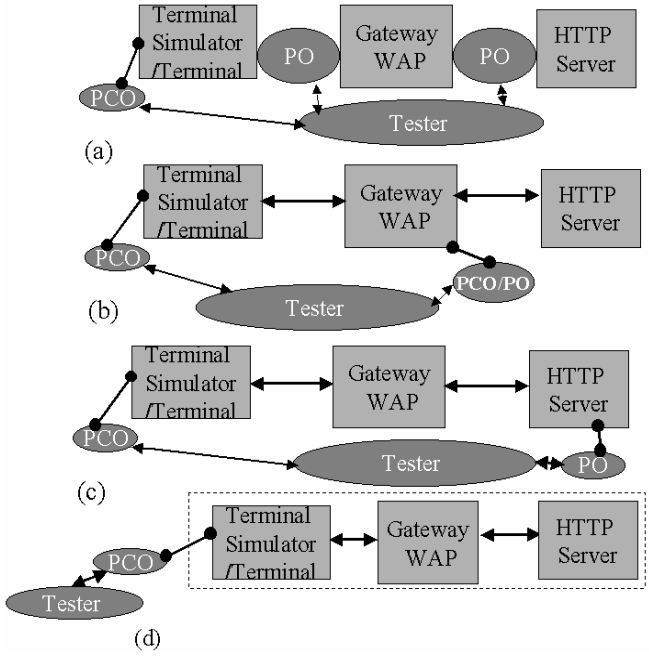

Fig. 2. Test architectures for interoperability testing

5. To test the conformance of the WAP stack, we use the test architecture proposed in Figure 3. According to the WAP specification, it is possible to access each WAP layer directly through service access points (SAPs). This facilitates the observation of the provided services of each layer [3]. However, this depends on the availability of application programming interfaces (APIs) of each layer. In our case, such APIs are available, as we used an open source gateway.

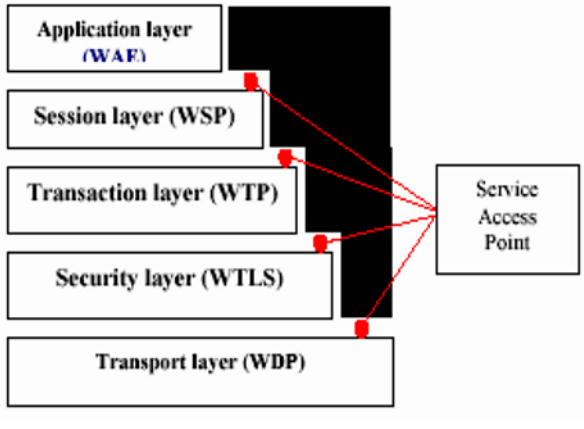

Fig. 3. Test architecture for layer conformance testing.

In this paper, we give an example of using the test architecture for testing protocols in the client

side and for testing WAP services. For the WAP services we are interested in testing location-based services.

3. EXPERIMENTS ON A WAP PROTOCOL STACK

This section describes how we implemented the test architecture for testing the interoperability between the server and the client and also the protocols implemented on the client side (i.e. the mobile phone). As a WAP gateway, we used the open source Kannel [1]. We have also installed an Apache HTTP server and three mobile phone simulators. To use a real mobile phone, we installed a remote access service (RAS) that allows access to our own Kannel WAP gateway in order to perform experiments on existing mobile networks.

Currently, three test architectures are available (Figure (b), (c) and (d)). Figure 4 shows the locations of POs and PCO in the WAP gateway that realizes the test architecture given in Figure.2.(b). POs were introduced between the WSP (Wireless Session Protocol), WTP (Wireless Transport Protocol) and WDP (Wireless Datagram Protocol) layers. These POs are used when we cannot program PCO on the gateway, it allows to test the behavior of the gateway that we want to test.

