# Application of a Formal Testing Methodology to Wireless Telephony Networks

Ana Cavalli[†], Amel Mederreg[†], Fatiha Zaidi[‡]

*Abstract*--**This paper presents the application of a formal testing methodology to protocols and services for wireless telephony networks. In particular, this methodology is focused on the test of components and has been adapted to perform interoperability testing. It covers all steps of the testing procedure: specification, test generation, and test execution on a given architecture. It permits to detect different kinds of implementation faults, as for instance output and transmission faults.**

**Telecommunication systems and mobility are the main focus of the application presented in this paper. In particular, two case studies illustrates the application of the methodology to a wireless telephone network: conformance testing of WAP (Wireless Application Protocol) protocol, and conformance testing of a service based on the subscriber location.**

*Index Terms*--**component testing, conformance testing, formal methods, interoperability testing, location based services, mobile services, WAP.**

## I. 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 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.

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 this approach is that, when testing real-world service 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 a new algorithm, *Hit-or-jump*, which is focused on components testing. It is based on the generation of partial reachability graphs, thus avoiding the combinatorial explosion of the number of states to be explored. It also avoid to perform redundant testing. An outline of this algorithm is provided in section 3.

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, our methodology covers all the steps of testing : formal system specification, test generation and test execution. This is not the case of the previous works that are focused on specific aspects of testing.

The proposed methodology is original in that it combines automated test generation methods with test execution on given architectures. The test generation methods and architecture facilitate the detection of output and transmission faults[16].

This paper presents the application of this testing methodology for conformance testing to two case studies: the conformance testing of WAP protocols layers and the conformance testing of a service based on the subscriber location.

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

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 on the WAP protocol stack and the experimental results. Section 4 presents the second case study: the specification and test of a location-based service and the experimental results. Finally, section 5 gives the conclusion and perspectives of this work.

## II. Testing Methodology

Our main objective is to define a methodology and architecture for the validation and experimentation of services and protocols. Our aim is to cover and to automate all the steps of the test production from the specification to its execution.

Moreover, our methodology deals with two kinds of tests : *interoperability* and *conformance testing*.

*Interoperability testing* is supposed to validate that different implementations interwork correctly (i.e. they provide the expected global service while complying with the standards). Interoperability testing allows for instance, to check the interoperability between an application running on a terminal and an application on a server. *Conformance testing* on the other hand, checks for the compliance of the implementation under test (e.g. the conformance of a given

[†]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).

WAP layer) with respect to its standards [3].

Our testing methodology is based on two main elements: the test generation method and the test architecture. The generated tests concern protocols and services as well.

The steps of the testing methodology are as follows:

1. To design a precise and concise specification. We use the SDL [10] language (which has been defined by the IUT-T) as a Formal Description Technique (FDT). The specification must take into account the test architecture and the interaction points with the environment. SDL allows to describe the architecture and the behavior of the system. The semantic model is based on Extended Finite State Machines [13]. Data are defined as abstract data types. SDL 96 supports objects which allow 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). In order to generate tests, we used a test generation algorithm that we developed in our group. This algorithm 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). However, we have adapted it to perform also interoperability testing.
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.C.

### *A. Test generation algorithm*

This section outlines the method used for test generation. A more complete presentation can be found in [6].

The aim of the method is to test a protocol, a service, a protocol layer or service components in their context. Testing components in the system is relevant because usually there is no a direct access to the component. The tested elements are defined as *Extended Finite-State Machines* (EFSMs) [13]. We use this type of machines because they allow to describe both the control and data part of the system. And also, because they are the underlying semantic model of the SDL language that is used as a formal description technique in our methodology.

### *B. Outline of the algorithm*

The algorithm allows to cover all the interactions of the component (protocol layer or service component) in its context. Our algorithm works as follows. First, we consider a partial graph from the EFSM. This partial graph is obtained by simulation. Then 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. This procedure avoids building of 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 a result, our algorithm produces a test sequence for the test purpose.

A more formal description can be found in [6]. 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].

### *C. 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) and UMTS (Universal Mobile Telecommunication System).

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 will be performed:

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 1 (a)), to detect transmission errors and to perform traffic analysis. This architecture can also be used for network performance evaluation.

