Journal of Universal Computer Science, vol. 9, no. 10 (2003), 1220-1243
submitted: 15/1/03, accepted: 3/9/03, appeared: 28/10/03

An Interoperability Testing Approach to Wireless Application Protocols

Ousmane Koné
(Université Paul Sabatier - IRIT
118 route de Narbonne F-31000 Toulouse
kone@irit.fr)

Abstract: Internet services can now be used from mobile terminals. The main standard supporting this technology, WAP, will enable new services since it is compatible with network technologies like IP and UMTS. In parallel, powerful methods must be proposed to validate the underlying protocols in order to guaratee reliability and interoperability of new products. Our work, based on formal methods, contributes to WAP testing efforts by proposing an approach to the development of interoperability tests. We illustrate this approach with the design of tests suites for the WSP-protocol operating over a WAP transaction service.

Key Words: Formal Testing, Compliance, Interoperability, WAP protocol.
Category: D.2.1, D.2.5, C.2.2

1 Introduction

The deregulation of the telecommunication industry has led to the rapid evolution of technologies and know-how in the field of communication systems. Wireless communication is a good example where the rapid growth of the developments is undeniable. Research is active to enhance technical solutions on various aspects such as cellular transmission and routing issues [8, 19, 41], application and service issues [6, 13, 40]. On the one hand, customer needs are growing and manufacturers permanently develop new equipments with improved quality of service. On the other hand, these services must be very quickly validated so that the proposed solutions are reliable and ready to work. Therefore, in parallel, efficient techniques in communication systems engineering must be developed in order to reduce the time-to-market. In this context, interoperability testing is holding a strategic position in the design of new technologies.

1.1 Compliance/Interoperability testing

The problem of interoperability has emerged from the possible heterogeneity of computer systems. One would like computer products to be open systems, so that they can be interchanged with few constraints and interwork with systems belonging to other manufacturers. In the context of communication systems, the interoperability problem occurs when at least two entities specified or implemented separately are required to cooperate. A good way of achieving a successful cooperation is to define reference standards. Then these standards must be checked for correctness and their implementations must also be checked for correctness. *Verification* is the activity which is concerned with analyzing specification properties in order to detect possible inconsistencies. *Conformance testing* is the activity which is concerned with experimenting an implementation in order to check its correctness according to a given specification. Substantial efforts have been undertaken is these fields of protocol engineering to ensure communication systems of good quality. But computer systems are more and more complex, and in practice, it is difficult to test them in an exhaustive manner. Moreover, these systems may have different manufacturer profiles which may be, possibly, incompatible. Consequently, one cannot guarantee that such systems, however conforming they may be, can successfully interwork. The need to experiment them in order to show their aptitude to interwork is therefore straightforward. Testing different implementations together, in view of checking their aptitude to interwork is referred to as *Interoperability testing*.

1.2 Internet and Mobility

The last decade has been marked by two challenging technologies: Mobile telephony and Internet. Mobility has become an integral part of everyday life. No need to recall the widespread use of mobile terminals for both business and private use. The other undoubtedly successful phenomenon is the Internet. The exploding number of existing servers, the electronic commerce, etc show the increasing interest of the World Wide Web (WWW). But the WWW has been developed mainly for desktop or larger computers. To couple mobility and access to Internet services has become another customer need which promises new challenges in network technology.

Wireless Application Protocol (WAP). WAP specifications [27] are standards defined for the development of communication systems that operate over wireless networks. Recent versions of these standards support UMTS and IP networks. WAP implementations must be tested against standards in order to check their conformance and their aptitude to operate with one another. According to the WAP forum, this testing process would follow two tracks/phases:

- Track one: *Certification testing*. This track would be focused on WAP application level. The application environment under test may be operating on both client or server side and checked against the standards.

- Track two: *Compliance testing*. This track would be concerned with the protocol level. Implementations would be tested through the lower level Service Access Points in order to check protocol operations for compliance with the standard.

To date, the main developments have been concerned with track one and recently, the WAP forum has announced the availability of certification test suites. We are not aware of developments regarding compliance issues which are the concern and contribution of our paper. Compliance involves protocol *inter-operations*, and, therefore, in this paper, we make no difference between *compliance testing* and *interoperability testing*.

1.3 Testing issues

A substantial research activity has been done in the field of conformance testing which compares some particular implementation to some reference standard [3, 15, 22, 38]. As explained before, this is a preliminary (but not sufficient) work towards enabling heterogeneous implementations to work together. But additional test experiment, where different occurrences of implementations are interconnected and faced one against the other, must be performed in order to show their aptitude to interoperate. The interoperability test design approach we adopted for WAP, follows a generic methodology that we named Top [2, 21, 24], and which handles interoperability test automation from specifications to test execution via a CORBA platform. We will not recall all aspects of Top in this paper, rather concerned with test cases design only. The test cases developed are automatically computed from a formal model of WAP specifications. Few works exist in the field on interoperability testing, compared to other testing approaches. Some organizations such as SPAG (Standards Promotion and Application Group) and ATM forum [37] contributed to clarify the understanding of interoperability testing issues. The Top approach follows the recommendations of these organizations, mainly for testing architecture aspects. The other aspects concern test cases generation. Methods from the literature generally compute interoperability test cases on the basis of behavior graph modeling the communicating entities [4, 9, 31, 35, 39]. These methods have been interesting contributions in automatic test generation. However in the future, the need to face the increasing complexity due to powerful features of new communication systems, plus concurrency, will grow up. The Top approach (implemented in the tool named Tescoms [2, 21]), tries to tackle this problem with an on-the-fly computation approach. We have successfully experimented the on-the-fly paradigm with complex real-time specifications [20], and recently, we presented the applicability of this approch to the interoperability of the WAP protocols [23, 25].

This paper presents our experiment with compliance/interoperability test suites development for WAP. After a brief presentation of our testing approach, we show the application and results with the WAP session layer. The rest of the paper is organized as follows: Section 2 recalls the main functionalities and architecture of WAP. Section 3 presents an overview of the Top methodology which,

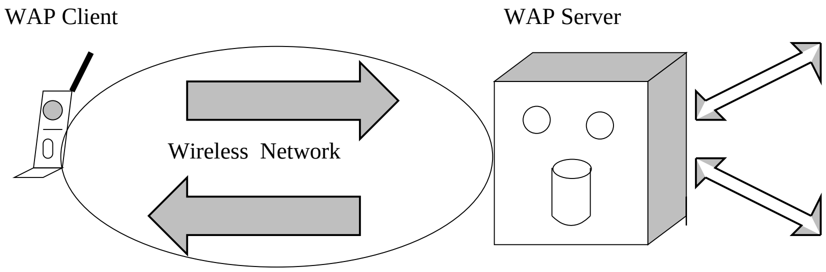

Figure 1: Wireless Network

in the sections after, is followed for test design. Section 4 addresses compliance testing and its illustration with the WAP Session layer (WSP). In section 5, we analyze the production of test suites for WSP.