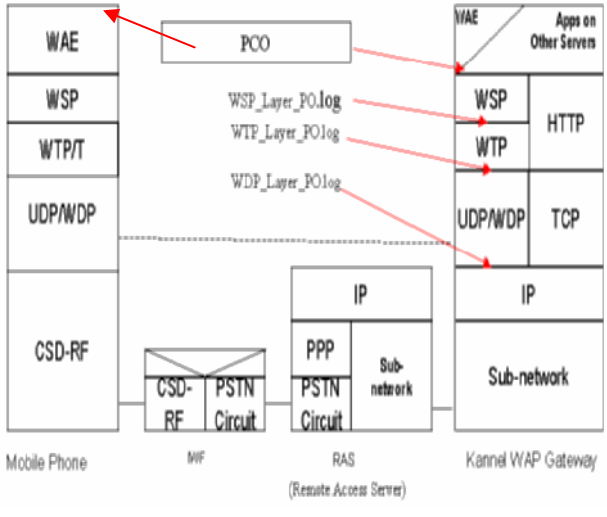

Fig. 4. Implementation of PCO and PO in Kannel WAP gateway

Figure.5 shows a PO collecting information from the WSP layer. This layer was designed to establish and release a session between a client and a server, to exchange content between the two applications and to suspend and resume a session. It has two types of sessions: connection oriented (which works over the WTP) and connectionless (which works over WDP). Using the PO, we observe that the connection between the client (*Nokia browser*) and the server (with IP address *157.159.100.113*) has been established using the connected mode (*9201*). The WSP message used to open the WAP page (*welcome.wml)* is *TR-Invoke.ind*[1]. The information that we can retrieve from this PO is descriptive enough to check the behavior of the layer. Indeed, we have the primitives' names, the transferred data, the reached states and so on.

[1] WTP primitive's used to establish a connection and to transport messages. More information could be found in [18]

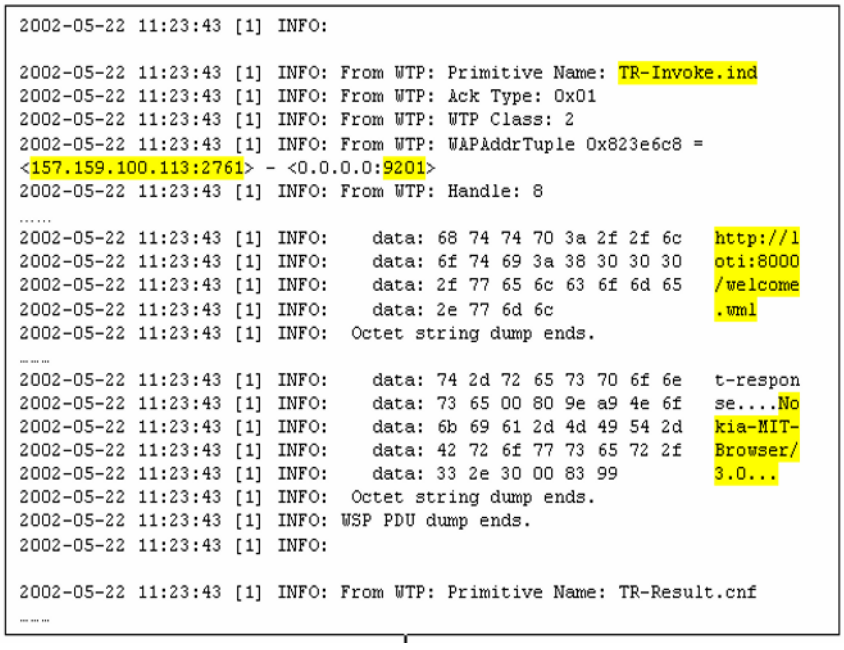

Fig. 5. WSP PO in Kannel WAP gateway

From a SDL description of the WAP layers, we use the algorithm presented in section 2.2 to generate automatically the tests. These tests are directly executed through the PCO on the real platform (Figure.4).

In the following we explain how we use our algorithm to generate a test sequence for the Abort Transaction test purpose. We choose it because we show below how we perform the test execution on the real platform. These test purpose permits to obtain interoperability test, indeed it expresses a end-to-end test sequence (from the client side to the server side). The Abort Transaction test purpose can be expressed by the steps shown in figure.6, which is used in the test generation algorithm explained above. When all test purpose steps of figure.6 have been exercised, we obtain a single test sequence (a suite of pair of inputs and outputs). The total sequence length is of 144 transitions (Table1). The sequence is executed on the platform according to the test architecture (Figure2(b)).