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 1 (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.

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 1 (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.

The fourth level considers the network as a black box that is observed and controlled through the PCO of the terminal (Figure 1 (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.

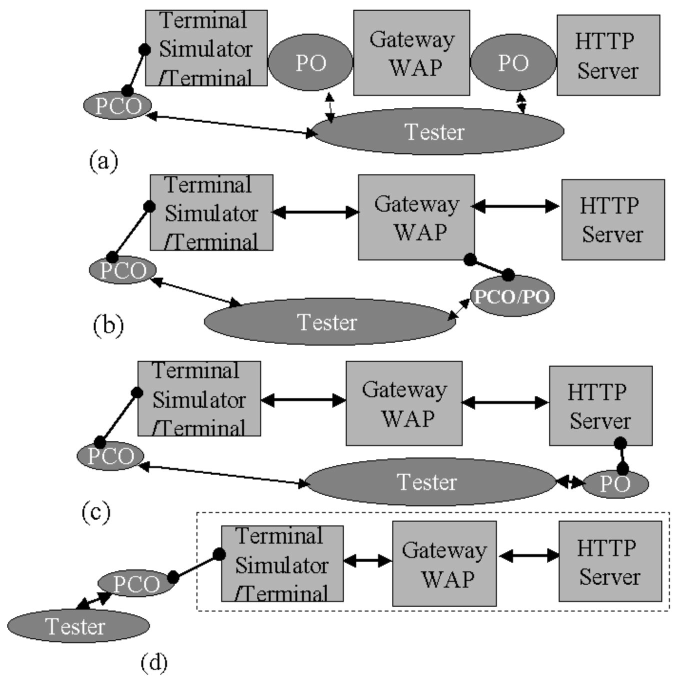


Fig. 1. Test architectures for interoperability testing

To test the conformance of the WAP stack, we use the test architecture proposed in Figure 2. 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.

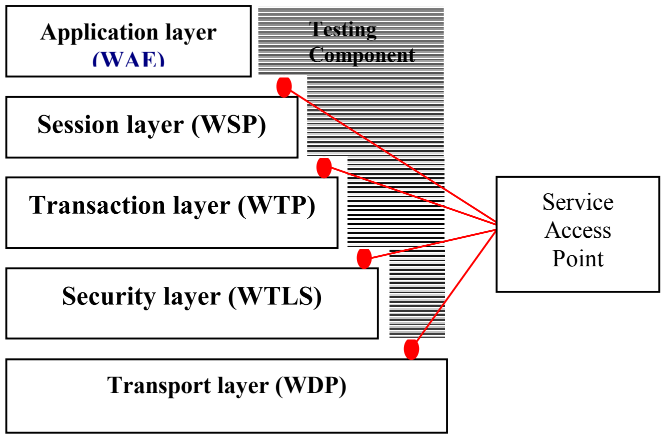


Fig. 2. Test architecture for layer conformance testing.

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.

## III. Real 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 the 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 real experiments on real existing mobile networks.

Currently, three test architectures are available (Figure 1 (b), (c) and (d)). Figure 3 shows the locations of POs and PCO in the WAP gateway that realizes the test architecture given in Figure 1.(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.

