A Methodology for Passive Interoperability Testing: Application to SIP

Nanxing CHEN* — César VIHO* — Fahmi Chelly**

* IRISA/Université de Rennes 1
Campus de Beaulieu, 35042 Rennes, FRANCE
nanxing.chen@irisa.fr; cesar.viho@irisa.fr
** Centre d'Etudes et de Recherche des Télécommunications (CERT)
Citée Technologique des Communications
Route de Raoued, Km 3.5, B.P-111-2088-Ariana-Tunisie
fahmi.chelly@cert.mincom.tn

RÉSUMÉ. L'objectif du test d'interopérabilité de protocoles est de s'assurer que les composants réseaux interagissent correctement et qu'elles rendent les services prévus. Pour effectuer le test d'interopérabilité, les approches classiques s'appuient sur la méthode dite active, dont l'objectif est de tester les implémentations en effectuant une suite de stimuli et d'observations sur celles-ci. Cependant, les stimuli injectés perturbent inévitablement le fonctionnement normal des composants. A l'inverse, le test dit passif a pour objectif de détecter des erreurs dans un système en s'appuyant uniquement sur les observations. Cet article présente une approche passive pour le test d'interopérabilité basée sur les modèles formels. A partir des spécifications d'implémentations des protocoles à tester et d'un objectif de test, un scénario de test d'interopérabilité passif est obtenu en calculant partiellement leurs interactions. De plus, nous proposons un algorithme d'analyse de trace, permettant de vérifier le scénario sur des traces d'exécution préalablement enregistrées. L'approche proposée a été appliquée sur une étude de cas du protocole SIP, où des situations de non-interopérabilité ont été détectées.

ABSTRACT. The purpose of interoperability testing is to ensure that protocol implementations communicate correctly while providing the expected services. To perform interoperability testing, conventional approaches rely on the active testing method, which stimulates the network components and observing their reactions. However, the stimuli disturb inevitably their normal operation. On the contrary, passive testing is a technique based on only observing the external behavior of the network components. This paper proposes a formal approach for passive interoperability testing. Given the specifications of the network components and a test objective, a passive test scenario is derived by partially calculating their interaction. In addition, a trace analysis algorithm is proposed to verify the interoperability between the implementations. The proposed approach was successfully performed on a SIP protocol case study, where non-interoperability situation was detected.

MOTS-CLÉS : test d'interopérabilité, test passif, modèle IOLTS, vérification de trace

KEYWORDS : interoperability testing, passive testing, IOLTS model, trace verification

1. Introduction

Interoperability testing [8] (*iop* for short in the sequel) is an important activity to guarantee the correct cooperation of heterogeneous network applications while providing the required quality of services. To perform interoperability testing, active testing method [3, 8] is currently widely used, which is done by actively stimulating the *implementations under test* (IUT) and verifying the corresponding responses. However, if the tester is not provided with a direct interface to stimulate the IUTs, or in operational environment where the normal operation of IUTs cannot be interrupted, active testing can be difficult or even impossible to perform .

To cope with the drawbacks of active testing, passive testing has been studied [6, 7], which aims to test a running system by only observing its external behavior. Its non-intrusive nature makes it an appropriate method in the context of operational networks, which is often required in interoperability testing. This paper, arguing for the passive testing technique, formalizes passive interoperability testing, as well as presents a model-based methodology to carry out the test. The suggested methodology has been successfully performed on a Session Initiation Protocol (SIP) case study.

The paper is organized as follows : In section 2, formal background and passive iop definitions are presented. Section 3 presents the model-based passive iop testing methodology. The application of the proposed approach and the experimental results are exhibited in section 4. Finally, section 5 gives the conclusions and the perspectives.