2 Wireless Application Protocols

2.1 General functionalities

WAP is aimed at enabling mobile terminals to access Internet-like services. Within the WWW, a computer program can ask another one to execute some request. The first program is called *client* and the second one is called *server*. In order to insure interoperability, the WWW is built over standards such as HTTP [11] and HTML [12]. The WWW model has influenced the WAP model very much. Distributed wireless applications are also composed of clients and servers (Figure 1). Mobile clients have access to both WAP-origin and WWW servers. A WAP-origin server is, for example, a Wireless Telephony Application server which is a direct access point of a given wireless network provider to the WAP-client. But the communication with a WWW server requires to go through a WAP-proxy server which stands for an intermediary node between the two technologies. The mobile application is programmed with WML (Wireless Markup Language) which is very close to HTML. The mobile terminal is provided with a micro-browser for WML. WAP functionalities are carried out by a family of protocols that constitute the WAP-stack.

2.2 WAP Architecture

The WAP architecture has some similarities with the ISO-OSI basic reference model [16]. Protocol features are organized into different layers (Figure 2). We shortly describe these layers below.

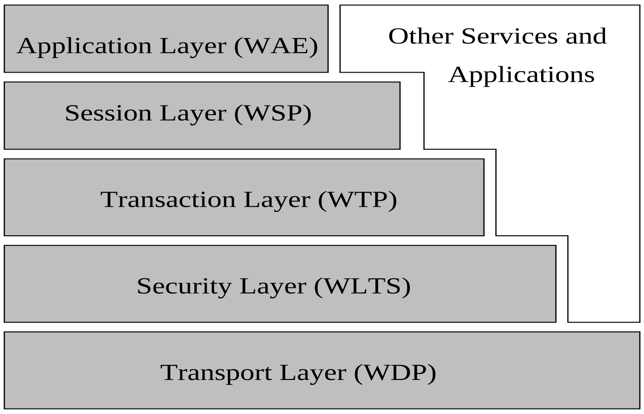

Figure 2: WAP protocol stack

Application layer. This layer provides the mobile terminal with a uniform environment - the WAE (Wireless Application Environment) - enabling access to standard telephony and WWW services. The WAE includes a WML browser.

Session layer. The session service is achieved through the WSP (Wireless Session Protocol). It is used by applications such as the WML browsers. All the internal mechanisms necessary to maintain the session life are transparent to the application. We will be back to this layer in section 4.

Transaction layer. The WTP (Wireless Transaction Protocol) is involved in the execution of transactions requested by upper layers. Optionally, transactions can be confirmed and WTP must manage the acknowledgments (delay them or not).

Security layer. Mobile applications necessarily operate over some transport service (next item). The WTLS (Wireless Transport Layer Security) is intended to secure the data transported and insures classical features such as data integrity, privacy and authentification.

Transport layer. As the WAP technology enables the possible use of various networks bearers, it is necessary to define a uniform transport service, accessible to standard upper layers. The role of the WDP (Wireless Datagram Protocol) is to insure this transparency.

Structure of the WAP services. The structure of WAP model (Figure 3) is similar to the one of the OSI basic reference model. An (i)-WAP layer is com-

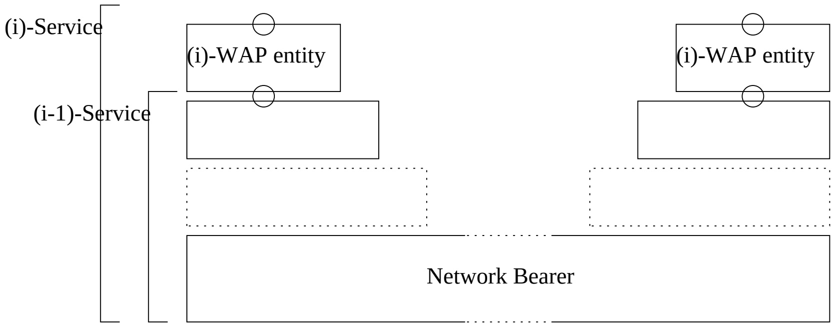

Figure 3: Structure of WAP services

posed of (i)-WAP entities using a (i-1)-WAP service. One of the particularities of WAP structure is that some (i)-service can be directly used by upper layers or applications. For example, the Transaction layer or further defined applications can directly access the Transport layer.

Normative references tell more about WAP. The reader may refer to the standard [27] for detailed information.

3 Overview of the Top approach

This section describes the main characteristics of the methodology adopted for WAP test development. The Top methodology is a contribution to normative testing area for the interoperability of communicating systems. Preliminary experiments with Top have been undertaken with standards such as ATM Adaptation Layer, or OSI protocols [14, 17]. Similarly to ISO 9646 [15], Top is organised in two phases: the Static Interoperability Review (SIR) and the Dynamic Interoperability Review (DIR). The SIR precedes the DIR. It is a necessary work as implementations' profiles can be different a priori.

3.1 Static Interoperability Review

Assume that we are interested in testing the interoperability of two devices A and B, which are typically two protocol implementations. S_A (resp. S_B) denotes the specification of device A (resp. device B). The SIR consists of an analysis of specifications S_A and S_B in order to get rid of incompatibility problems. This phase is destinated to check that the options implemented by A are compatible with the ones implemented by B. It is also a preparatory work to the selection of `TP` (*Test Purpose*, cf. next subsection) in dynamic test cases generation. For instance, it is no use testing the `Suspend/Resume` functionality (cf. WSP protocol, next section) if the client implements it while the server does not. The SIR follows some essential characteristics of the OSI conformance methodology and framework (ISO 9646) and WAP forum, in the sense that it also requires the examination of both specifications and ICS (Implementation Conformance Statement): The ICS stipulates the features of the specification effectively implemented by the device under test. Notice that in conformance testing methodology, checking the ICS may be necessary to select relevant tests for the DUT (Device Under Test, or IUT: Implementation Under Test). This phase is named Static Conformance Review (SCR). Similarly, in the interoperability methodology, as a preliminary study, the features of implementation A must be analysed against the ones of implementation B. Notice that this work is undertaken by hand, with a careful reading of ICS. A (semi-automatic) time analysis was also added to the SIR as it appeared that real-time features of communicating entities might influence their aptitude to interoperate [5, 26]. This aspect is an optional investigation of the SIR since very few standards specify real-time requirements.