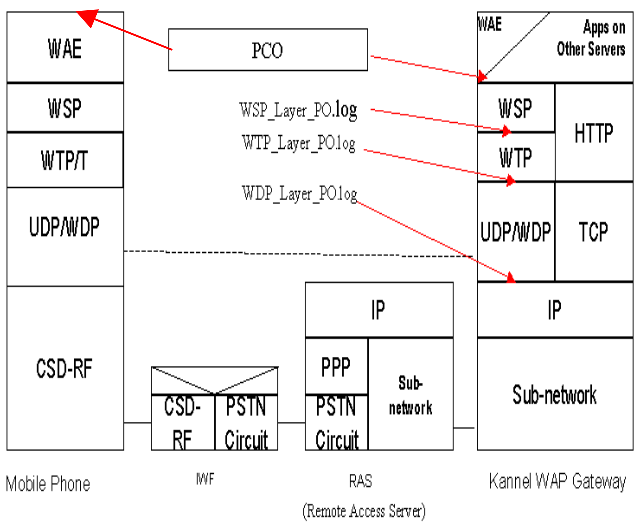
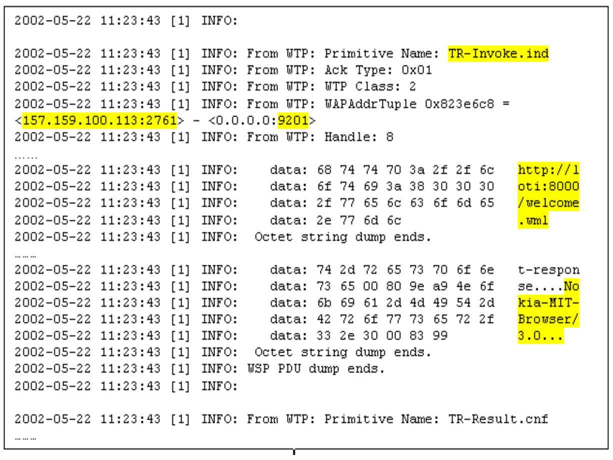


Fig. 3. Implementation of PCO and PO in Kannel WAP gateway

Figure 4 shows PO for 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*. 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.



```
2002-05-22 11:23:43 [1] INFO:

2002-05-22 11:23:43 [1] INFO: From WTP: Primitive Name: TR-Invoke.ind
2002-05-22 11:23:43 [1] INFO: From WTP: Ack Type: 0x01
2002-05-22 11:23:43 [1] INFO: From WTP: WTP Class: 2
2002-05-22 11:23:43 [1] INFO: From WTP: WAPAddrTuple 0x823e6c8 =
<157.159.100.113:2761> - <0.0.0.0:9201>
2002-05-22 11:23:43 [1] INFO: From WTP: Handle: 8
……
2002-05-22 11:23:43 [1] INFO:    data: 68 74 74 70 3a 2f 2f 6c   http://l
2002-05-22 11:23:43 [1] INFO:    data: 6f 74 69 3a 38 30 30 30   oti:8000
2002-05-22 11:23:43 [1] INFO:    data: 2f 77 65 6c 63 6f 6d 65   /welcome
2002-05-22 11:23:43 [1] INFO:    data: 2e 77 6d 6c               .wml
2002-05-22 11:23:43 [1] INFO:  Octet string dump ends.
………
2002-05-22 11:23:43 [1] INFO:    data: 74 2d 72 65 73 70 6f 6e   t-respon
2002-05-22 11:23:43 [1] INFO:    data: 73 65 00 80 9e a9 4e 6f   se....No
2002-05-22 11:23:43 [1] INFO:    data: 6b 69 61 2d 4d 49 54 2d   kia-MIT-
2002-05-22 11:23:43 [1] INFO:    data: 42 72 6f 77 73 65 72 2f   Browser/
2002-05-22 11:23:43 [1] INFO:    data: 33 2e 30 00 83 99         3.0...
2002-05-22 11:23:43 [1] INFO:  Octet string dump ends.
2002-05-22 11:23:43 [1] INFO: WSP PDU dump ends.
2002-05-22 11:23:43 [1] INFO:

2002-05-22 11:23:43 [1] INFO: From WTP: Primitive Name: TR-Result.cnf
………
```

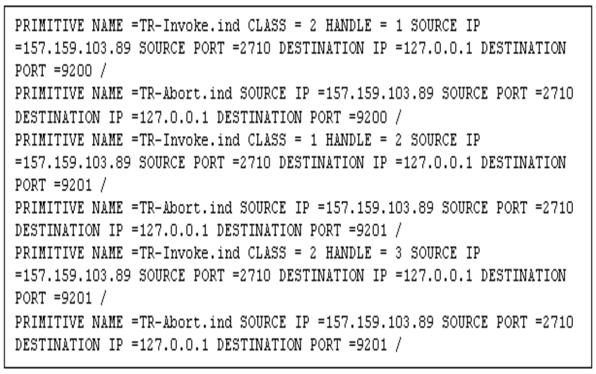
Fig. 4. WSP PO in Kannel WAP gateway

From a SDL description of the WAP layers, we use the algorithm presented above to generate automatically the tests. These tests are directly executed through the PCO on the real platform (see Figure 3). Table I shows the significant 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.

