A New Method for Interoperability Test Generation

Alexandra Desmoulin and César Viho

IRISA/Université de Rennes 1,
Campus de Beaulieu,
35042 Rennes Cedex, France
{alexandra.desmoulin,viho}@irisa.fr

Abstract. Interoperability testing aims at verifying the possibility for two or more components to communicate correctly while providing the foreseen services. In this paper, we describe a new method for generating interoperability test cases. This method is equivalent to classical methods in terms of non-interoperability detection. Contrary to classical approaches, this method avoids the well-known state-space explosion problem. It has been implemented in the CADP Toolbox and applied to a simplified version of the ISDN connection protocol. The obtained results confirm the real contribution of this method: test cases has been derived while classical approaches face the state-space explosion problem.

1 Introduction

Interoperability testing is used to verify that different protocol implementations communicate correctly while providing the services described in their respective specification. Contrary to conformance testing which is precisely characterized with testing architectures, formal definitions [1, 2] and tools for generating automatically tests [3, 4], interoperability is not formally defined. Some formal definitions [5, 6] and methods for generating interoperability tests [7, 8] exist, but there is no precise characterization of interoperability for the moment, and consequently no method based on formal definitions.
In this paper, we consider interoperability formal definitions of [9]. These definitions, called interoperability criteria, describe the conditions that two implementations must satisfy to be considered interoperable: providing the expected service while interacting correctly. Based on a proved equivalence between two of these criteria, a new method to generate automatically interoperability test cases is described. This method is equivalent to classical methods in terms of non-interoperability detection. But it avoids the construction of the specification interaction that may lead to the well-known state-explosion problem [6]. Moreover, we apply this method -implemented in the CADP Toolbox [10]- on a simplified version of the ISDN (Integrated Service Digital Network) connection protocol [11]. The obtained results show that the proposed method is a real contribution as we are able to derive interoperability test cases while classical methods were not applicable because of the state-space explosion problem.

This paper is structured as follows. First, we present the formal background including interoperability formal definitions needed in Section 2. Then, Section 3 focuses on methods for generating interoperability test cases. We present first classical method and then our new method. Section 4 describes the results of the application of both methods on a simplified version of the ISDN connection protocol. Conclusion and future work are in Section 5.

2 Formal background

In this section, we present the different notions that are used in the following. First, interoperability is defined in Section 2.1. Section 2.2 presents the formal model (IOLTS) and the related definitions used for interoperability formal approach. Finally, Section 2.3 describes the interoperability formal definitions (iop criteria) considered for interoperability test generation.

2.1 Preliminary definitions

Different kinds of tests exist for testing if protocol implementations will work correctly in an operational environment. For example, conformance testing verifies that an implementation behaves as described in its specification. It considers events observed on the interfaces of the Implementation Under Test (IUT), and compares these events with events foreseen in the specification. The IUT is a black-box: testers do not have any knowledge of its internal structure.
In this paper, we consider another kind of test: interoperability testing. This kind of test has two goals. It verifies that different IUTs (two in this study) can communicate correctly, *and* that they provide the services described in their respective specification while communicating. In an interoperability testing architecture, we can differentiate two kinds of interfaces. The Lower Interfaces are the interfaces used for the interaction while the Upper Interfaces are used for the communication of the implementations with upper layer. Testers are connected to these interfaces but they can *control* (send message) only the upper interfaces. The lower interfaces are only *observable*.
Depending on the access to the interfaces, different architectures can be distinguished. For example, the interoperability testing architecture is called *unilateral* if only the interfaces of one IUT are accessible during the interaction, *bilateral* if the interfaces of both IUTs are accessible but separately, or *global* if the interfaces of both IUTs are accessible with a global view.

2.2 IOLTS model and related definitions

We use IOLTS (Input-Output Labeled Transition System) [12] to model specifications. As usual in the black-box testing context, we also need to model IUTs, even though their behaviors are unknown. They are also modeled by an IOLTS.

Definition 1. *An IOLTS is a tuple* $M=(Q^M,\Sigma^M,\Delta^M, q_0^M)$. Q^M *is the set of states and* $q_0^M \in Q^M$ *the initial state.* Σ^M *denotes the set of observable events on*

the interfaces: $p?m \in \Sigma^M$ *(resp.* $p!m \in \Sigma^M$*) stands for an input (resp. output) where* p *is the interface and* m *the message.* Δ^M *is the transition relation.*

Based on this model, $\Gamma(q)$ is the *set of observable events* (executed on the interfaces of M) from the state q and $\Gamma(M, \sigma)$ the set of observable events for the system M after the succession of events (*trace*) σ. In the same way, $Out(M, \sigma)$ (resp. $In(M, \sigma)$) is the set of possible outputs (resp. inputs) for M after the trace σ. $Traces(q)$ is the set of possible observable traces from q. We can also define $\bar{\mu}$ as $\bar{\mu}{=}l_i!a$ if $\mu = l_j?a$ and $\bar{\mu} = l_i?a$ if $\mu = l_j!a$.

