Network Protocol Interoperability Testing based on Contextual Signatures and Passive Testing *

Fatiha Zaidi [1,2]
[1]Univ. Paris-Sud, LRI,Orsay, F-91405;
[2]CNRS, Orsay,F-91405
Fatiha.Zaidi@lri.fr

Emmanuel Bayse
Stéria Télécom
F-97782 Issy-les-Moulineaux
Emmanuel.Bayse@steria.fr

Ana Cavalli
Telecom Sud-Paris -CNRS SAMOVAR
F-91011 Evry Cedex
Ana.Cavalli@int-evry.fr

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing Tools*; I.6 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design, Reliability, Verification

Keywords

Interoperability testing, contextual signatures, passive testing, communication protocols, WAP

ABSTRACT

This paper presents a methodology for interoperability testing based on contextual signatures and passive testing with invariants. The concept of contextual signature offers a framework to add information on the states, the values of parameters, as well as logical connectors that increases the expressive power of invariants. This allows expressing horizontal and vertical interoperability properties, i.e., between layers of a protocol stack or end-to-end communication between distant entities. In order to test interoperability, we have defined a correlation algorithm between the events collected from different network views (client or network side). Once the correlation has been performed, we apply the contextual signatures that characterize interoperability properties to check their validity. To illustrate the application of the proposed approach, a real case study is proposed: the Wireless Application (WAP) protocol. The results of the experimentation performed on this protocol are also presented.

1. INTRODUCTION

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

Future communication networks, in particular those based on Internet, lead to the emergence of new families of protocols and services for telecommunication systems. These systems will have, as infrastructure, heterogeneous networks which need to be interconnected and to interoperate while preserving the required quality of service. They will be confronted with interoperability problems between protocol variants and services, including vertical and horizontal protocol interoperability and end-to-end interoperability. This requirement is of a crucial importance to obtain a successful integration of communication components in telecommunication systems.

In this context, conformance testing alone is not sufficient to guarantee successful communication between different components. The reason is that this kind of test is only devoted to checking the behaviour of a component in relation to its specification. The ISO 9646 standard [1] mentions that conformance testing can increase the probability of interoperability, but cannot guarantee the successful interaction between different implementations. It must be noted that conformance testing is a necessary step to insure interoperability but it is not enough. It could still be possible that communication components do not communicate correctly even if each conforms to the same standard.

Until nowadays, the majority of research works on interoperability testing are based on active testing that implies interferences with the implementation under test. The main objectives of the work presented in this paper is to propose a methodology for interoperability testing based on passive testing, which is innovative and original. To the best of our knowledge, no work based on passive testing techniques has been applied to perform interoperability testing. Furthermore, the originality of our approach is based on the absence of interference with the implementation under test, which is an advantage when it is necessary to test a system with no direct access to their components. The case study presented here is a good illustration: to perform vertical interoperability, i.e between protocols layers, no direct access to the embedded layer is given; also for end-to-end interoperability it is not always possible by using active testing techniques to have a global view of the communication between the entities of the system. In our case, we can observe what occurs at each entity side and to obtain a global view through the correlated traces. To illustrate the suitability of our approach, a real case study is presented: the Wireless Application Protocol (WAP). To perform the experiments, a free software protocol stack has been installed, namely Kannel (version2) [10].

The paper is organized as follows. In section 2, related works to the context of the research are presented. Section 3 gives the definition of interoperability testing, interoperability testing architecture and an outline of the proposed methodology. This section also includes the definition of invariants and contextual signatures that are

used to describe interoperability properties. Section 3.6 presents the correlation algorithm to produce end-to-end traces. The application of the proposed approach and the experimental results are exhibited in section 5. Finally, section 6 gives the conclusions and perspectives of this work.

2. RELATED WORKS

Many works address the problems of interoperability testing. There are mainly two kinds of approaches, those that are based on a formal framework and those based on experimental results. The authors of [16] have proposed a solution for interoperability test generation. Their work is based on the generation of the reachability graph of the communicating system. Koné and Castanet in [12] have formalized the relation of interoperability by adapting the existing conformance relations. [9] proposed algorithms to generate a test suite that are based on stable states of their composition algorithm. Previous works can be classified as methods based on reachability analysis, that often suffer from the well-known state space explosion problem when applied to realistic or real case studies. Methods based on experimentations have also been proposed, such as [18] but they lack a formal framework.