2. Preliminaries

2.1. Passive iop testing architecture

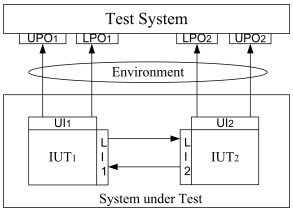

Figure 1. *Passive interoperability testing architecture*

The passive interoperability architecture considered in this paper (cf. Fig.1) involves a *test system* (TS) and a *system under test* (SUT) composed of two *implementations under test* (IUT) from different product lines. The communication between two IUTs is assumed to be asynchronous, noted $IUT_1 \|_A IUT_2$, as messages may be exchanged by traversing several protocol layers. Consequently, the communication between two IUTs is modeled as two unidirectional FIFO channels [2]. In this paper, we consider the $black\ box$ passive testing. i.e., testing activities rely only on observing the external behavior of the IUTs, including the services provided by the IUTs as well as their interactions. To do so, *points of observation* (PO) are installed : respectively, *upper points*

of observation (UPO) collect the messages (services) provided by the IUTs to their upper layer through their *upper interfaces* (UI), while *lower points of observation* (LPO) collect the messages exchanged between peer IUTs through their *lower interfaces* (LI). Traces retrieved by different POs are correlated into a global trace by the test system, which will then be analyzed to check the interoperability.

2.2. Formal model

The testing theory is based on the notions of specifications, implementations and passive interoperability criterion (cf.Definition 2). In this paper, we use the IOLTS (Input-Output Labeled Transition System) to model specifications and IUTs [2]. (c.f. an example in section 3.1.3)

Definition 1 An IOLTS is a tuple $M = (Q^M, \Sigma^M, \Delta^M, q_0^M)$ where Q^M is the set of states of M with q_0^M its initial state. Σ^M is the set of observable events at the interfaces of M. $\Sigma^M = \Sigma_U^M \cap \Sigma_L^M$, where Σ_U^M (resp. Σ_L^M) is the set of events at the upper (resp. lower) interfaces. Σ^M can also be partitioned to distinguish inputs (Σ^{M^I}) and outputs (Σ^{M^O}). An input (resp. output) a at interface p is noted by $p?a$ (resp. $p!a$). Also, $Out(q)$ (resp.$In(q)$) represents the set of possible outputs (resp.inputs) at state q. Δ^M is the set of transitions, where each transition is noted by $(q, \alpha, q') \in \Delta^M$.

Definition 2 Passive interoperability criterion : two IUTs are considered interoperable iff after a trace of their interaction, all observed outputs sent from each IUT during its interaction with its counterpart are foreseen in the corresponding interaction of their specifications.

3. Passive Interoperability Testing

The existing passive testing methods broadly consist of $tracemapping$ and $invariant$. Trace mapping aims at evaluating the behavior of the system by comparing its observed behavior (trace) strictly with its specification. But iop testing involves several network components, to model the whole system therefore encounters state explosion. Invariant approach was proposed in [7] to focus on the important properties defined as invariants. However it is not based on a rigorous definition of interoperability. In this paper, we propose another method to formalize passive iop testing while avoiding state explosion. The outlines of the testing method are the following : *(i) iop test case (ITC)* generation. It takes three inputs : the specifications (noted S_1 and S_2) on which the IUTs are based, and an *iop test purpose* (ITP) which describes an important property to be verified [1] (cf.Definition 3 and an example in Section 3.1.3.). ITC aims to obtain different ways of specifications' interaction, noted ($S_1 \|_A S_2$) to reach the ITP. *(ii)* Then, a *passive iop test case* is derived by extracting properly relevant observable events w.r.t the ITP and assigning appropriate verdicts. Once the traces recording finishes, the verification of the ITP is performed and a verdict $\in \{Pass, Fail, Inconclusive\}$, is emitted. Respectively, $Pass$ means the test purpose is reached without any error detected, $Fail$ means at least one error is detected, and

$Inconclusive$ means the behavior of IUTs is not forbidden by the specifications, however does not correspond to the test purpose.

Definition 3 An iop test purpose is a complete deterministic acyclic IOLTS : ITP =(Q^{ITP}, Σ^{ITP}, $\triangle^{ITP}$, q_0^{ITP}), where : $\Sigma^{ITP} \subseteq \Sigma^{S_1} \cup \Sigma^{S_2}$. $\triangle^{ITP}$ is the transition relation. Q^{ITP} is the set of states with q_0^{ITP} its initial state. ITP is associated with two disjoint trap states sets $Accept^{ITP}$ (resp.$Refuse^{ITP}$) $\subseteq Q^{ITP}$. Specifically, $Accept^{ITP}$ stands for *the targeted behavior* to be observed. While $Refuse^{ITP}$ stands for other behavior allowed in the specifications, but does not help in reaching the targeted behavior. $Accept^{ITP}$ and $Refuse^{ITP}$ are only directly reachable by the observation of outputs produced by the IUTs. ITP is complete, which is done by inserting "$*$" label at each state q of the ITP, where "$*$" is an abbreviation for the complement set of all other events leaving q. It allows to to describe a property without taking into account the complete sequence of detailed specifications interaction.