1- The WSP client's Layer sends an *Tr-Invoke.req* primitive to the WTP Layer to open the session.
2- From the WTP gateway's Layer, the WSP Layer receives an *Tr-Invoke.ind* primitive that informs the gateway that the client wants to open an session.
3- The WSP gateway's layer sends *Tr-Abort.req* primitive to the WTP Layer to abort the session and finish it.
4- The WSP client's Layer receives the *Tr-Abort.ind* primitive from the WTP that informs the client that the gateway had aborted the session.

Fig. 6. The Abort Transaction test purpose

To execute the tests, we submit the inputs provided by the test sequence to the implementation. For this we program a script to interact with the gateway. Figure.7 shows a part of a script that has been

produced for the WSP layer. A verdict is established by comparing the client and the PCO with the expected outputs of the test sequence. Moreover, to have a more accurate verdict we use the PO when an error occur.

```
PRIMITIVE NAME =TR-Invoke.ind CLASS = 2 HANDLE = 1 SOURCE IP
=157.159.103.89 SOURCE PORT =2710 DESTINATION IP =127.0.0.1 DESTINATION
PORT =9200 /
PRIMITIVE NAME =TR-Abort.ind SOURCE IP =157.159.103.89 SOURCE PORT =2710
DESTINATION IP =127.0.0.1 DESTINATION PORT =9200 /
PRIMITIVE NAME =TR-Invoke.ind CLASS = 1 HANDLE = 2 SOURCE IP
=157.159.103.89 SOURCE PORT =2710 DESTINATION IP =127.0.0.1 DESTINATION
PORT =9201 /
PRIMITIVE NAME =TR-Abort.ind SOURCE IP =157.159.103.89 SOURCE PORT =2710
DESTINATION IP =127.0.0.1 DESTINATION PORT =9201 /
PRIMITIVE NAME =TR-Invoke.ind CLASS = 2 HANDLE = 3 SOURCE IP
=157.159.103.89 SOURCE PORT =2710 DESTINATION IP =127.0.0.1 DESTINATION
PORT =9201 /
PRIMITIVE NAME =TR-Abort.ind SOURCE IP =157.159.103.89 SOURCE PORT =2710
DESTINATION IP =127.0.0.1 DESTINATION PORT =9201 /
```

Fig. 7. WSP PCO in Kannel WAP gateway

Figure.8 shows the content of the PO log file of the WSP Layer in the server side obtained for the test of the Abort Transaction exchange. According to the client answer and the PO of the WSP Layer the verdict is Pass for the Abort Transaction exchange.

```
2002-05-22 11:25:27 [1] INFO: From WTP: Primitive Name: TR-Invoke.ind
2002-05-22 11:25:27 [1] INFO: From WTP: Ack Type: 0x01
2002-05-22 11:25:27 [1] INFO: From WTP: WTP Class: 2
2002-05-22 11:25:27 [1] INFO: From WTP: WAPAddrTuple 0x8250520 =
<157.159.100.113:2769> - <0.0.0.0:9201>
.....
2002-05-22 11:25:29 [1] INFO: From WTP: Primitive Name: TR-Abort.ind
2002-05-22 11:25:29 [1] INFO: From WTP: Abort code: 0xe1
2002-05-22 11:25:29 [1] INFO: From WTP: Handle: 15
2002-05-22 11:25:29 [1] INFO: From WTP: WAPAddrTuple 0x8245678 =
<157.159.100.113:2769> - <0.0.0.0:9201>
```

Fig. 8. WSP PO in Kannel WAP gateway

Table I shows some of the test scenarios produced and executed on the test platform. Let us notice that almost all produced scenarios have a pass verdict. It means that the WSP layer is correctly implemented on the client side. It means also that the client and the gateway interoperate correctly. Nevertheless, the last scenario in Table I has no associated verdict. The raison of this is that we cannot exercise this scenario on the platform because it is not implemented in the gateway. We confirmed this by analyzing the content of the PO log file. We can also conclude the non-implementation of the scenario in the gateway by using the PCO in the interface of the client. Indeed if the client after disconnection asks to open a page, it will have a message saying that the client is disconnected and it has to be connected before asking to open a page.