[8] based their work on interoperability by proposing a test generation technique from the joint behaviour of end-users interfaces. This method avoids the combinatorial explosion problem by classifying the edges concerning interoperation as black edges and those concerning local activities as white edges. They propose an efficient algorithm to retrieve the significant transitions, in other words the black edges. [4] proposed to consider transitions on a communicating graph representing interactions between distant entities and they have developed two algorithms to cover such transitions. All these works are based on an active testing approach, where the tester has the ability to stimulate the implementation under test (IUT) and verify whether the output obtained for each input is according to the specification. When the tester is not provided with a direct interface to interact with the IUT, passive testing is better adapted than active testing for performing conformance and interoperability testing. For these reasons we consider here the use of passive testing techniques. Even though passive testing techniques are not new (see for example the approach shown in [2]) in the last years a very active research on passive testing has been developed.

The passive testing approach proposed here is based on invariant analysis, in which the invariants are properties the IUT is expected to satisfy. It is inspired by a previous work [6], which concerned mainly conformance testing [3]. The difference with the work presented here relies on the introduction of contextual signatures that permit to express interoperability properties. This was not possible by using only conformance requirements. Moreover, the testing approach proposed in [3] is based on local observations, while in the work presented in this paper a correlation mechanism is provided to obtain a global trace that is needed to test end-to-end properties.

The concept of contextual signatures describes, in a structural way, a set of properties that involves different communication elements. The concept of signature is inspired from the system Bro [15] that is a Network Intrusion Detection System (NIDS). In our case, contextual signatures are extended to provide information such as dependence between invariants, constraints on the variables and, also, the notion of state. Moreover, this concept provides a structure to specify the requirements related to the interoperability between different protocols implementations.

3. PRELIMINARIES

3.1 Interoperability definition

In the following, we provide the definition of vertical and horizontal interoperability that we consider:

Definition 1. *Horizontal interoperability* layers of the same level of distant entities are interoperable if they successfully pass the conformance tests and if it can been shown that together they satisfy the level N service as specified by the standard [7].

Definition 2. *The multi-protocols or vertical interoperability [14]* focuses on the interoperability between layers of the same protocol stack. Performing this test is important in order to warranty that the protocol layer will correctly interoperate with the other layers of the protocol stack.

3.2 Interoperability testing architecture

The testing architecture is based on a more accurate level of observation in order to establish a diagnostic regarding the implementation behaviors. This is the reason the architecture relies on the set up of Points of Observation (PO). It is necessary to observe the messages exchanged by the distant entities. A sniffer is used to collect the traffic carried by the communication medium. For the communication protocols area, there are several sniffers available, such as *Ethereal*, *Wireshark* [19] (new release of Ethereal) and *Snort* [17]. We install points of observation at the interface levels of the protocol layer to be tested. These local observers are needed to perform vertical interoperability between protocol layers at different levels. For horizontal interoperability, the sniffer is used to collect the protocol messages that are exchanged between distant entities. In the testing architecture used, we have local observers that collect the messages at the protocol layer interface. Furthermore, the architecture relies on a global observer that reconstructs a global trace from the trace retrieved by the local observers and from the PDU exchanged at the communication level. In this work, we consider the horizontal interoperability between two entities.

To achieve the construction of the global trace, i.e. an end-to-end trace, we correlate the events that are collected by the different observers. The correlation is feasible because of the specific nature of communicating protocols. We assume that we have all the required information on Service Data Units (SDU) and also the corresponding PDU. Once a global trace is reconstructed, we can check an end-to-end property using the concept of contextual signatures.

3.3 Outline of the proposed methodology

As mentioned previously, to perform interoperability testing, end-to-end traces have to be constructed in order to verify the expected interoperability properties. To achieve this, the following steps need to be performed:

Step 1: Properties formulation. The properties are retrieved from the standards and expressed the main requirements related to the communication with distant entities.