3.1. Passive interoperability test case derivation

3.1.1. Passive interoperability test case generation

Given the specifications and an ITP, our aim is to build a passive iop test case ($PITC$, $Verdicts$) which is $sound$. To do so, firstly, an iop test case ITC = $S_1 \times S_2 \times ITP$ is built to calculate the different ways of $S_1 \|_A S_2$ to reach the ITP. Formally, ITC =(Q^{ITC}, Σ^{ITC}, $\triangle^{ITC}$, q_0^{ITC}) where : $\Sigma^{ITC} \subseteq \Sigma^{S_1} \cup \Sigma^{S_2}$. Each state ($q^{S_1}, q^{S_2}$,$q^{ITP}$,$f_1$,$f_2$) $\in Q^{ITC}$ is a composite state such that $q^{S_1} \in Q^{S_1}$, $q^{S_2} \in Q^{S_2}$, $q^{ITP} \in Q^{ITP}$; f_1 (resp.f_2) represents the input queue of S_1 (resp.S_2). q_0^{ITC}=($q_0^{S_1}, q_0^{S_2}$,q_0^{ITP}, ε, ε) is the initial state, where ε denotes an empty input queue. $Accept^{ITC}$ and $Refuse^{ITC}$ are two sets of trap states in ITC, where $Accept^{ITC}$= $Q^{ITC} \cap (Q^{S_1} \times Q^{S_2} \times Accept^{ITP})$, $Refuse^{ITC}$ = $Q^{ITC} \cap (Q^{S_1} \times Q^{S_2} \times Refuse^{ITP})$. The set of transitions $\triangle^{ITC}$ is obtained in the following way : Let q^{ITC} =(q^{S_1}, q^{S_2}, q^{ITP}, f_1, f_2) be a state of ITC, and a be an executable event at either q^{S_1} or q^{S_2} or q^{ITP}. A new transition $(q^{ITC}, a, p^{ITC}) \in \triangle^{ITC}$ is created by using the corresponding *asynchronous interaction operation rules* defined below if the current state of ITP is in neither in $Accept^{ITC}$ nor $Refuse^{ITC}$. Otherwise, no new transition is created.[1]

Definition 4 Asynchronous Interaction Operation Rules

Rule 1 $\frac{a \in \Sigma_U^{S_i}, (q^{S_i}, a, p^{S_i}) \in \triangle^{S_i}, (q^{ITP}, a, p^{ITP}) \in \triangle^{ITP}}{(q^{ITC}, a, p^{ITC}) \in \triangle^{ITC}, p^{ITC} = (q(q^{S_i}/p^{S_i}), p^{ITP}, f_1, f_2)}$ 2

Rule 2 $\frac{a \in \Sigma_L^{S_i^O}, (q^{S_i}, a, p^{S_i}) \in \triangle^{S_i}, (q^{ITP}, a, p^{ITP}) \in \triangle^{ITP}}{(q^{ITC}, a, p^{ITC}) \in \triangle^{ITC}, p^{ITC} = (q(q^{S_i}/p^{S_i}), p^{ITP}, f(f_j.a))}$ 3

Rule 3 $\frac{a \in \Sigma_L^{S_i^I}, a \in head(f_i), (q^{S_i}, a, p^{S_i}) \in \triangle^{S_i}, (q^{ITP}, a, p^{ITP}) \in \triangle^{ITP}}{(q^{ITC}, a, p^{ITC}) \in \triangle^{ITC}, p^{ITC} = (q(q^{S_i}/p^{S_i}), p^{ITP}, f(f_i \backslash a))}$ 4

1. There is no need to further calculate the interaction, as $Accept^{ITC}$ means ITP has been reached, while $Refuse^{ITC}$ means the interaction of specification will not lead to the desired target.
2. Here, a is an upper interface event of S_i (i = {1,2}), $q(q^{S_i}/p^{S_i})$ indicates that in the global state q^{ITC}, the local state q^{S_i} is change to p^{S_i}, and other local states keep unchanged.
3. Here, a is a lower interface output of S_i, $f(f_j.a)$ means that a is put into the tail of the input queue of S_j.
4. Here, a is the first element in the input queue of S_i, $f(f_i \setminus a)$ means a is removed from the input queue of S_i.