Exchange	Test Sequence Length	Verdict
Successful Session Establishment	30	Pass
Completed Transaction	94	Pass
Session Suspension and Resume	84	Pass
Aborted Transaction	144	Pass
Refused Session Establishment	83	Pass
Active Session Termination	53	

TABLE I
WSP LAYER TEST SCENARIOS

4. SERVICE EXPERIMENTS LOCATION BASED SERVICES

The WAP offers a service that allows the users to access to the existing applications in the web (emails news..) every time and every where using the mobile phones. It is interesting to test both the correctness of the WAP protocols layers and also the application running on the top of it. This later is important because of the constraints imposed by the mobiles and the networks. Among the applications proposed by the WAP, the Location-Based Services (*LBS*) are the most specific applications on mobile telephony. The LBS are services that use the knowledge of the location of a user device. The knowledge of the user's location at any time adds value to the type of services that can be offered in the mobile telecommunication area. In this section we present the result of test experiments on the LBS.

From the SDL formal specification of the services, a set of scenarios is automatically generated using the method presented in Section 2.2. These scenarios allow us to do conformance testing and to detect errors related to unexpected or erroneous messages. Once the tests are generated, we can apply them to the WML service implementation in order to test the service functional behavior based on the architecture presented in Figure (c). In the following, we present some location-based services specification and the results of our experimentation.

4.1. Location based services description

A LBS service can be illustrated using the following scenario. A user requests a location-based service from an application server. When the application server receives the request, it sends the user information to the Mobile Positioning Center (*MPC*) to find out the current position of the user. The *MPC* uses one of the existing positioning technologies such as *GPS*, *A-GPS*, *E-OTD*, *CGI-TA* and *TOA* to get the current user position information [2]. After obtaining the answer from *MPC,* the

application server gives the result from the requested service to the user (Figure.9).

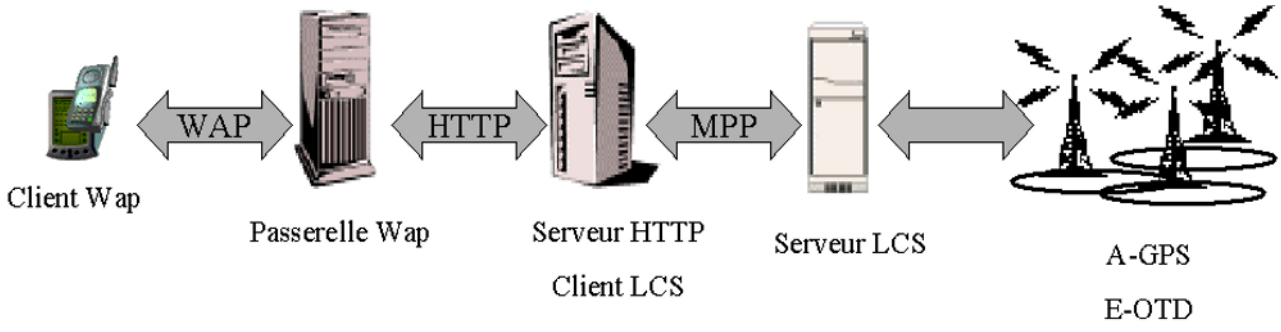

Fig. 9. Location Based Services Architecture

This kind of service, however, may pose problems with respect to the subscriber's privacy. For example, when a mobile user uses a searching service through his mobile phone and is looking for a restaurant, we can give the advertisement of a restaurant near the user. The subscriber may not want to let others know her/his location. To solve this problem we used a temporary identification of the user instead of the SIM (Subscriber Identification Module) number of the mobile phone. With the SIM number, which is a mobile permanent identifier, and a user authorization a service provider can localize a user at each moment. The temporary identification of the user given by the operator on user's demand is destroyed after each service provider use. A link between the temporary identification of the user and the SIM number is stored in a database which is updated. This solution allows the subscriber to be located only when she/he wants to be localized.