Step 2: Properties as contextual signatures. Properties have to be formulated by means of contextual signatures. Contextual signatures have to be conform to their Data Type Definition (DTD). The verification is done automatically by a parser. These signatures enclosed *simple* or *obligation* invariants that expressed local properties; i.e. related to a local entity. To represent interoperability properties, the contextual signatures are used to connect the invariants by logic-connectors. All the information presented in the section 3.5 are also furnished.

Step 3: Extraction of execution traces from local observers by

means of a sniffer installed on each side entity. The traces are in XML format.

Step 4: Transformation of traces in an adapted format. Traces are transformed by the application of filtering rules defined by a XSL sheet. These rules hold the network information needed to perform the checking of the properties described by the contextual signatures. The XSL expressions constructed to filter information, or perform data formatting, can be easily adapted to different types of trace sources.

Step 5: Syntax checking. Once the traces are filtered, they are checked through a DTD (Data Type Definition) procedure that validates the structure of the file containing the filtered network traces.

Step 6: Traces correlation. Finally, the validated traces are correlated. The algorithm for the correlation of the traces is described in detail in the section 3.6.

Step 7: Verification of the contextual signatures on the traces. Once an end-to-end trace is obtained by applying the correlation algorithm, the verification of the interoperability properties is performed and a verdict is emitted (Pass, Fail or Inconclusive). This last step is based on the application of pattern matching algorithms as mentioned in subsection 3.4. These algorithms have been adapted to check contextual signatures.

3.4 Invariants definitions

Two kinds of invariants are defined: the *simple* and *obligation* invariants. The *simple invariant* can be defined as follows:
Let $M = (S, \mathcal{I}, s_{in}, \mathcal{O}, Tr)$ be an `FSM`. An `FSM` is defined by a finite set of states, a set of input actions, a set of output actions, an initial state s_{in} and a set of transitions Tr. Each transition $t \in Tr$ is a tuple $t = (s, s', i, o)$. Intuitively, a transition $t = (s, s', i, o)$ indicates that if the machine is in state s and receives the input i then the machine emits output o and moves to the state s'.
Intuitively, a trace such as $i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n/O$ is a *simple invariant* for M if each time that the trace $i_1/o_1, \ldots, i_{n-1}/o_{n-1}$ is observed, if we obtain the input i_n then we necessarily get an output belonging to O, where $O \subseteq \mathcal{O}$. In addition to sequences of input and output symbols, the wild-card characters ? and * are allowed, where the wild-card character ? represents any single symbol and the * represents any sequence of symbols. This notion of invariant allows us to express several interesting properties. For example, by using the invariant, we can test that an user is disconnected each time he requests a disconnection.

$$I_1 = req\_disconnect/\{disconnected\}$$

The idea is that each occurrence of the symbol $req\_disconnect$ is followed by the output symbol $disconnected$. An invariant that expresses a property as "if y happens then we must have that x had happened before" is called an *obligation invariant*. *Obligation invariants* may be used to express properties where the occurrence of an event must be necessarily preceded by a sequence of events. For example, the intuitive meaning of an invariant such as

$$I = request\_page/req\_ack, *, ?/\overline{\{page\_sent\}}$$

is that if the event $page\_sent$ is observed in the trace then we must have that a page had been requested before and that the server has acknowledged the reception of the request.

Invariants can be checked on the specification and on the implementation. For the specification, in [3] we have developed algorithms which are better, with respect to complexity, than classical algorithms for model checking. These algorithms are usually exponential on the number of transitions. In our case, the proposed algorithm possesses, in the worst case, a complexity that is linear with respect to the number of transitions in the specification if the invariant does not contain the wild character *. The complexity is quadratic if the symbol * appears in the invariant. For the verification of the implementation we have developed a new algorithm that checks the properties on the real implementations traces. This algorithm is based on classical algorithms for pattern matching on strings (see e.g. [11] and [5]). It possesses a complexity $\mathcal{O}(m.n)$, in the worst case, where m and n are, respectively, the length of the trace and of the invariant. Let us remark that in most practical cases the length of the invariant is several orders of magnitude smaller than the length of the trace. Thus, we may consider that the previous complexity is almost linear with respect to the length of the trace.