TABLE I
WSP LAYER TEST SCENARIOS

| Test Purposes | 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 | |

To perform the tests, we program a scrip to interact with the gateway . In other words, the script corresponds to the test scenario that we want to check on the client side. The figure 5 shows a part of a script that has been produced for the WSP layer. A verdict is established according to the client answer. Moreover, we can use the PO to have a more accurate verdict when an error occurs.



```
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. 5. WSP PCO in Kannel WAP gateway

## IV. SERVICE EXPERIMENTS LOCATION BASED SERVICES

The WAP is 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 application on the mobile telephony telecommunication. The LBS are services that use the knowledge of the location of a device user. 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 **Erreur! Source du renvoi introuvable.**. 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 to test the service functional behavior based on the architecture presented in Figure 1. (c).

In the following, we present location-based service specification and results of our experimentation.

### A. 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.

This kind of service, however, may pose problems with respect to the subscriber's privacy. For example, when a mobile user uses 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.

## B. *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 6):

• *Terminal* block contains the *Mobile* process, which describes one or more terminals and 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.

*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.

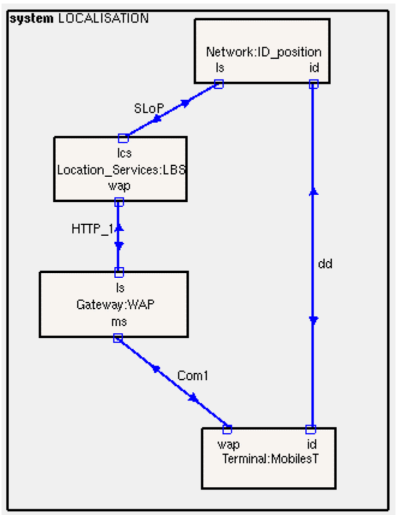


Fig. 6.SDL System

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.

TABLE II
METRICS OF THE SERVICE SPECIFICATION

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

## C. *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. For simplicity, we focused here on the following four test purposes 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.

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).

TABLE III
RESULTS OBTAINED

| 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 |

Notice that we could generate one test case for each test purpose. Figure 7 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 model of Figure 1.(c) to test the existing WML implementation. We didn't find erroneous behavior of the implemented services.