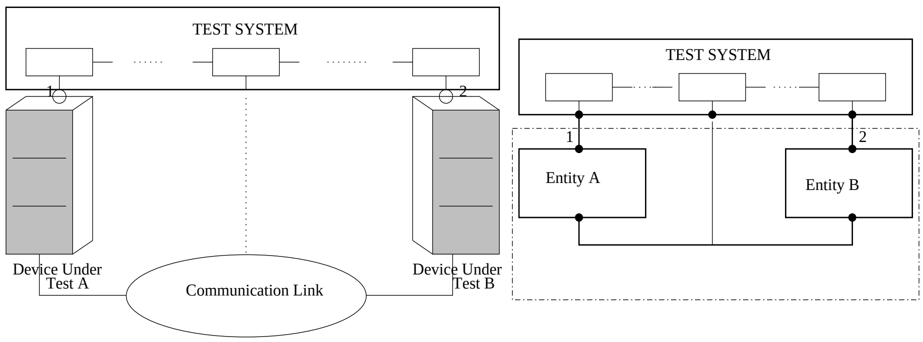

Figure 4: Functional views of the Test Architecture

3.2 Dynamic Interoperability Review

DIR is concerned with the testing of different implementations which are currently working together. This phase necessitates the design and the execution of test cases showing that the implementations under test can interoperate or not. The final aim of cooperating implementations is to provide some expected service. We lay emphasis on the service aspect even if we consider that interoperability necessitates the conformance to the specified protocol. Three issues are to be considered for DIR: test architecture, test cases generation (test suite design) and test deployment.

Test architecture and deployment. The test architecture represents the functional configuration in which the testing experiment will be undertaken. Figure 4 depicts functional views of the architecture we adopted for interoperability. In this figure, entity A and entity B represent DUTs (Device (or Implementation) Under Test). The Service Access Points (SAP) are labeled with 1 and 2. The test system interacts with the DUTs through the SAP. An optional interaction point enables it to access the PDU (Protocol Data Unit) at lower communication level. Notice that this architecture is similar to the one proposed by the ATM forum for interoperability. The monitor point of ATM forum corresponds to the optional interaction point in Figure 4.

But those architectures, including the ones defined in standard [15], describe only functional configurations. In this case, they do not provide information for a concrete design of PCO for the interaction between the test system and DUTs. This aspect is left to test laboratory and test execution concern. In our methodology, we have designed a portable test platform for handling test deployment issues [24]. The platform is featured over the CORBA standard, which facilitates test deployment for a wide range of DUTs. We do not develop test deployment aspects in this paper.

Test case generation. A test campaign consists of submitting interactions to implementations while observing their reaction. Test cases can be automatically computed if formal models of reference specifications are available. Our test generation tool implements an *on the fly* exploration technique, which is recognised to be efficient against complex specifications. One straight mean of obtaining interoperability test cases would consist of computing a reachability/global behaviour graph to be used as a reference model of the communicating entities under test. Then standard test generation tools could be used to select test cases from that reference model. With such an approach, one may consider managing the possible large size of the resulting behaviour graph. Our algorithm does not compute a global behaviour graph **before** test selection. It directly extracts test paths by analysing the sub-specifications (model of entity A and model of entity B). As illustrated by Figure 5, our algorithm works with input/output automata for modelling the specifications and the Test Purposes. A *Test Purpose* (TP) is an (abstract) *requirement* while a *Test Case* is an actual (dynamic) behaviour that meets a given TP. For instance, assume we are interested in the following requirement, depicted by the TP in Figure 5: “Some input (resp. Request) `a1` may be followed by some output (resp. Indication) `e2`”. A test case which can show this particular feature, against implementations of Figure 5, is the behaviour $\xrightarrow{a1}\xrightarrow{c}\xrightarrow{e2}$ (bold part of the figure).

The theoretical foundations of this algorithm can be found in [20, 21]. For

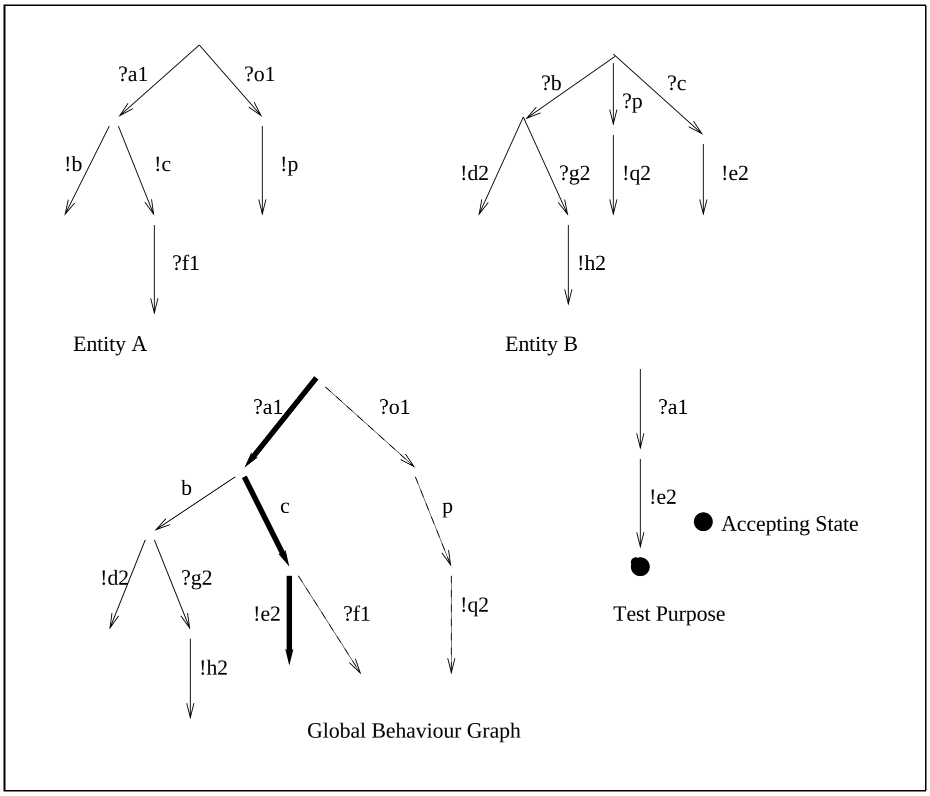

Figure 5: Computation of specification models - ?x=input x, !y=output y

space and readability reasons, we do not provide more theoretical details here. The technique presented in [20] is adapted here for testing concurrent implementations (but without real-time considerations). We illustrate it with the example of Figure 5.