3.5 Concept of contextual signatures

Contextual signatures are defined by a XML (eXtensible Markup Language) tree. This representation has been chosen because it contributes to structure the signatures and to define the syntax of the data. XML also offers means to check the syntax of a document through a DTD (Data Type Definition). The validity of a document (in our case a signature) can be checked with a parser. The DTD can be changed accordingly to the protocol that we consider, we can change the list of possible values for some attributes.

We will not go through all the details of the contextual signatures. We will only present here the key elements of this new structure. Nevertheless, we can mention that several elements defined by the `info-proto` element allow adding information on the protocol that is being considered. The invariant definition presented above is enclosed in the `content` element which characterizes the property to be tested. The `content` element is described by two attributes: one that indicates the type of invariant, i.e. *simple* or *obligation*, and another that formally describes the property using regular expressions (see section 3.4).

The contextual signatures have also a `context` element that integrates the implementation's contextual information. In particular, we add state information. This knowledge will help analyze the trace for *obligation* invariants. Indeed, with this element we can set the state from which the invariant will be checked. The attribute `current-state` indicates the state that the implementation needs to go through in order to check the invariant, making it a kind of pre-condition to be checked.
The `dependency` element is used to define relationship with several signatures. In other words, it allows to specify a property that combines several invariants (i.e. several individual signatures). Hence, the signatures can be connected by logical connectors. The structure of this element is defined by several attributes as follows:

- `logic-connector` indicates the link between the current signature and the one specified by the attribute `id-sign-request`. We can use a conjunction (and) or a disjunction (or) to define this link.
- `id-sign-request` identifies the signature that will be related to the current signature.

3.6 Correlation algorithm

This section presents the correlation algorithm. We introduce first the notation used and secondly we give the general idea of the algorithm, which is described in detail in figure 1.

Notation used by the algorithm

The traces collected by the local observers, at the protocol interface levels of a client or server side, are represented as a succession of inputs and outputs. The inputs and outputs represent services primitives (SDU), events representing the PDU reception or emission, or events related to *timers*.

The function *getPacket* allows identifying, from the trace $TRACE_{Network}$, the packet where the PDU transaction corresponds to an event observed in $TRACE_{trans}$, indicating the reception or the emission of this PDU.

The correspondence is established by the name of the message and the parameter TID. The function returns the corresponding packet to be used by the algorithm of the figure 1. If the packet is not retrieved in the $TRACE_{Network}$, then the correlation fails. We consider that for each event observed in $TRACE_{trans}$, which entails the sending of a PDU, we have to observe this PDU in $TRACE_{Network}$. In the same way an event observed in $TRACE_{trans}$, corresponding to the reception of a PDU, has to be observed in $TRACE_{Network}$.

If this is not the case, an error has occurred either caused by the loss of the PDU or by due to an erroneous interoperation behavior between the entities. The events that are used by the algorithm, i.e. inputs and outputs, are associated to their parameters. A transaction trace of an entity is represented formally in the following way:

$TRACE_{trans} = i_1(\overrightarrow{p_{i_1}})/o_1(\overrightarrow{p_{o_1}}), \ldots, i_n(\overrightarrow{p_{i_n}})/o_n(\overrightarrow{p_{o_n}})$ where:

- i_k, o_k are respectively the inputs and outputs such that $1 \leq k \leq n$.
- $\overrightarrow{p_{i_k}} = (p_1, \ldots, p_m)$ is a vector which is characterized by a finite set of variables and the values of the parameters of the input i_k.
  We note $p_j.name$ and $p_j.value$ the corresponding access to the name and value of parameter p_j with $1 \leq j \leq m$. We use the same notation for outputs, i.e. $\overrightarrow{p_{o_k}} = (p_1, \ldots, p_l)$.
  The boolean variable $pck.marked$ in the function below is used to mark the packet that has been already retrieved from the trace. The variable $pck.trans_pdu$ is explained below.