With the user position information a lot of applications can be developed. We actually have considered the following services (which we have specified in SDL):

- *Nearness* service: provides addresses that the user could be interested in, such as nearest Chinese restaurant or theater;

- *Itinerary* service: provides itinerary from the user's current position to a chosen destination by feet, by bus, by car, etc.;

- *Assistance* service: provides information in case of emergency. For example it can give the phone number of the nearest hospital;

- *Traffic* service: provides recent traffic information such as map of traffic situation;

- *Search* service: provides keyword searching on the databases for the above four services.

4.2. SDL Description of Location based services

The specification of the location-based services was done using SDL-96 in a way that it could be easily changed to add or remove some functionality. The specification of the LBS includes user, mobile terminal, application server, WAP gateway and MPC. The system is composed of four blocks (Figure.10):

• *Terminal* block contains the *Mobile* process, which describes one or more terminals and the user behavior. Each user has a profile that allows to determine which service she/he has subscribed to.

•*Network* block is composed of two sub-blocks : The *LCS* block which describes MPC behavior, and the *Operator* block which gives the temporary identification to the mobile and the corresponding SIM to the LCS. This identification is used to determine the position of the mobile;

• *Gateway* block describes the WAP gateway behavior.

•*Location_Services* block describes the application server behavior. The *Location_Services* block is composed of eight processes: *Menu* process; five service processes for *Nearness*, *Itinerary*, *Assistance*, *Traffic* and *Search* services; *Location* process, which converts the localization information from spatial coordinates to postal address, and *Manager* process which is the only static process of this block. It creates the other dynamic processes (Figure.11).

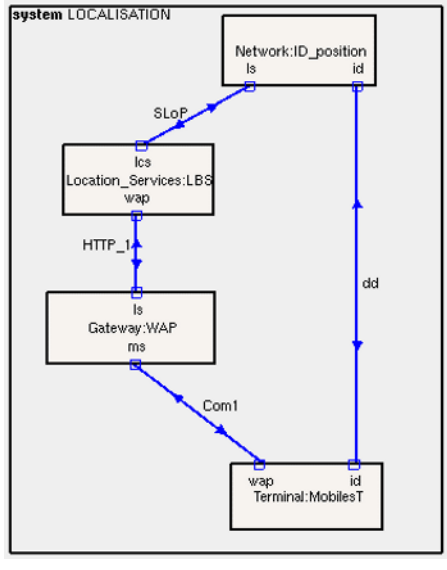

Fig. 10.SDL System

Location process, *Operator* process, *Menu* process, *Nearness*, *Itinerary* and *Traffic* service process have their own database. For simplicity, the functionality of *MPC* is specified by a database that contains the position information of each user. Moreover the databases are specified by arrays. A number of functions are described in SDL specification for manipulating arrays and reducing operation redundancy.