3.3 The test generation algorithm

In this section, we sketch the Top test generation algorithm. The algorithm basically uses a DFS (Depth First Search) to explore a concurrent graph until a successful test case is computed. A transition is firable in this graph if it is firable in both specifications and test purpose, or it is firable in specifications only. The firability of transitions is examined, depth-wise, with some successor function, namely `lnext()`, and the algorithm successfully exits when the traversal of the test purpose automaton has been completed (This is done when an accepting state, i.e. a state belonging to the set `Accept(TP)`, is reached).

- (s_1, s_2, s_p) is an identifier containing a reference to the actual global state of (specification S_A, specification S_B, TP).
- `lnext()` returns a list of all possible successors for a given global state (s_1, s_2, s_p). In the example of Figure 5, the transition labeled with a1 is a possible successor of $(s_0(S_A), s_0(S_B), s_0(TP))$, where s_0 denotes the initial state of a given automaton.
- `GStack` is a global stack containing the actual global states to be evaluated. Elements of this stack are of the form $((s_1, s_2, s_p), L_s)$ where (s_1, s_2, s_p) is a global state identifier and L_s its related successors.
- `Explored` is the set of global states already explored.
- `IOTest` is a stack containing the path under search.
- `Plus()` adds one element to a given set.
- `LIFO-in()` is a push function adding one element on the top of a stack.
- `LIFO-out()` is a pop function removing one element (the top element) from a stack.
- `First()` returns the value of the first element on the top of a stack.
- The symbol $\emptyset$ denotes an empty set or stack.

The algorithm is started with the initialization of the sets and stacks to empty set.

If exercised with the specifications and test purpose of Figure 5, the algorithm of Figure 6 produces a test case which is depicted with the bold part of the figure. As stated before, this algorithm is based on a DFS which has been shown to be powerful for graph exploration and test generation [1, 3, 7, 10]. The DFS is very popular and is known to be a linear time algorithm [1, 20]. This algorithm has been adapted and implemented in the so-called TESCOMS tool [2, 21] by our students, in C++ language. The input of the tool are finite input/output automata modelling the specifications and the test purpose.

4 Compliance testing with TOP

This section addresses test case generation for WAP. In the context of compliance testing, we rather consider protocol level and we propose the design of tests aimed to check whether WAP protocol implementations interoperate, i.e. whether they communicate successfully and provide the expected service. Compliance testing involves the observation of implementations under test through Service Access Points.

```
begin
GStack := ∅; Explored := ∅; IOTest := ∅;      // Initialise to emptyset
Plus((s0(SA), s0(SB), s0(TP)),Explored);
LIFO-in(((s0(SA), s0(SB), s0(TP)), ∅),IOTest);
LIFO-in(((s0(SA), s0(SB), s0(TP)),lnext((s0(SA), s0(SB), s0(TP)))),GStack);
while (GStack ≠ ∅) do
begin
    ((s1, s2, sp), Ls) := First(GStack);
    if (Ls ≠ ∅) then
    begin
       // Visit the next successor of current state (s1, s2, sp)
       (μ, (s'1, s'2, s'p)) := First(Ls);
       LIFO-out(Ls);
       if (s'1, s'2, s'p) ∉ Explored then    // If not explored yet
       begin
          LIFO-in(((s'1, s'2, s'p),lnext((s'1, s'2, s'p))),GStack);
          LIFO-in((μ, (s'1, s'2, s'p),),IOTest); // Update the test searched
          Plus((s'1, s'2, s'p),Explored);
          if (s'p ∈ Accept(TP)) then // An accepting state is reached
             return IOTest;          // Stop the search and exit
       end
    end
    else
    begin
       LIFO-out(GStack);   // There is no more successor to visit
       LIFO-out(IOTest);    // Current test case does not match
    end
end
return ∅;      // Search failed
end
```

Figure 6: Algorithm skeleton

4.1 Top meets WAP

The presented testing methodology is suitable to WAP communicating systems. Compare Figure 3 and Figure 4. As explained in section 2, each WAP layer is intended for providing some service. This service is to be used by actual or further upper layers and applications. It is obvious that the reliability of a given layer depends on the reliability of the underlying services. Testing the correctness of the service against standard specifications is therefore fundamental. The WAP architecture enables access to each layer of the WAP stack and the Top architecture is powerful to test distributed systems through their service boundary, with possible access to lower level protocol operations.

Moreover, WAP specifications include, for each layer, the protocol description by state/transition tables. We modeled the specifications (client entity, server entity) with the SDL Formal Description Technique [18] and then in the form of finite input/output automata, which were further input to our test generation tool. The following subsections tell more about test design for WAP.

Because of the recursive structure of WAP service layers ((i)-Service ≡ (i)-Protocol + (i-1)-Service), our work can be applied to each WAP layer. In this paper, we focus on our experiment that started with the WAP session layer (WSP), which is the top of the layers underlying the WAP Application Environment.

4.2 WSP specification and reference model

WSP enables client and server applications to exchange contents through a maintained communication session. WSP specifications [29] include a connectionless protocol over a datagram service, and a connection-mode protocol over a transaction service. In the sequel, we consider the connection-mode protocol only, as the other one is fairly simple.

WSP is mainly featured to establish and release a session between a client and a server, exchange contents of the two applications, suspend and resume the session. The WSP specification is defined over a set of service primitives and their ordering by time sequence charts. A service primitive is of the form `S-Service-type`, (e.g. `S-Connect-req`), where `S` is the label of the session layer, `Service` indicates the name of the service (e.g. `Connect`). The type (e.g. `request`) is also indicated. Protocol operations are performed through the use of a transaction (`TR`) service. The PDUs exchanged between the WSP peer-entities are encapsulated in the `TR` primitives: `TR-Service(PDU)` denotes the encapsulation of a given `PDU` in a `TR-Service` primitive. For example, `TR-Invoke(Connect)` represents the encapsulation of the `Connect` PDU in `TR-Invoke`. The session protocol is described with a set of state tables. We used a model of the client and the server protocol entities in terms of input/output automata as this was the formalism accepted by our test generation tool (In the appendix section, a subset of the transitions illustrate the specifications). In order to check the correctness of our model, we defined it in the SDL language before, so that it could be verified with the `ObjectGeode` simulator/debugger. The specification file contained about 3000 lines of SDL code. Of course, the verification of the model revealed some modeling errors (some transitions were forgotten) and mainly some unused transitions. It appeared that the unused transitions corresponded to the reaction of entities against possible errors introduced by the underlying layer. For the time being, our test design methodology does not deal with error generation which falls in the category of robustness testing. Error cases due to the communication environment are various and not controllable. Our test design process is based on automatic computation of the behavior part assuming a normal operating environment. Further studies will investigate error generation. Another specification issue concerns optional services (e.g the `Push` service). Optional services are not systematically included in our reference model. We described these services as SDL processes which may be added or not to the mandatory part of the specification.