4. ALGORITHM CORRELATION

With respect to the traces captured by the sniffer, these must be processed, as described previously (transformation, validation, formatting). In this way, we obtain a succession of packets, each one integrating the PDU transaction and session exchanged between the two distant entities. The formal representation of a network trace has the following form:

$TRACE_{Network} = \{pck_1, \ldots, pck_u\}$ where pck_i is a packet (such that $1 \leq i \leq u$) which the structure is composed by the couple $trans\_pdu(\overrightarrow{p_{trans}})$ and $session\_pdu(\overrightarrow{p_{session}})$. They characterize, respectively, the PDU transaction and session and their associated parameters; or, when the session layer is not solicited, only the PDU $trans\_pdu(\overrightarrow{p_{trans}})$ and its associated parameters.
$\overrightarrow{p_{trans}} = (p_1, \ldots, p_x)$ and $\overrightarrow{p_{session}} = (p_1, \ldots, p_y)$ are vectors which characterize, respectively, a finite set of parameters variables for the PDU transaction and for the session. One can access the parameters as explained in $\overrightarrow{p_{i_k}}$. By adding session layer information to enrich the expression of our invariants, one can express properties related to services primitives and to PDU. We note $pck_i.trans\_pdu(\overrightarrow{p_{trans}})$ and $pck_i.session\_pdu(\overrightarrow{p_{session}})$ respectively the access to the PDU $trans\_pdu(\overrightarrow{p_{trans}})$ and $session\_pdu(\overrightarrow{p_{session}})$ of the packet pck_i.

Inputs: $TRACE_{trans} = i_1(\overrightarrow{p_{i_1}})/o_1(\overrightarrow{p_{o_1}}), \ldots, i_n(\overrightarrow{p_{i_n}})/o_n(\overrightarrow{p_{o_n}})$
$TRACE_{Network} = pck_1, \ldots, pck_u$
$SET\_PDU = \{...\}$ e.g. for the WAP
$SET\_PDU = \{Invoke, Result, Ack, Abort\}$

Outputs: `false` indicates the failure of the correlation or returns the new built trace $TRACE'_{trans}$.

$TRACE'_{trans} := TRACE_{trans}$;
$k := 0$; $fault\_correlate :=$ `false`;
while $k \leq n$ **and not** $fault\_correlate$ **do begin**
  //case of a correlation with an entry of $TRACE_{trans}$
  if ($i_k \in SET\_PDU$) **then begin**
    //selection of the parameter TID for the entry i_k
    $\{\exists p \in \overrightarrow{p_{i_k}} \mid p.name = trans.TID\}$;
    //selection of the corresponding packet
    $pck := getPacket(i_k, p.value)$;
    if $pck = null$ **then** $fault\_correlate :=$ `true`;
    else begin
      //entry of a couple of // PDU trans and session
      $input\_pdu := (pck.trans\_pdu, pck.session\_pdu)$;
      //We append this entry in $TRACE'_{trans}$ //with respect of the format of //input/ output needed to //apply the invariants.
      $TRACE'_{trans} = \ldots i_{k-1}/o_{k-1}, \mathbf{input\_pdu/i_k}, i_k/o_k, i_{k+1}/o_{k+1} \ldots$
    end
  end

  //case of a correlation with an output //of $TRACE_{trans}$
  if ($o_k \in SET\_PDU$) **then begin**
    //selection of the parameter TID //for the output o_k
    $\{\exists p \in \overrightarrow{p_{o_k}} \mid p.name = trans.TID\}$;
    //selection of corresponding packet
    $pck := getPacket(o_k, p.value)$;
    if $pck = null$ **then** $fault\_correlate :=$ `true`;
    else begin
      //output made of a couple of PDU //trans and session
      $output\_pdu := (pck.trans\_pdu, pck.session\_pdu)$;
      //we append this output in //$TRACE'_{trans}$ with respect of the //format of input/output
    end
  end
  $k := k + 1$;
end
if $fault\_correlate$ **then return** (`false`)
else return ($TRACE'_{trans}$);

Figure 1: Correlation algorithm of protocol events.