The ITC generation is based on partial reachability graph calculation. It intends to unfold the specifications and to mark the sequences of their interaction that reach $Accept^{ITC}$ and $Refuse^{ITC}$. In this paper, we use depth-first traversal to minimize memory requirement : from the initial state q_0^{ITC}, recursively at each step new transition(s) are created according to the asynchronous interaction operation rules until no more state can be explored. Indeed, $Accept^{ITC}$ and $Refuse^{ITC}$ limit states exploration, thus reduce state space explosion. (c.f. an example in section 3.1.3.)

3.1.2. Passive interoperability test case derivation

The derivation of *a passive iop test case* (PITC) consists of two steps : *(i)* Traversing the ITC graph and extracting adequately only the observable behavior as required by passive testing. In fact, The ITC can be divided into two parts. The first part involves the states and transitions that traverse the initial state of ITC and (not including) the first transition concerned by the test purpose. Indeed, this part is usually used in active testing (referred as $preamble$) to put the SUT into a desired state to begin the verification of the second part, representing the relevant events to the ITP (also called $testbody$). Therefore, in PITC, only the second part is preserved. Moreover, input events (from the point view of IUTs) should be removed : In black box testing, an input of IUT cannot be directly observed. Verifying an input implies verifying its causal related observable outputs. All the events that need to be removed in ITC are replaced by an action τ. The other events in ITC are extracted by applying τ - reduction [4] followed by determinization. *(ii)* Assigning verdicts to PITC. Specifically, the state in $Accept^{ITC}$ is associated with $Pass$ verdict. While states in $Refuse^{ITC}$ are associated with $Inconclusive$ verdict (traces allowed by specifications but cannot lead to $Accept^{ITC}$). Note that $Fail$ verdict is not explicitly specified due to the uncontrollability nature of passive testing. Indeed, we require that the PITC be sound, i.e., interoperable IUTs cannot be rejected. Verdict assignment issue will be discussed in detail in Section 3.2.

3.1.3. An example

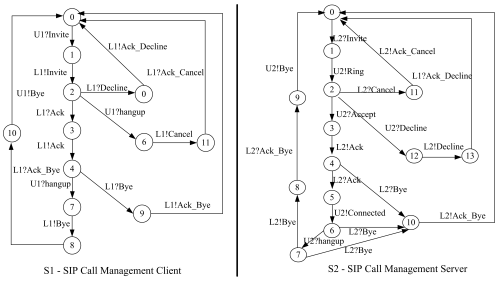

Figure 2. *Simplified SIP CMC and CMS specifications*

Fig.2 illustrates an example of a simplified Session Initiation Protocol (SIP) [5], which is a frequently used application layer signaling protocol for creating, modifying and terminating sessions of participants. In Fig.2, S_1 and S_2 represents respectively the specification of a Call Management Client (CMC) and a Call Management Server (CMS). Specifically, if CMC receives a connection request message from its upper layer user ($U1?Invite$), it transmits this request to CMS ($L1!Invite$). Upon the reception, the user

of CMS can either accept ($U2?Accept$) or refuse ($U2?Decline$) the connection. Besides, CMC user can hang up at any time ($U1?hangup$). If the user decides to hang up before the connection is established ($L1!Ack$ is sent), CMC will cancel the $Invite$ request ($L1!Cancel$). Otherwise, CMC needs to issue a Bye request toward the CMS ($L1!Bye$).