An implementation can be *quiescent* in three different situations: either the IUT can be waiting on an input, either it can be executing a loop of internal (non-observable) events, or it can be in a state where no event is executable. For an IOLTS M, a quiescent state q is treated as an observable (practically with timers) output event. An IOLTS with quiescence modeled is noted $\Delta(M)$.

Two other operations need to be modeled: asynchronous interaction and projection. The asynchronous interaction is used to calculate the IOLTS modeling the behavior of a system composed by different communicating entities. For two IOLTS M_1 and M_2, the asynchronous interaction is noted $M_1 \|_{\mathcal{A}} M_2$. The way to obtain this model of the interaction is described in [9]. First, M_1 and M_2 are transformed into IOLTS representing their behavior in an asynchronous environment. Then, these two IOLTS are composed to obtain $M_1 \|_{\mathcal{A}} M_2$.
The projection of an IOLTS on a set of events is used to represent the behavior of the system reduced to specific events (such as events observable on certain interfaces). The projection of M on the set of events executable on its lower interface Σ_L^M is noted M/Σ_L^M.

Conformance formal definition Contrary to interoperability testing, conformance is precisely formalized with formal definitions [1, 2] and tools for generating automatically tests like TGV [3] or TorX [4]. Among the different formal definitions existing for conformance, the **ioco** conformance relation [2] says that an implementation I is **ioco**-conformant with respect to its specification S if I never produces an output which could not be produced by S after the same trace. Moreover, I may be quiescent only if S can do so. Formally : I **ioco** $S =_{\Delta} \forall \sigma \in Traces(\Delta(S)), Out(\Delta(I), \sigma) \subseteq Out(\Delta(S), \sigma)$.

2.3 Interoperability formal definitions: iop criteria

Even though some formal definitions exist in [9, 13], there is no precise characterization for interoperability (*iop* for short in the following). Here, we present some formal definitions, called *iop criteria*. They consider different possible architectures for testing the interoperability of two IUTs (one-to-one context) and are based on both purposes of interoperability: verifying that each entity actually receives the outputs sent by the peer entity and that the messages sent by the IUTs on their upper interfaces correspond to the service described in the specifications. Thus, outputs must be verified on both interfaces. As only outputs can

be observed, verifying that an input μ is actually received by the peer entity implies to determine the set of outputs that can happen only due to the reception of μ. This set of outputs is calculated based on causal dependencies. The set of outputs on M that are causally dependent of the input μ after the trace σ is noted: $CDep(M, \sigma, \mu)$.

The **global iop criterion** considers both kinds of interfaces and both IUTS globally. It says that, after a trace of the interaction of the specifications, all outputs observed during the interaction of the implementations must be foreseen in the specifications, and that outputs sent by one IUT via its lower interfaces must be effectively received by the interacted IUT. This iop criterion corresponds to the most used testing architecture.

Definition 2 (Global iop criterion iop_G). $I_1 iop_G I_2 =_{def}$
$\forall \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $Out(I_1 \|_{\mathcal{A}} I_2, \sigma) \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$
and $\forall \{i, j\} = \{1, 2\}$, $i \neq j$,
$\forall \sigma \in Traces(S_i \|_{\mathcal{A}} S_j)$, $\sigma_i = \sigma / \Sigma^{S_i} \in Traces(S_i)$, $\sigma_j = \sigma / \Sigma^{S_j} \in Traces(S_j)$, $\forall \mu \in Out(I_i, \sigma / \Sigma^{S_i})$, $\forall \sigma' \in [(\Sigma^{S_i} \cup \Sigma^{S_j} \cup \{\delta(i), \delta(j)\}) \setminus \bar{\mu}]^* \cup \{\epsilon\}$, $\sigma.\mu.\sigma'.\bar{\mu} \in Traces(S_i \|_{\mathcal{A}} S_j)$, $\bar{\mu} \in In(I_j, \sigma_j.(\sigma' / \Sigma^{I_j})) \Rightarrow Out(I_j, \sigma_j.(\sigma' / \Sigma^{I_j}).\bar{\mu}.\sigma_k) \in CDep($ $S_j, \sigma_j.(\sigma' / \Sigma^{I_j}), \bar{\mu})$, $\sigma_k \in (\Sigma_I^{S_j})^* \cup \{\epsilon\}$

In [9], we prove the equivalence of the global criterion with the so-called bilateral iop criterion iop_B (defined below via the unilateral iop criterion iop_U) in terms of non-interoperability detection. This equivalence is used for developing our new interoperability test generation method.
The **unilateral iop criterion** iop_U (view I_1) considers interfaces of IUT I_1 while interacting with I_2. It says that, after a trace of S_1 observed during the interaction, all outputs observed in I_1 must be foreseen in S_1, and that I_1 must be able to receive outputs sent by I_2 via its lower interfaces.