General idea of the algorithm

The aim of the algorithm described by the figure 1 is to correlate the events observed by the trace $TRACE_{trans}$ with the PDU observed in the trace $TRACE_{Network}$. This in order to constitute a new trace $TRACE'_{trans}$ which is an end-to-end one. The PDU observed in the network is considered as input or output, depending on the direction of the communication messages and is consequently used in this way by the interoperability invariants. In the following we provide an explanation of the correlation algorithm.

For each input i_k and/or output o_k of the $TRACE_{trans}$ denoted by the external *while* loop, we check that it belongs to the PDU

SET_PDU set specified as an input of the algorithm (inside the first external *if*). This set of PDU names can change accordingly to the protocol to be tested. If the current input i_k or output o_k (*second external if*) belongs to the set SET_PDU then we look forward in the $TRACE_{Network}$ for the packet identified by the name of the PDU and its TID (using the function $getPacket$). If the packet is not found in the $TRACE_{Network}$ (instruction if $pck = null$) with the required event (i.e. i_k or o_k) then we have a correlation error. This error could be produced by the loss of a packet or by an erroneous behaviour in the transaction layer. In this case, the correlation process is stopped ($fault\_correlate :=$ `true`).
On the other hand, if the packet is found in the trace, we add the information included in the packet to the trace $TRACE'_{trans}$ (initially $TRACE_{trans}$). If we consider i_k then the information of the packet is an input of the trace $TRACE'_{trans}$ and i_k becomes an output (instruction *else*, i.e. $TRACE'_{trans} = \ldots i_{k-1}/o_{k-1}$, $\mathbf{input\_pdu/i_k}, i_k/o_k, i_{k+1}/o_{k+1} \ldots$). If we consider o_k then the information of the packet is an output of the trace $TRACE'_{trans}$ and o_k becomes an input. This part is treated in the second external *if* denoted in the algorithm by *case of a correlation with an output*. This procedure is repeated until the trace $TRACE_{trans}$ is completely covered or a correlation error has been found. In this way, an end-to-end trace $TRACE'_{trans}$ is obtained on which interoperability contextual signatures can be checked.

5. APPLICATION TO THE WAP

This section presents the application of the proposed methodology to a real case study, the WAP protocol. We also present an example of horizontal interoperability property and give the experimental results.

5.1 The Wireless Application Protocol

WAP (Wireless Application Protocol) specifies an application framework and network protocols for wireless devices such as mobile phones, pagers or PDAs (Personal Digital Assistant). One of the main objectives is to bring Internet content and advanced data service to these devices.
WAP is composed of several layers, the topmost of which is the Wireless Application Environment (WAE) that offers a framework for the integration of the different Web and mobile telephony applications. Next comes the Wireless Session Protocol (WSP), which offers to the WAE layer two types of services: connection or connectionless. In the first case, a session oriented service can be provided supporting session initiation, suspension and resumption. The connectionless services are on top of a transport datagram service which can be provided either by a Wireless Datagram Service (WDP) or by UDP (User Datagram Protocol). The session-oriented service is provided on top of the Wireless Transaction Protocol (WTP). This protocol is a confirmed transaction protocol that is a light weight version of TCP.

5.2 Interoperability signatures

This section presents an example of horizontal interoperability invariant, which is expressed by contextual signatures using the grammar that was described in section 3.5. The interoperability properties to be checked on the traces are defined in *XML* and respect the structure of the *DTD*. We do not exhibit vertical interoperability property for the sake of space.
Figure 2 presents a property that checked that the layers $\text{WTP}_{Initiator}$ and $\text{WTP}_{Responder}$ dedicated to the transaction management, interoperate correctly to perform a connection establishment. The figure is decomposed on two simple invariants that are enclosed in the two signatures.