4.3 On the selection of test cases

In the WSP specifications, implementation features are organized in groups of functionality (`Session creation, Session suspend/resume, Push facilities` etc). Two kinds of features are specified: mandatory and optional. Mandatory features must be implemented by both client and server. In the WSP standard, each optional functionality has been specified as a whole, without relationship with the other ones. This style of specification facilitates the Static Interoperability Review. For the definition of target interoperability tests, it is enough to compare the client and the server ICS and select the intersect of the implemented options. We used the guide of WAP Implementation Conformance Statement [28] as a basis to test selection. Optional services are mainly the `Push` services and the `Method Invocation` facilities. During the SIR one can observe that most of the optional services can be invoked only after the connection establishment (the `Suspend/Resume` service is a particular case which can be started before). Instead of creating one test case per option, one could imagine to compute all these optional behaviors at a time, after the connection establishment. But in practice, implementations have not the same ICS and it is reasonable to compute optional services separately.

Even if test patterns are produced automatically with our tool, test purposes are to be defined formally before, and this work is done by hand. In this paper, we will not include all the test purposes considered. We will present two examples to illustrate the experiment with the WSP layer. The first example is concerned with a mandatory service (the session creation, the session release). The second example (the `Push` service) is optional. In addition, we include the `Suspend/Resume` test case which is much longer. The specifications represent 3000 lines of code and cannot be included in this paper. However, in order to help the reader follow the test generation results, we have supplied in the appendix section a portion of the specification model, including only the transitions involved in the connection phase: the beginning of specifications (the syntax of transitions is described in the following section).

5 Compliance/Interoperability test suites for WSP

5.1 Session Creation and Release

Test method: The session creation starts with an `S-Connect-req` (Connection Request), and ends with an `S-Connect-cnf` (Connection confirmation). The session release starts with an `S-Disconnect-req` initiated by either the client or the server (in this example, it is the server) and ends with the notification of the disconnection. Remember that the test purpose is an abstract definition of the expected behavior. It can be modeled with a finite automaton, presented here as a sequence of transitions (one line per transition). The syntax follows: Each line/transition describes an interaction as well as the implementation which executes it. In the example below, "1" represents the identifier of the WAP-client implementation and "2" represents the identifier of the WAP-server implementation. Let us consider the first line/transition of the example: `TP1` is the initial state and `TP2` is the destination state of the transition. "?" means that the interaction is received and `1.S-Connect-req` means that the interaction is the execution of `S-Connect-req` by the WAP client (identifier "1").

Test Purpose: Identifiers `WSP_CO_C001,WSP_CO_C002` of the ICS document.

```
TP1 ? 1.S-Connect-req 1 TP2
TP2 ? 2.S-Connect-res 2 TP3
TP3 ! 1.S-Connect-cnf 1 TP4
TP4 ? 2.S-Disconnect-req 2 TP5
TP5 ! 1.S-Disconnect-ind(DISCONNECT) 1 TP6
TP6 ! 2.S-Disconnect-ind(USERREQ) 2 TP7
```

The TESCOMS tool computes the client and server WAP-specifications according to the test purpose. The tool produces an output which is a test pattern to be used for experimenting the expected feature (in this example, the connection establishment and the connection release). The output of the tool has the following syntax: The lines of the form `[XXX XXX XXX]` concern internal identifiers related to the global state which may be actually reached by the distributed system: In the first line example, `NULL` represents the starting state of the client, the second `NULL` represents the starting state of the server, and `TP1` is the beginning of the behavior to be tested. Between the global states, the interactions to be executed are displayed (with tabulations).

Test case (dynamic behavior):

```
[NULL NULL TP1]
 1:    ? 1.S-Connect-req 1
[NUCIA1 NULL TP2]
 2:     ! TR-Invoke(Connect) 1
[CONNECTING NUCIA1 TP2]
 3:      ! TR-Invoke.ack 2
[CONNECTING NUCIA2 TP2]
 4:       ! 2.S-Connect-ind 2
[CONNECTING CONNECTING TP2]
 5:        ? 2.S-Connect-res 2
[CONNECTING CIC2A1 TP3]
 6:         ! TR-Result(ConnectReply) 2
[CICEA1 CONNECTING-2 TP3]
 7:          ! TR-Result.ack 1
[CICEA2 CONNECTED TP3]
```

```
 8:           ! 1.S-Connect-cnf 1
[CONNECTED CONNECTED TP4]
 9:            ? 2.S-Disconnect-req 2
[CONNECTED CENUA1 TP5]
10:              ! TR-Invoke(Disconnect) 2
[CENUC1 CENUA2 TP5]
11:               ! 1.S-Disconnect-ind(DISCONNECT) 1
[NULL CENUA2 TP6]
12:                ! 2.S-Disconnect-ind(USERREQ) 2
[NULL NULL TP7]
```

Test case analysis:

`1:` A Connection request is received by implementation `1` (the client session entity).

`2:` The client invokes the transaction layer for sending its `Connect` PDU.

`3:` Upon reception of that PDU, the server (session entity) acknowledges it through the transaction layer.

`4:` The server sends a Connection indication to the upper layers.

`5:` Then it receives the response notifying that the application accepted the connection.

`6:` This notification is sent to the client, as a result of its `Connect` request.

`7:` The client acknowledges the result.

`8:` The client session user is sent a confirmation notifying that the connection has been established.

`9:` The server side receives a Disconnection request.

`10:` Then the request is encapsulated by the transaction layer.

`11,12:` The upper layers are notified that the session has been released.