Definition 3 (Unilateral iop criterion iop_U). $I_1 iop_U I_2 =_{def}$
$\forall \sigma_1 \in Traces(\Delta(S_1))$, $\forall \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$,
$\sigma / \Sigma^{S_1} = \sigma_1 \Rightarrow Out((I_1 \|_{\mathcal{A}} I_2) / \Sigma^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1), \sigma_1)$
and $\forall \sigma_1 = \sigma / \Sigma^{S_1} \in Traces(\Delta(S_1))$ *such that* $\sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $\forall \mu \in Out(I_2, \sigma / \Sigma^{I_2})$, $\forall \sigma' \in [(\Sigma^{S_1} \cup \Sigma^{S_2}) \setminus \bar{\mu}]^* \cup \{\epsilon\}$, $\sigma.\mu.\sigma'.\bar{\mu} \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $\bar{\mu} \in In(I_1, \sigma_1.(\sigma' / \Sigma^{I_1})) \Rightarrow Out(I_1, \sigma_1.(\sigma' / \Sigma^{I_1}).\bar{\mu}.\sigma_i) \in CDep(S_1, \sigma_1.(\sigma' / \Sigma^{I_1}), \bar{\mu})$, $\sigma_i \in (\Sigma_I^{S_1})^* \cup \{\epsilon\}$

The **bilateral total iop criterion** iop_B is verified iff both (on I_1 side and I_2 side) unilateral criteria are verified: $I_1 iop_B I_2$ $(= I_2 iop_B I_1) =_{def} I_1 iop_U I_2 \wedge I_2 iop_U I_1$.

3 Interoperability test generation

3.1 Preliminary definitions

Iop test purpose In practice, a test purpose is an informal description of behaviors to be tested. Generally it is an incomplete sequence of actions. Formally,

a test purpose TP can be represented by a deterministic and complete IOLTS equipped with trap states used to select targeted behaviors. Complete means that each state allows all actions. In this study, we consider simplified iop test purposes with only one possible action after each state ($\forall\ \sigma,\ |\Gamma(TP, \sigma)| \leq 1$) and one $Accept^{TP}$ trap state used to select the targeted behavior.

Iop test cases During interoperability tests, three kinds of events are possible for the tester: sending of stimuli to the upper interfaces of the IUTs, receiving inputs from these interfaces, and observing events (input and output) on the lower interfaces. Thus, a test case TC can be represented by $TC = (Q^{TC}, \Sigma^{TC}, \Delta^{TC}, q_0^{TC})$, an extended version of IOLTS. $\{PASS, FAIL, INC\} \subseteq Q^{TC}$ are trap states representing interoperability verdicts. $\Sigma^{TC} \subseteq \{\mu|\bar{\mu} \in \Sigma_U^{S_1} \cup \Sigma_U^{S_2}\} \cup \{?(\mu)|\mu \in \Sigma_L^{S_1} \cup \Sigma_L^{S_2}\}$. $?(\mu)$ denotes the observation of the message μ (that can be an input or an output) on a lower interface.

Iop verdicts The execution of the test case TC on the system composed of the two IUTs gives an interoperability verdicts. PASS means that no interoperability error was detected during the tests. FAIL stands for the iop criterion is not verified. INC (for Inconclusive) is for the case where the behavior of the SUT seems valid but it is not the purpose of the test case.

3.2 Classical methods

In practice, most of interoperability test suites are written "by hand". This is done by searching "manually" paths corresponding to the test purpose in the specifications. Considering the number of possible behaviors contained in the specification interaction, this "manual" test derivation is an error-prone task.
Methods for automatic interoperability test generation (as in [7, 8, 14, 15, 16]) also consider algorithms that search paths corresponding to the test purpose in the composition of the specifications (sometimes called reachability graph). The study described in [6, 13] considers an interoperability formal definition that compares events executed by the system composed of the two implementations with events foreseen in the specifications. Thus, traditional methods for deriving interoperability test cases are based on a global approach and on a general interoperability definition corresponding to the formal iop global criterion iop_G. The classical method can be summarized, as in Figure 1(a), by two main steps. The first one is the calculation of the specification interaction (completely or based on the defined test purpose depending on the method). The second step corresponds to the interoperability test case derivation based on the model of the specification interaction and on the test purpose.