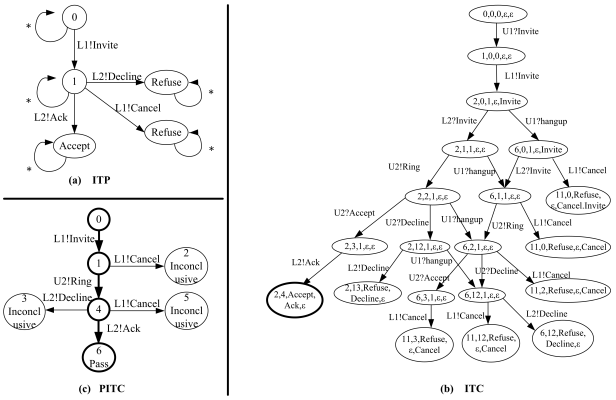

Figure 3. *An Example of PITC Derivation*

In this example, the ITP (Fig. 3-a) aims at testing the connection establishment : After CMC sends an $Invite$ request, after a number of interactions, CMS sends Ack to indicate that a call is ready to be setup. Meanwhile, the possibility that CMC cancels the $Invite$ request ($L1!Cancel$) or CMS refuses the connection ($U2?Decline$) exists but they do not belong to the test objective (Thus they are associated with $Inconclusive$ verdict). Fig. 3-b illustrates the ITC : the necessary interaction of S_1 and S_2 to reach the ITP is calculated ($Accept^{ITP}$ reached). Then, by reducing the state-space of ITC and verdict assignment, the PITC is obtained (Fig. 3-c).

3.2. Trace analysis and verdicts

PITC represents the expected behavior of the IUTs to be observed w.r.t the ITP. It is in form of a directional acyclic deterministic tree, associated with $Pass$ or $Inconclusive$ at trap states. In passive testing, no assumption is made about the moment when the recording of trace begins. To deal with the issue, we propose a trace analysis algorithm, which aims at checking whether the trace σ reaches the $Pass$ and $Inconclusive$ verdict in the PITC tree. This can be done by checking recursively the states in PITC. In detail, we call *States_under_reading* the set of currently checked states of PITC. Initially, it contains only the initial state of PITC. Then, for each event a taken in order from the trace σ, we check whether a can be accepted by the states in *States_under_reading*. If it is the case, these states are replaced by the destination states in the PITC tree led by a. Other states (except the initial state q_0^{PITC}) are deleted for consistency reason[5]. Formally :

(1) $Pass\_reached$ (resp. $Inc\_reached$) : Boolean value=$True$ if trace reaches $Pass$ (resp.$Inconclusive$) verdict.

(2) *pick* (σ) *:* Take the first element from trace σ. i.e. σ=a.σ', $pick(\sigma) = a, \sigma = \sigma$'.

5. q_0^{PITC} is always in the set because a trace may contain several identical events that match the label at q_0^{PITC}, however not all of them will lead to $Pass$ state. The complexity of the trace analysis algorithm is $O(M \times N)$, where where M is the size of the trace, N is the number of states in *States_under_reading*.

Algorithm 1: Trace verification algorithm

Input: Trace σ, $PITC$
Output: $Pass\_reached$, $Inc\_reached$
Initialization : $State\_under\_reading = q_0^{PITC}$, $Pass\_reached = False$, $Inc\_reached = False$;
while $\sigma \neq \emptyset$ and *not* $Pass\_reached$ **do**
  pick (σ) ;
  forall the *state q in* $State\_under\_reading$ **do**
    if $a \in Out(q)$ **then**
      if $q == q_0^{PITC}$ **then**
        add $(State\_under\_reading, p)$ *where* $(q_0^{PITC}, a, p) \in \triangle^{PITC}$
      end
      else
        $q = p$ where $(q, a, p) \in \triangle^{PITC}$
      end
    end
    if $a \notin Out(q)$ *and* $q \neq q_0^{PITC}$ **then**
      remove$(State\_under\_reading, q)$;
    end
    if $q ==$ $Pass$ **then**
      $Pass\_reached = True$; *exit* //* exit from the for loop
    end
    if $q ==$ $Inconclusive$ **then**
      $Inc\_reached = True$
    end
  end
  Return $Pass\_reached$, $Inc\_reached$
end