5.2 Sample `Push` interaction

The `Push` Service offers the possibility to a server to spontaneously send some information to a client. As this feature is optional, the client must have a subscription to the related service before. Then information can be automatically received without explicit request.
Test method: The basic service primitives to be used are `S-Push-req` and `S-Push-ind`. Note that only the server can initiate a `Push` request, this primitive is not present in a WAP client specification. We define the following test purpose for characterizing the `Push` Service.
Test Purpose: Identifier `WSP_CO_C009` of the ICS document.

```
TP1 ? 1.S-Connect-req 1 TP2
TP2 ? 2.S-Connect-res 2 TP3
TP3 ! 1.S-Connect-cnf 1 TP4
TP4 ? 2.S-Push-req 2 TP5
TP5 ! 1.S-Push-ind 1 TP6
```

The following test case is obtained:
Test case (dynamic behavior):

```
[NULL NULL TP1]
 1:    ? 1.S-Connect-req 1
[NUCIA1 NULL TP2]
 2:     ! TR-Invoke(Connect) 1
[CONNECTING NUCIA1 TP2]
 3:      ! TR-Invoke.ack 2
[CONNECTING NUCIA2 TP2]
 4:       ! 2.S-Connect-ind 2
[CONNECTING CONNECTING TP2]
 5:        ? 2.S-Connect-res 2
[CONNECTING CIC2A1 TP3]
 6:         ! TR-Result(ConnectReply) 2
[CICEA1 CONNECTING-2 TP3]
 7:          ! TR-Result.ack 1
[CICEA2 CONNECTED TP3]
 8:           ! 1.S-Connect-cnf 1
[CONNECTED CONNECTED TP4]
 9:            ? 2.S-Push-req 2
[CONNECTED CECEA1 TP5]
10:             ! TR-Invoke(Push) 2
[CECEA1 CONNECTED TP5]
11:              ! 1.S-Push-ind 1
[CONNECTED CONNECTED TP6]
```

Test case analysis:

`[1-8]`: Connection establishment

`9, 11`: The `Push` interaction is started after the connection has been established (this protocol works in connection-mode).

`10`: The interaction is performed through the transaction service.

The previous `Push` interaction is a non-confirmed service. The confirmed `Push` is also available in the service specification.

5.3 The `Suspend/Resume` service

During the communication between a client and a server, some unpredictable event may occur, involving either the client or the server to be temporarily busy (or disturbed somehow). In such case, instead of closing the connection, and then asking for another new connection, the `Suspend/Resume` service can be invoked.

The entity which asks for suspending the session uses an `S-Suspend-req`. The session can be resumed with the `S-Resume-req`. The test case below shows a scenario which can be used to experiment this service.

Test case (dynamic behavior):

```
[NULL NULL TP1]
 1:    ? 1.S-Connect-req 1
[NUCIA1 NULL TP2]
 2:     ! TR-Invoke(Connect) 1
[CONNECTING NUCIA1 TP2]
 3:      ! TR-Invoke.ack 2
[CONNECTING NUCIA2 TP2]
 4:       ! 2.S-Connect-ind 2
[CONNECTING CONNECTING TP3]
 5:        ? 2.S-Connect-res 2
[CONNECTING CIC2A1 TP4]
 6:         ! TR-Result(ConnectReply) 2
[CICEA1 CONNECTING-2 TP4]
 7:          ! TR-Result.ack 1
[CICEA2 CONNECTED TP4]
 8:           ! 1.S-Connect-cnf 1
[CONNECTED CONNECTED TP5]
 9:            ? 1.S-Suspend-req 1
[CESEA1 CONNECTED TP6]
10:             ! TR-Invoke(Suspend) 1
[CESEA2 CESEA1 TP6]
11:              ! 2.S-Suspend-ind(SUSPEND) 2
[CESEA2 SUSPENDED TP7]
12:               ! 1.S-Suspend-ind(USERREQ) 1
[SUSPENDED SUSPENDED TP7]
13:                ? 1.S-Resume-req 1
[SERIA1 SUSPENDED TP8]
14:                 ! TR-Invoke(Resume) 1
[RESUMING SERIA1 TP8]
15:                  ! TR-Invoke.ack 2
[RESUMING SERIA2 TP8]
16:                   ! 2.S-Resume-ind 2
[RESUMING RESUMING TP9]
17:                    ? 2.S-Resume-res 2
[RESUMING RIR2A1 TP10]
18:                     ! TR-Result(Reply) 2
[RINUCEA1 RESUMING-2 TP10]
19:                      ! TR-Result.ack 1
[RINUCEA2 CONNECTED TP10]
20:                       ! 1.S-Resume-cnf 1
[CONNECTED CONNECTED TP11]
21:                        ? 2.S-Disconnect-req 2
[CONNECTED CENUA1 TP12]
22:                         ! TR-Invoke(Disconnect) 2
[CENUC1 CENUA2 TP12]
23:                          ! 1.S-Disconnect-ind(DISCONNECT) 1
[NULL CENUA2 TP13]
24:                           ! 2.S-Disconnect-ind(USERREQ) 2
[NULL NULL TP14]
```

The full execution of such test cases by WAP client and server entities show their aptitude to interoperate or not. The occurrence of an unexpected interaction reveals an error.

5.4 Analysis of the produced test suite

Function	PDU/Capabilities	Mandatory Feature	Tests computed
Session Creation	Connect PDU	Yes	Yes
Capabilities Negotiation	Connect PDU, Connect Reply PDU	Yes	No
Session Release	Disconnect PDU	Yes	Yes
Session Suspend, Session Resume	Suspend PDU, Resume PDU	No	Yes
Push	Push PDU	No	Yes
Confirmed Push	ConfirmedPush PDU	Yes	Yes
Extended Methods	Proprietary Methods	No	No
Methods GET, POST	Get PDU, Post PDU, Reply PDU	No	Yes
Methods DELETE, HEAD, OPTION, TRACE	Get PDU, Reply PDU	No	Yes
Method PUT	Post PDU, Reply PDU	No	Yes

Table 1: Analysis of dynamic behaviour computed

Table 1 and Table 2 recap the features covered by the computed WSP test suite. In Table 1, the column "Function" represents service functionalities. For each function, we present the PDUs involved in the operation of the function, or the related capabilities. Some of the functionalities specified in the standard are mandatory while the other ones are optional. Most of the mandatory features have been computed (80%). Capabilities represent a set of service facilities and parameter settings related to the operation of the service provider. There are different kinds of facilities defined in the standard, but the service provider may recognise additional facilities. Some examples of facilities are *Aliases, Extended methods, Protocol options*. The basic methods of WAP correspond to the ones defined in the HTTP/1.1 standard, and the *Extended methods* correspond to those that are beyond HTTP/1.1. *Proprietary methods* fall in this category. During the connection, the peer entities can perform a *Capabilities negotiation* in order to agree on a common communication profile. The `Connect PDU` can be used to specify the requested capabilities of the initiator, and the `ConnectReply PDU` would contain the capabilities acknowledged by the responder. Since all the actual capabilities are not provided in the current standard, and as these capabilities depend on the actual WSP service users, we did not look into this aspect. For the time being, we have treated the connection procedure (session creation PDUs) that conveys user defined facilities. 83% of the dynamic optional functions have been treated. 100% of the standard defined PDUs and also 100% of the standard defined ASPs are involved in the test suite. This guarantees that each elementary protocol operation can be experimented by the computed test suite.