The invariant with the signature *sign-01* checks that a connection request received by the $\text{WTP}_{Initiator}$ (received after the primitive $TR - Invoke.req$ and the parameter $pdu = Connect$) entails the sending of the PDU $Invoke$ to the network. In other words, this invariant controls that a transaction related to the connection establishment is initiated by the $\text{WTP}_{Initiator}$ layer and transmitted through the network to its corresponding distant entity to perform the requested service. The simple invariant is described as follows: $I1 = TR - invoke.req(pdu, 2, false, TID)/?, Invoke(TID)/$ $\{Invoke(TID, 2, ...)\}$
The parameter value TID, which is instantiated in the messages $TR - Invoke.req$ and $Invoke$, allows to warranty that the information sent to the network are those of the initial request of the service.
The invariant represented by the signature *sign-02* checks that the transaction related to the client connection is taken in charge by the $\text{WTP}_{Responder}$ (activated by the event $RcvInvoke$). It also checks that the transaction realizes the requested service by sending the PDU $Result$ to the network. The second invariant is described as follows: $I2 = Invoke(TID, 2, ..)/RcvInvoke(TID, 2, ...)$, $*, TR - Result.req/\{Result(TID, false)\}$
The two invariants are connected by means of the identifier of the transaction (TID). The value of this parameter permits identifying the transactions that are performed by the distant layers and allows to follow the requested service.
These invariants are checked on the correlated traces as previously explained. Two correlations are needed: *(i)* one is performed between the traces of $\text{WTP}_{Initiator}$ and the network traces captured by the sniffer; *(ii)* a second one is realized between the traces of $\text{WTP}_{Responder}$ and the network. Using our TESTINV tool, the invariant defined by the signature *sign-01* is applied on the traces of the first correlation. The invariant with the signature *sign-02* is applied on the traces of the second correlation. This property holds if both invariants return a *Pass* verdict as the two invariants are connected by a logical connector, i.e. an and.

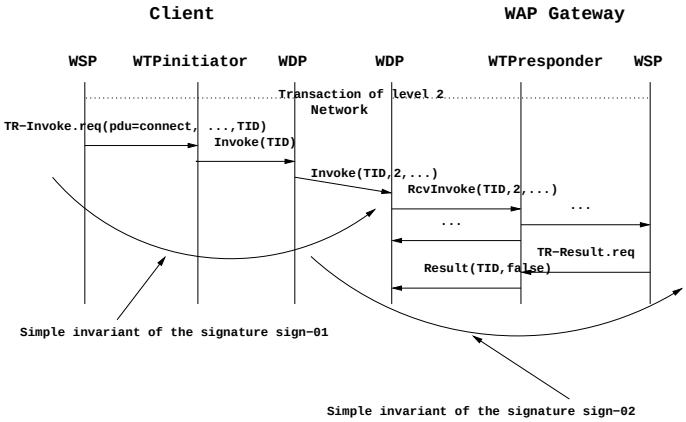

Figure 2: Horizontal Interoperability Property

5.3 Experimental Results

Contextual signatures have been applied to communication traces between several types of Wap clients, i.e. a real mobile phone, a PDA (Personal Digital Assistant) or a mobile simulator and a specific server. We have used the defined POs that have been added

at the interface layers of a PDA and of the simulator for the client side. The server side has also been logged on. We have used an open-source WAP stack, i.e. the Kannel stack (version 2). Interested readers can refer to [6] for more detailed information on the Platonis platform.
The experiments were carried out on traces of more than 1000 messages. We have defined several contextual signatures related to connection establishment (SIG1, corresponding to the signature of the properties exhibited by figure 2) of a mobile client; connection establishment with a PDA (SIG2), a page download of the mobile client (SIG3); and, disconnection requested by the PDA (SIG4). We have performed experiments with several clients to observe different TID parameters. For sake of simplicity, we just give the results for these four signatures (see table 1). Note that the time in seconds (second row) corresponds to the time obtained on a Microsoft Windows XP, Pentium IV, 800mMHz, with 2Go of RAM.
We have also checked on the specification the soundness of the invariants enclosed in these contextual signatures. We were able to perform such verification because the formal description of the WAP protocol (the client and the WAP server) was available in the frame of the `PLATONIS` project.

Numerous experiments have been performed. They are related to the connection establishment with the different level of transport class, i.e. 0, 1 or 2 that conditions the number of messages exchanged between the distant entities. We have tried different values for the parameters of the messages. We have also tested the interoperation based on the data transfer and disconnection phase. In some of the experiments, we obtained the *Inconclusive* verdict. This is because of the length of the traces that were too short. We have adjusted the length to avoid this problem. We also obtained such verdict for a contextual signature related to *push* primitives. We cannot observe such primitives initiated by the gateway since no subscription of provider has been done. The results obtained have established the correct interoperation of two entities regarding the test objectives based on the expert requirements.