After the execution of trace verification, an appropriate verdict is emitted accordingly : *(i)* If $Pass\_reached$=$True$, Verdict=$Pass$ as the ITP is satisfied. *(ii)* If $Pass\_reached$=$False \wedge Inc\_reached$=$True$, Verdict=$Inconclusive$. *(iii)* Otherwise, i.e., the trace of IUTs does not reach $Pass$ or $Inconclusive$ verdict in the PITC, a $Fail$ verdict is emitted. It should be mentioned that $Fail$ verdict means that the ITP is not reached. To further identify fault, postmortem analysis is needed (which is not in the scope of testing). This is because in passive testing, it is impossible for the test system to control/configure the SUT to the desired states. Moreover, as passive testing is often done in operational environment, delay and packet loss may take place. Besides, recorded traces may be not long enough to encompass the whole test purpose. In these cases, the difference between SUT's behavior and the PITC does not allow to easily conclude the non-interoperable behavior. Nevertheless, different reasons that lead to $Fail$, i.e., short length of trace, time-out, unspecified messages or badly formatted packets can help further diagnosis analysis.

4. Experimental Results

The proposed method was implemented and applied on a SIP protocol case study. In our experiments, 4 different SIP phones : Blink, Ekiga, Jitsi and Linphone, have been installed. During the test, two SIP phones of different pair-wise combinations were connected. 100 global traces produced by the SIP phones (from 5 seconds to 5 minutes) were

collected, filtered by a Wireshark sniffer and stored in pcap format. We have chosen ITPs concerning the basic functionalities of SIP. For sake of simplicity, we just give the results for 3 ITPs : The establishment of the media session, call cancellation and the call termination. Totally we got 86% $Pass$ verdicts, 10.7% $Inconclusive$ verdicts and 3.3% $Fail$ verdicts due to different reasons (short length of traces, non-observation of the events specified in PITC, time-out, etc). A postmortem diagnosis was carried out and the none-interoperability situation was detected during the call termination of Ekiga and Jtisi phones : After Jtisi sends Bye request, instead of an Ack_Bye, Ekiga Siphone reports a timeout and replies $481\ Call/Leg\ Transition\ doesnot\ exist$. By using passive testing, non interoperability situation was detected while the environment of operational networks was not disturbed.

5. Conclusion and Future Work

This paper proposes a model based methodology for passive interoperability testing. Given the specifications of the IUTs and an iop test purpose, a passive iop test case can be derived and executed on the trace produced by the IUTs to check their interoperability. The method was implemented and carried out successfully on a SIP case study. Future work will consider extending this method to multi-components interoperability testing.

6. Bibliographie

[1] Schulz, S., Wiles, A., Randall, S. : *TPLan - A notation for expressing test purposes.* ETSI, TestCom/FATES, LNCS 4581, pp. 292–304, 2007.

[2] Verhaard, L., Tretmans, J., Kars, P. Brinksma, Ed. : *On asynchronous testing.* In Protocol Test Systems, vol C-11 of IFIP Transactions, pp. 55–66. 1992.

[3] R.Hao, D.Lee, R.K.Sinha and N.Grieth. *Intergated system interoperability testing with applications to VoIP.* IEEE/ACM Trans. Netw., 12(5) : pp.823-826, 2004.

[4] Cormen, T. H., Leiserson, Charles E., Rivest, Ronald, L. : *Introduction to Algorithms.* MIT Press and McGraw-Hill. ISBN 0-262-03141-8. pp.558–565, 1990.

[5] Handley, M., Shulzrinne, H., Schooler, E., Rosenberg, J. : *SIP : Session Initiation Protocol.* Request For Comments (Proposed Standard) 2543, Internet Engineering Task Force 1999.

[6] Andrés, C., Merayo, M.G., Núñez, M. : *Passive Testing of Timed Systems.* ATVA 2008 : pp. 418-427, 2008

[7] Zaidi, F., Cavalli, A., Bayse, E. : *Network Protocol Interoperability Testing based on Contextual Signatures.* The 24th Annual ACM Symposium on Applied Computing SAC'09, 2009.

[8] Desmoulin, A., Viho, C. : *Automatic Interoperability Test Case Generation Based on Formal Definitions.* Lecture Notes in Computer Science, Volume 4916/2008, pp. 234-250, 2008.