## V. Conclusion

This paper has presented the application of a testing methodology to two case studies. Concerning the methodology, the contribution of the paper is two fold: first, the proposed methodology covers all the steps of testing (specification, selection, test generation and test execution); and, second, it introduces new architectures for interoperability testing. This is different from other works in this area where the focus is primarily on specific testing aspects: test generation, test execution, etc.

Two case studies have been presented: conformance testing and interoperability testing of WAP protocol layers; and, conformance testing of a service based on user location. The results of the experimentation are very promising. Faults can be observed and detected in very short times. 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 interest of using formal methods for conformance and interoperability testing for the validation of real telecommunication systems. It also proposes new architectures that (to our knowledge) have not been introduced so far.

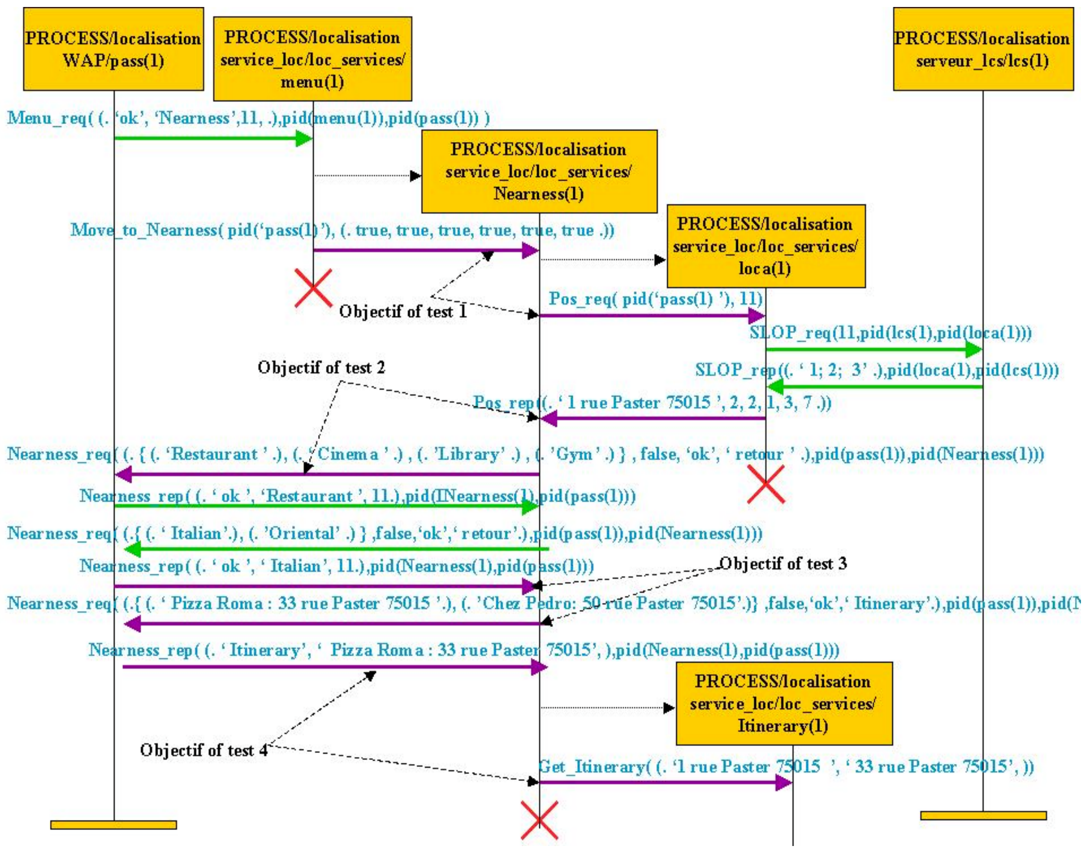

PROCESS/localisation WAP/pass(1)
PROCESS/localisation service_loc/loc_services/ menu(1)
PROCESS/localisation serveur_lcs/lcs(1)
Menu_req( (. 'ok', 'Nearness',11, .),pid(menu(1)),pid(pass(1)) )
PROCESS/localisation service_loc/loc_services/ Nearness(1)
Move_to_Nearness( pid('pass(1)'), (. true, true, true, true, true, true .))
PROCESS/localisation service_loc/loc_services/ loca(1)
Objectif of test 1
Pos_req( pid('pass(1) '), 11)
SLOP_req(11,pid(lcs(1),pid(loca(1)))
Objectif of test 2
SLOP_rep((. ' 1; 2; 3' .),pid(loca(1),pid(lcs(1)))
Pos_rep((. ' 1 rue Paster 75015 ', 2, 2, 1, 3, 7 .))
Nearness_req( (. { (. 'Restaurant ' .), (. 'Cinema ' .) , (. 'Library' .) , (. 'Gym' .) } , false, 'ok', ' retour ' .),pid(pass(1)),pid(Nearness(1)))
Nearness_rep( (. ' ok ', 'Restaurant ', 11.),pid(INearness(1),pid(pass(1)))
Nearness_req( (.{ (. ' Italian'.), (. 'Oriental' .) } ,false,'ok',' retour'.),pid(pass(1)),pid(Nearness(1)))
Nearness_rep( (. ' ok ', ' Italian', 11.),pid(Nearness(1),pid(pass(1)))
Objectif of test 3
Nearness_req( (.{ (. ' Pizza Roma : 33 rue Paster 75015 '.), (. 'Chez Pedro: 50 rue Paster 75015'.)} ,false,'ok',' Itinerary'.),pid(pass(1)),pid(N
Nearness_rep( (. 'Itinerary', ' Pizza Roma : 33 rue Paster 75015', ),pid(Nearness(1),pid(pass(1)))
PROCESS/localisation service_loc/loc_services/ Itinerary(1)
Objectif of test 4
Get_Itinerary( (. '1 rue Paster 75015 ', ' 33 rue Paster 75015', ))

Fig. 7. Part of Nearness service MSC

## VI. 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