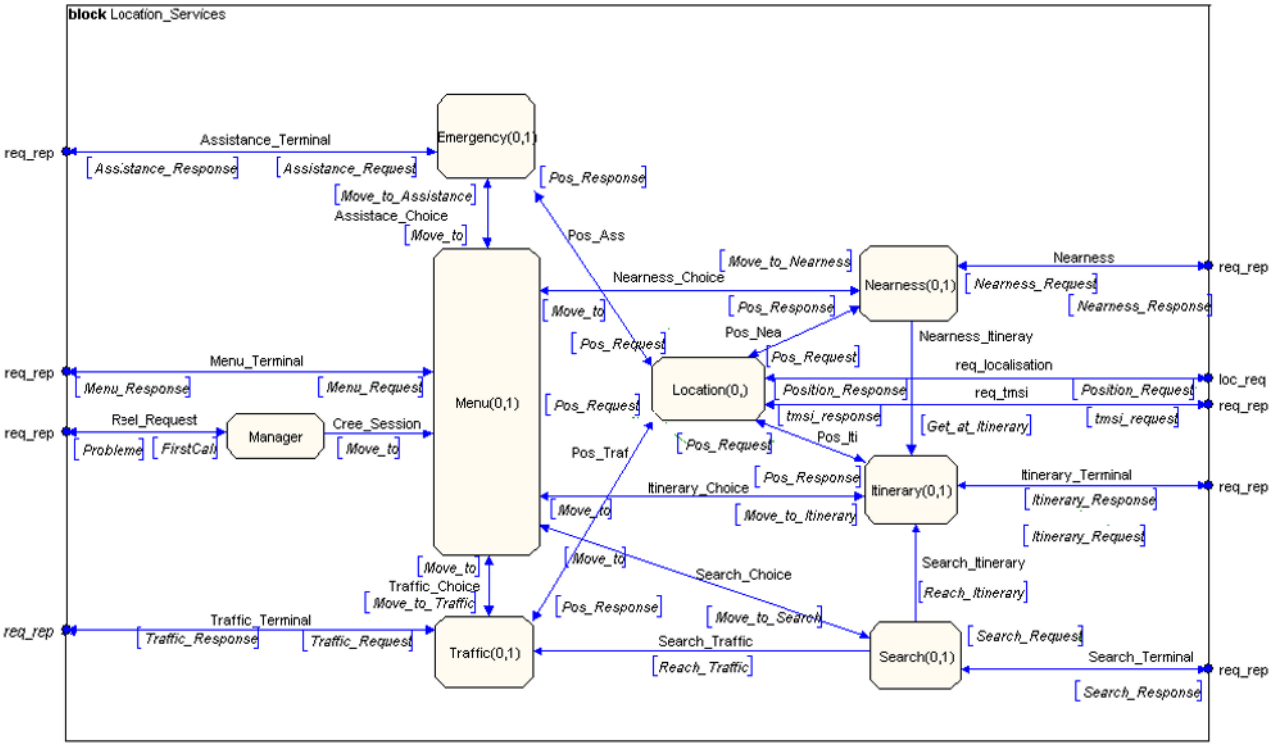

Fig. 11. Block Location_Services

IN ORDER TO GIVE A GENERAL IDEA OF THE COMPLEXITY OF THE SDL SYSTEM SPECIFICATION, WE PRESENT SOME SIGNIFICANT METRICS OF THE GLOBAL SYSTEM (TABLE II). IT WAS SIMULATED USING EXHAUSTIVE SIMULATION TO VERIFY THAT THE SPECIFICATION IS FREE FROM LIVELOCKS OR DEADLOCKS [17]. TO SIMULATE THE SYSTEM, WE USE A CONFIGURATION FILE WHICH INITIALIZES SOME VARIABLES SUCH AS DATABASES, SIM NUMBERS, AND THE NAME OF THE STREET OR THE KEYWORD TO LOOK FOR.

Lines	11709
Blocks	5
Process	14
Procedure	29
States	80
Signals	40
Macro definition	0
Timers	1

TABLE II
METRICS OF THE SERVICE SPECIFICATION

4.3. Generation of test scenarios for the Nearness service module

In this section, we present the experimental results for the generation of test sequences for the Nearness service module. The same method has been applied to the other services. As we have no direct access to the Nearness service module, the embedded testing technique is used. Whenever a

new service is developed, it will be added to the application server without affecting the other components (since we are using service testing in context).

The first step for generating test sequences is to define test purposes. To illustrated the application of the method, figure.12 shows four test purposes that have been tested for the Nearness service.

- *Test purpose 1*: To test whether, following a request from the user, the service requests the mobile position from the location service.
- *Test purpose 2*: To test whether the service when it gets the mobile position from the location service, asks the user to choose between different points of interest which are located near the mobile (e.g. restaurants; theaters, …).
- *Test purpose 3*: To test whether the service, when it receives the user's selection, asks her/him if she/he wants to end the service or access the Itinerary service.
- *Test purpose 4*: To test whether, when the Nearness service receives the user's intention to use the Itinerary service, sends a request to the Itinerary service to check for the best itinerary.

Fig. 12. Test purposes for the Nearness service

When all the above test purposes have been reached, the test generation algorithm terminates. The results obtained are illustrated in Table III. Once all the tests purposes have been exercised, we obtain a single conformance test sequence. The obtained sequence is of length 46 transitions and the execution times are relatively short (on a Sun Sparc Ultra 5).