Mandatory functions	Optional functions	PDU involved	ASP involved
80%	83%	100%	100%

Table 2: Percentage of computed features

In the overall test suite, there are also some parts of test cases that are repeated. For instance, all the `Method Invocation` facilities have basically the same operation scheme. At the WSP level, methods can be invoked by three basic PDUs, namely the `Get PDU, Post PDU` and the `Reply PDU`. The current `Method` reference is encapsulated in these PDUs.

6 Conclusions

In this paper, we have proposed an interoperability testing approach experimented against wireless application protocols. The approach is based on automatic test computation, which is a good way of insuring test soundness and test reproductibility. Our test computation tool is based on an on-the-fly technique (adapted here to concurrent specifications), which is a good way of tackling specifications' complexity.

According to the WAP forum framework, the development of WAP systems goes through application-level testing and protocol-level testing. WAP forum efforts have been focused on application level and actually, certification tests as well as some WAP-certified products including WML-browsers are now available (check the WAP forum web site [27]).

Our work is a contribution to testing methodology of protocol interoperations against WAP standards. Of course, for space reasons, we could not include in this paper all the test cases designed. We illustrated our work with WSP, which is the first layer below application level. Almost all the mandatory functionalities have been experimented in dynamic test generation. We did not complete the aspects related to capability negotiation, which are rather a concern of the actual telecommunication service provider. Furthermore, the number of possible facilities is unknown a priori. A preliminary static interoperability review of the actual WAP-applications communicating through the WSP service may allow for the selection of a subset of relevant facilities to be experimented.

As our testing approach is suitable for layered protocols, we can experiment it with the other WAP lower layers with few limitations. For the time being, we have been interested in concrete experimentation/deployment of the produced test cases. Most of the telecommunications operators have already done some developments related to the WAP technology, and we have obtained, for free, a prototype of a WAP client, and a WAP server (with full source code in C language). These implementations allow us to conduct concrete test experimentations. Our test platform is currently operational with the ORBIX [34] Object Request Broker, which environment already includes C++ and JAVA compilers.

References

1. A. Aho, J. Hopcroft and J. Ullman. Data Structures and Algorithms, Addison-Wesley, 1983.
2. X.Balliet, N.Eloy. *Development of a distributed test system with CORBA*. Co-authored Master's thesis. Institut National Polytechnique, Nancy France, March 2000.
3. A.Belinfante, J.Feenstra,T.G.Vries, J.Tretmans, N.Goca, L.Feijs, S.Mauw, L.Heerink. Formal test automation: A simple experiment. In. Testing of Communicating Systems. pages 179-196. Kluwer Academic, 1999.
4. A.Cavalli, L.P.Lima. A pragmatic approach to generating test sequences for embedded systems. In. Testing of Communicating Systems Vol. 10. Chapman & Hall, 1997.
5. R.Castanet, O.Koné, P.Laurençot. On the fly test generation for real time protocols. Proc. International Conference on Computer Communication and Networks. Louisianne, USA. October 1998.
6. S.K. Das, R. Jayaram, N.K. Kakani and Sanjoy K. Sen. A call admission and control scheme for quality-of-service (QoS) provisioning in next generation wireless networks. Wireless Networks. Baltzer Science Publishers Vol. 6 (2), 2000. 17-30.
7. Fernandez J. C., Jard C., Jéron T., Viho C. Using on-the-fly verification techniques for the generation of test suites. In CAV'96. LNCS 1102, Springer Verlag, 1996.
8. FRAMES Workshop. Resource management for UMTS wireless access. Delft, The Netherlands, January 1999.
9. A.Fukada et al. A conformance testing for communication protocols modeled as a set of DFSMs with common inputs. In. Testing of Communicating Systems Vol. 10. Chapman & Hall, 1997.

10. Grabowski J. Test case generation and test case specification with Message Sequence Charts PhD thesis. University of Berne, February, 1994.
11. Hypertext Transfer Protocol. HTTP/1.1, RFC2068. R.Fielding et al. January, 1997.
12. HTML 4.0 Specification, W3C Recommendation REC-HTML40-971218. D.Raggett et al. September 1997.
13. T.D. Hodes, R. H. Katz. Composable ad hoc location-based services for heterogeneous mobile clients. Wireless Networks. BALTZER SCIENCE PUBLISHERS, Vol. 5 (5), 1999. 411-427.
14. ITU-T Recommandation Q.2110: B-ISDN ATM Adaptation Layer - ITU Telecommunication Standard Sector 1994.
15. ISO/IEC 9646, Information Technology – Open Systems Interconnection-Conformance testing methodology and framework - Part 1-5. 1991.
16. ISO/TC 97/SC 16/WG1 IS 7498. Basic Reference Model for Open System Interconnection. 1983.
17. Information Processing Systems. Open Systems Interconnection. Transport protocol International Standard ISO 8073, 1988 (F)-AFNOR.
18. ITU-T Recommandation Z.100: Specification and Description Language SDL, Contribution Com X-R215-E, 1987.
19. Juntong Liu and Gerald Q. Maguire Jr. GMRM: An Efficient Routing Model for an Integrated Wireless Mobile Packet Switch Network. The 3rd Workshop on Personal Wireless Communication (PWC98), Tokyo, Japan, 1998.
20. O.Koné. *A local approach to the testing of real-time systems.* The Computer Journal, Volume 44. British Computer Society, Oxford Press. 2001.
21. O. Koné, R. Castanet. Test generation for interworking systems In. *Computer Communications*, Elsevier Science Publishers, Vol. 23 N.7 Mars 2000. pp 642-652.
22. O.Koné, R.Castanet. Formal Methods in Protocol Conformance Testing. *TSI journal* Technique et Science Informatiques N.5/1999, Editions HERMES Paris.
23. O.Koné. *Compliance of wireless application protocols.* IFIP TestCom International Conference - Testing Internet Technologies and Services. Kluwer Academic Publishers, 2002.
24. O.Koné, R.Castanet. *The* TBROKER *platform for the interoperability of communications systems.* IEEE 5^{th} CSCC World Conference. Crete, July 2001.
25. O. Koné. J.P. Thomesse. Design of interoperability checking sequences against WAP. Proc. IFIP International Conference on Personal Wireless Communications. Kluwer Academic Publishers, September 2000.
26. L.Kaiser, O.Koné. Verification method of interoperability for real time systems. 4^th IFAC International Symposium on Intelligent Components and Instruments for Control Applications, Buenos Aires, Septembre 2000.
27. Wireless Application Protocol. Architecture Specification, WAP forum, April, 1998. URL: `http://www.wapforum.com`
28. Wireless Application Protocol. Conformance Statement, Compliance Profile and Release List. WAP forum, April 1998. URL: `http://www.wapforum.com`.
29. Wireless Session Protocol Specification. WAP forum, URL: `http://www.wapforum.com`, 1999
30. L'Internet du futur. RNRT Report. French Ministry of Education. URL: `http://telecom.gouv.fr/RNRT`, 2000
31. D.Lee, K.Sabnani, D.Kristol, S.Paul. Conformance testing of protocols specified as communicating FSMs. In. Proc. IEEE INFOCOM'93, San Francisco, 1993.
32. D.Lee, D.Su. Modeling and testing of protocol systems. In. Testing of Communicating Systems Vol. 10. Chapman & Hall, 1997.
33. G.Luo, G.v.Bochmann, A.Petrenko. Test selection based on communicating non deterministic machines using a generalized Wp-method. IEEE Transactions on Software Engineering, SE-20(2):149-162, 1994