Invariant	$SIG1$	$SIG2$	$SIG3$	$SIG4$
Time	3.15	2.60	3.19	1.15
Verdict	$Pass$	$Pass$	$Pass$	$Pass$

Table 1: Verification of horizontal interoperability signatures on traces.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a new methodology for interoperability testing that improve previous approaches and advances the state of the art. The use of contextual signatures eases the expression of interoperability testing properties. Two types of interoperability properties: vertical and horizontal may be described by contextual signatures. These are very useful to respectively check interoperation between layers of the same protocol stack and check that distant entities of the same layer can interoperate.

Another important contribution of this work is the definition of a correlation algorithm. This last enables the application of pattern matching to check the correctness of execution traces with respect to interoperability properties defined as contextual signatures. This algorithm has been implemented in the TESTINV software tool and applied to a real protocol, i.e. the WAP.

We plan to continue our work on contextual signatures and passive testing in the field of network security. The techniques on passive testing we have developed [13] have shown to be well adapted to detect anomalies, attacks and intrusions. However this past work presented some limitations needed to reduce the number of false positives. The use of contextual signatures in this framework could help refine the analysis and reduce their number.

7. REFERENCES

[1] I. 9646-1. *Information Technology - OSI - Conformance testing methodology and framework - Part 1: General Concepts*.

[2] J. Ayache, P. Azema, and M. Diaz. Observer: A concept for on-line detection of control errors in concurrent systems. In *9th Symposium on Fault-Tolerant Computing*, 1979.

[3] E. Bayse, A. Cavalli, M. Nunez, and F. Zaidi. A passive testing approach based on invariants: application to the wap. In *Computer Networks*, volume 48, pages 247–266. Elsevier Science, 2005.

[4] C. Besse, A. Cavalli, M. Kim, and F. Zaidi. Automated Generation of Interoperability Testing. In *TestCom 2002*, pages 169–184, Berlin, 2002. Kluwer Academic.

[5] R. Boyer and J. Moore. A fast string searching algorithm. In *Communications of the ACM*, pages 762–772, 1977.

[6] A. Cavalli, A. Mederreg, F. Zaidi, P. Combes, W. Monin, R. Castanet, M. Mackaya, and P. Laurencot. A multi-service and multi-protocol validation platform experimentation results. In *TestCom 2004*. Springer, 2004.

[7] J. Gadre, C. Rohrer, C. Summers, and S. Symington. A COS study of OSI interoperability. In *Computer Standards and Interfaces*, pages 217–237, 1990.

[8] N. Griffeth, R. Hao, D. Lee, and R. Sinha. Integrated System Interoperability Testing with Applications to VOIP. In *FORTE/PSTV'00*, Pize, Italy, October 2000.

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

[10] Kannel foundation. *Kannel: Overview*. http://www.kannel.org/overview.shtml.

[11] D. Knuth, J. Morris, and V. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(1):323–350, 1977.

[12] O. Koné and R.Castanet. Test generation for interworking systems. In *Computer Communications*, volume 23, pages 642–652. Elsevier Science, 2000.

[13] J. Orset, B. Alcalde, and A. Cavalli. A formal approach for detecting attacks in ad hoc networks. *International Journal of Network Security*, 22:141–149, 2007.

[14] Y. Park, M. Kim, and S. Kang. Conformance Testing of Multiprotocol IUTs. In *The IFIP 12th IWTCS*, Budapest, Hungary, September 1999.

[15] V. Paxson. *The Bro 0.8 User Manual*. ICSI Center for Internet Research, March 2004.

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

[17] Snort. An open source network intrusion prevention and detection system. http://www.snort.org.

[18] G. Vermeer and H. Blik. Interoperabillity testing : Basis for the acceptance of communication systems: Theory and practice. In *Protocol Test Systems, VI (C-19)*. North-Holland, 1994.

[19] Wireshark. An award-winning network protocol analyser. http://www.snort.org.