Test purpose	Test sequence length	Duration
#1	20	4.0 s
#2	3	2.1 s
#3	36	8mn 23 s
#4	10	7.4 s

TABLE III
RESULTS OBTAINED

Notice that we could generate one test case for each test purpose. Figure.13 shows a part of the generated test sequences represented as MSC trace. Note that the generated test sequences include the behavior of all components (mobile, menu, Nearness and LCS).

The generated test sequences are executed following the architecture depicted by figure **Erro! A**

origem da referência não foi encontrada..(c) to test the existing WML implementation. We didn't find erroneous behaviors of the implemented services.

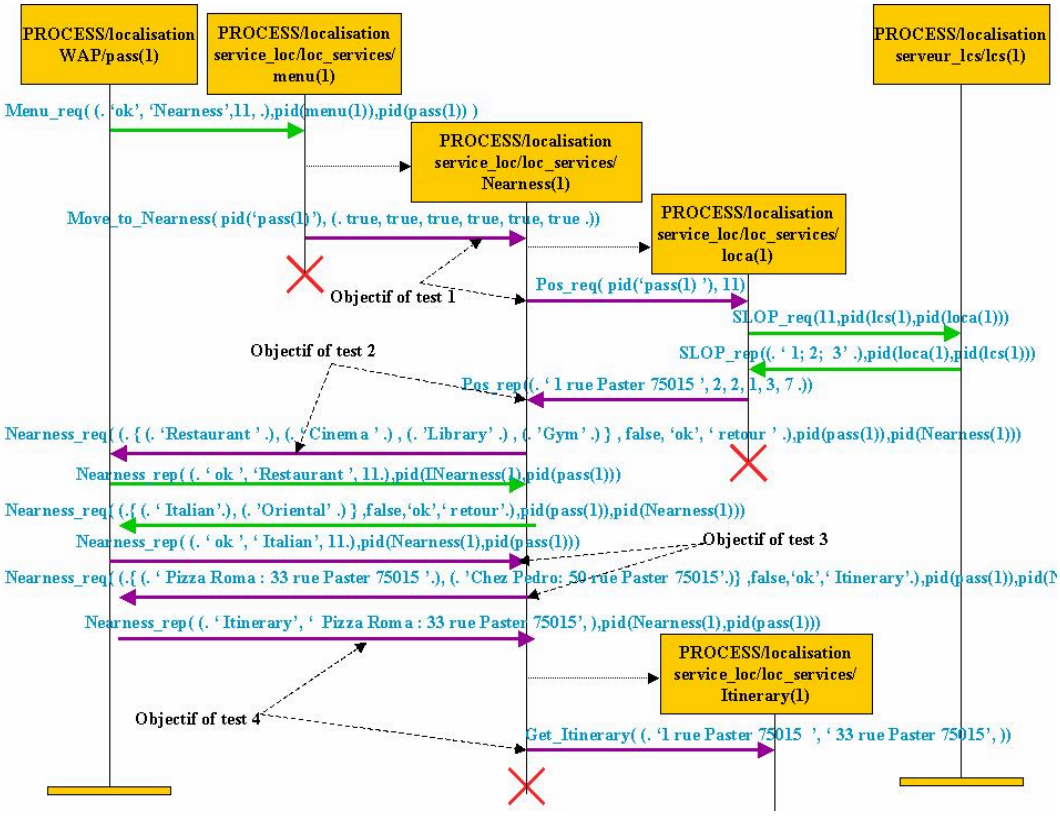

Fig. 13. Part of Nearness service MSC

5. CONCLUSION

In this paper we present a testing methodology and their application to two case studies. Concerning the methodology, the contribution of the paper is two fold: first, the proposed methodology covers all the phases of testing (specification, test generation and test execution); and second, it introduces several new architectures that are generic and include different aspects of the testing of network communication protocols and services. By integrating the test generation and execution with different architectures we achieve major improvements with respect to other works in this area where the focus is primarily on specific testing aspects: test generation, test execution, etc.