34. Orbix 2000. IONA Technologies.
URL: `http://www.orbix.com`
35. O.Rafiq, L.Cacciari. Controllability and Observability in distributed testing. Information and Software Technology, Elsevier, Vol. 41 (1999) 767-780.
36. K.Sabnani, A.Dahbura. A protocol test generation procedure. Computer Networks and ISDN Systems 15, 1988 (pp 285-297).
37. The ATM Forum. URL: `http://www.atmforum.com`
38. J.Tretmans. A Formal Approach on Conformance Testing. PhD thesis, University of Twente, the Netherlands, 1992.
39. A.Ulrich, S.T.Chanson. An approach to testing distributed software systems. In. Proc. IFIP symposium on Protocol Specification Verification and Testing, Warsaw, Poland, June 1995.
40. T. Whalen and J.P. Black. Adaptative Groupware for Wireless Networks. 2nd IEEE Workshop on Mobile Computing Systems and Applications. New Orleans, Louisiana, Feb. 1999.
41. A.L. Wijesinha, S.P. Kumar and D.P. Sidhu. Handover and new call blocking performance with dynamic single-channel assignment in linear cellular arrays. Wireless Networks. Baltzer Science Publishers, Vol. 6 (2), 2000. 121-129.

Appendix

Client side (subset of full specification)

```
NULL ? 1.S-Connect-req 1 NUCIA1
NUCIA1 ! TR-Invoke(Connect) 1 CONNECTING
CONNECTING ? TR-Invoke.ack 1 CONNECTING
CINUA1 ! TR-Abort 1 CINUA2
CINUA2 ! 1.S-Disconnect-ind(USERREQ) 1 NULL
CONNECTING ? Disconnect 1 CINUB1
CINUB1 ! TR-Abort 1 CINUB2
CINUB2 ! 1.S-Disconnect-ind(DISCONNECT) 1 NULL
CONNECTING ? Suspend 1 CINUC1
CINUC1 ! TR-Abort 1 CINUC2
CINUC2 ! 1.S-Disconnect-ind(SUSPEND) 1 NULL
CONNECTING ? TR-Result(ConnectReply) 1 CICEA1
CICEA1 ! TR-Result.ack 1 CICEA2
CICEA2 ! 1.S-Connect-cnf 1 CONNECTED
...
```

Server side (subset of full specification)

```
NULL ? TR-Invoke(Connect) 2 NUCIA1
NUCIA1 ! TR-Invoke.ack 2 NUCIA2
NUCIA2 ! 2.S-Connect-ind 2 CONNECTING
CONNECTING ? 2.S-Connect-res 2 CIC2A1
CIC2A1 ! TR-Result(ConnectReply) 2 CONNECTING-2
```

```
CONNECTING ? 2.S-Disconnect-req 2 CITIA1
CITIA1 ! TR-Result(Redirect) 2 CITIA21
CITIA21 ! 2.S-Disconnect-ind(USERREQ) 2 TERMINATING
CITIA1 ! TR-Result(Reply) 2 CITIA22
CITIA22 ! 2.S-Disconnect-ind(USERREQ) 2 TERMINATING
CONNECTING ? Disconnect 2 CINUA1
CINUA1 ! TR-Abort 2 CINUA2
CINUA2 ! 2.S-Disconnect-ind(DISCONNECT) 2 NULL
CONNECTING ? Suspend 2 CINUB1
CINUB1 ! TR-Abort 2 CINUB2
CINUB2 ! 2.S-Disconnect-ind(SUSPEND) 2 NULL
CONNECTING ? TR-Invoke(Resume) 2 CICIA1
CICIA1 ! TR-Abort 2 CONNECTING
CONNECTING ? TR-Abort 2 CINUC1
CINUC1 ! 2.S-Disconnect-ind(abortreason) 2 NULL
TERMINATING ? Disconnect 2 TINUA1
TINUA1 ! TR-Abort 2 NULL
TERMINATING ? Suspend 2 TINUB1
TINUB1 ! TR-Abort 2 NULL
TERMINATING ? TR-Result.ack 2 NULL
TERMINATING ? TR-Abort 2 NULL
C2NUA1 ! TR-Abort 2 C2NUA2
C2NUA2 ! TR-Invoke(Disconnect) 2 C2NUA3
C2NUA3 ! 2.S-Disconnect-ind(USERREQ) 2 NULL
CONNECTING-2 ? Disconnect 2 C2NUB1
C2NUB1 ! TR-Abort 2 C2NUB2
C2NUB2 ! 2.S-Disconnect-ind(DISCONNECT) 2 NULL
CONNECTING-2 ? Suspend 2 C2NUSEA1
C2NUSEA1 ! TR-Abort 2 C2NUSEA2
C2NUSEA2 ! 2.S-Disconnect-ind(SUSPEND) 2 NULL
C2NUSEA2 ! 2.S-Suspend-ind(SUSPEND) 2 SUSPENDED
CONNECTING-2 ? TR-Invoke(Resume) 2 C2RIA1
C2RIA1 ! TR-Abort 2 CONNECTING-2
C2RIA1 ! TR-Invoke.ack 2 C2RIA2
C2RIA2 ! TR-Abort 2 C2RIA3
C2RIA3 ! 2.S-Suspend-ind(RESUME) 2 C2RIA4
C2RIA4 ! 2.S-Resume-ind 2 RESUMING
CONNECTING-2 ? TR-Invoke(Disconnect) 2 C2NUC1
C2NUC1 ! TR-Abort 2 C2NUC2
C2NUC2 ! 2.S-Disconnect-ind(DISCONNECT) 2 NULL
CONNECTING-2 ? TR-Invoke(Suspend) 2 C2SEA1
```

```
C2SEA1 ! TR-Abort 2 C2SEA2
C2SEA2 ! 2.S-Suspend-ind(SUSPEND) 2 SUSPENDED
CONNECTING-2 ? TR-Result.ack 2 CONNECTED
...
```