Two case studies are presented. The conformance and interoperability testing of WAP protocol layers and of services based on the subscriber location. We generate a number of test sequences based on given test purposes to check relevant aspects of the protocol and services behavior. The sequences are executed on an open source WAP a protocol stack (Kannel) in order to test protocol layers. Location based services are described and different test scenarios are generated based on test purposes and executed on the services implantation. Output and transmission faults can be observed and detected in very short delays.

Based on this work we are now in the process of testing other services that have been implemented using the WAP Modeling Language (WML). It is expected that service providers will be able to use these techniques to test their services and configurations, as for instance, services described using WML.

The work presented in this paper shows the relevance of using formal methods for the validation of real telecommunication systems. It also proposes new architectures that (to our knowledge) have not been introduced so far.

6. REFERENCES

[1] http: //www.kannel.org.

[2] http://www.wirelessdevnet.com/channels/lbs/features/mobilepositioning.html.

[3] Wireless application protocol architecture specification. Technical report, WAP Forum, April 1998. http://www.wapforum.org/.

[4] Cédric Besse, Ana Cavalli, Myungchul Kim, and Fatiha Zaïdi. Automated generation of interoperability tests. In Kluwer, *Testing Internet Technologies and Services*, Berlin, March 2002.

[5] R. Castanet and O. Kone. Deriving coordinated testers for interoperability. *Protocol Test Systems*, VI (C-19): 331–345, 1994. Elsevier Science Publisher B. V.(North-Holland).

[6] A. Cavalli, D. Lee, Ch. Rinderknecht, and F. Zaïdi. Hit-or-Jump: An Algorithm for Embedded Testing with Applications to IN Services. In *Proceedings of FORTE/PSTV'99*, Beijing, China, Octobre 1999.

[7] A. R. Cavalli, B. Chin, and K. Chon. Testing methods for SDL Systems. In *Computer Networks and ISDN Systems*, volume 28, pages 1669–1683, 1996.

[8] The PLATONIS Consortium. The platonis project. In *First International Workshop on Services Applications in the Wireless Public Infrastructur*, Mai 2001. http: //www-lor.int-evry.fr/platonis.

[9] N. Griffeth, R. Hao, D. Lee, and R. K. Sinha. Integrated system interoperability testing with applications to VOIP. In *FORTE/PSTV'00*, October 2000.

[10] ITU. *Recommendation Z.100 : CCITT Specification and Description Language (SDL)*, 1992.

[11] S. Kang and M. Kim. Interoperability test suite derivation for symmetric communication protocols. In *FORTE/PSTV'97*, 1997.

[12] D. Lee, K. Sabnani, D. Kristol, and S. Paul. Conformance Testing of Protocols Specified as Communicating Finite State Machines - A Guided Random Walk Based Approach. In *IEEE Transactions on Communications*, volume 44, No.5, May 1996.

[13] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. *Proc. of the IEEE*, 84(8): 1090–1123, august 1996.

[14] O.Dubuisson. *ASN.1*. Springer, 1999.

[15] O. Rafiq and R. Castanet. From conformance testing to interoperability testing. *The 3rd Int. Workshop on Protocol Test Systems*, 1990.

[16] M. Clatin, R. Groz, M. Phalippou, and R. Thummel. Two approaches linking test generation with verification techniques. In A. Cavalli and S. Budkowski, editors, *Protocol Test Systems VII*. Chapman & Hall, 1996.

[17] http://www.telelogic.com

[18]WAP Forum WAP™ WSP Version 4-May-2000. http://www.wapforum.org/.

[19] Digital cellular telecommunications system (Phase 2+): General Packet Radio Service (GPRS), Service description; Stage 2 (GSM 03.60), version 7.4.1 Release 1998, GSM 03.60, ETSI, Sophia Antipolis Cedex.

[20] 3GPP TS 25.305 version 3.7.0 Release 1999. "Universal mobile telecommunications system (UMTS);stage 2 functional specification of UE positioning in UTRAN". Technical report, ETSI TS 125 305 V8.3.0 (2001-10), January 2002.