The problem with this (these) classical method(s) is that we can have state space explosion when calculating the asynchronous interaction of the specifications [6]. Indeed, the number of states in the specification asynchronous interaction is in the order of $O((n.m^f)^2)$ where n is the number of states in the specifications, f the size of the input FIFO queue on lower interfaces and m the

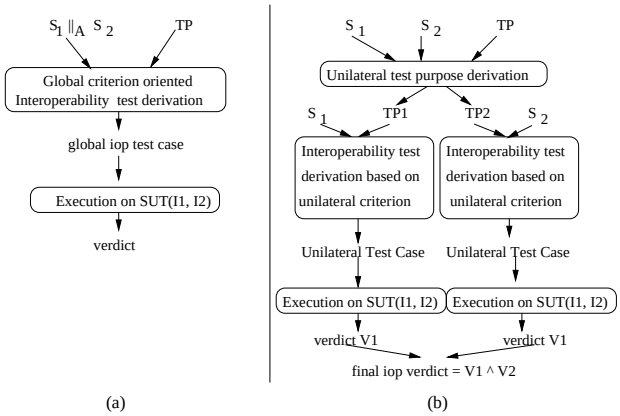

Fig. 1. Interoperability test generation: classical and new methods

number of messages in the alphabet of possible inputs on lower interfaces. This calculation can be infinite if the size of the input FIFO queues is not bounded.

3.3 New approach: bilateral criterion based method

The equivalence -in terms of non-interoperability detection- between global and bilateral iop criteria (cf. theorem 1 in [9]) suggests that iop tests derived based on the bilateral iop criterion will detect the same non-interoperability situations as tests generated using classical method. Moreover, the bilateral method (see Figure 1(b)) avoids the calculation of the specification interaction.

General principles and verdict management Let us consider an iop test purpose TP, as described in Section 3.1. The bilateral method can be decomposed in two main steps: cf. Figure 1(b). The first step of the bilateral method is the derivation of two unilateral iop test purposes TP_{S_i} from the global interoperability test purpose TP. Each TP_{S_i} contains only events of S_i and represents the iop test purpose TP in the point of view of S_i. The second step is the unilateral test case derivation. For this step, we can use a conformance test generation tool $\mathcal{F}$ such that $\mathcal{F} : (S_i, TP_{S_i}) \rightarrow TC'_i$, $i \in \{1, 2\}$. The unilateral test cases TC_i are obtained from TC'_i after some modifications due to differences in interface controllability between conformance and interoperability contexts.
As bilateral and global iop criteria are equivalent in terms of non-interoperability detection, we have: $verdict(TC, I_1 \| _{\mathcal{A}} I_2) = verdict(TC_1, I_1 \| _{\mathcal{A}} I_2) \wedge verdict(TC_2, I_1 \| _{\mathcal{A}} I_2)$. The verdicts $verdict(TC, I_1 \| _{\mathcal{A}} I_2)$, $verdict(TC_1, I_1 \| _{\mathcal{A}} I_2)$ and $verdict(TC_2, I_1 \| _{\mathcal{A}} I_2)$ are interoperability verdicts; $verdict(TC, I_1 \| _{\mathcal{A}} I_2)$ is a global interoperability verdict and the two others are unilateral verdicts. The rules for the combination of these two unilateral verdicts to obtain the final bilateral iop_B verdict are obvious: $PASS \wedge PASS = PASS$, $PASS \wedge INC = INC$, $PASS \wedge FAIL = FAIL$, $INC \wedge FAIL = FAIL$, $INC \wedge INC = INC$ and $FAIL \wedge FAIL = FAIL$.

Unilateral interoperability test purposes derivation The algorithm of figure 2 shows how to derive two unilateral interoperability test purposes from one global interoperability test purpose. Let us consider an event μ of the test purpose. If μ is an event of the considered specification, it is added to the test purpose. If μ is an event from the other specification, there is two possibilities. Either μ is an event to be executed on lower interfaces: in this case, the mirror event $\bar{\mu}$ is added. Either the event is an event to be executed on the upper interfaces: in this case, the algorithm searches a predecessor of μ, such that this predecessor is an event to be executable on lower interfaces. The algorithm adds the mirror of this predecessor to the test purpose.

Input: S_1, S_2: specification, TP: iop test purpose; **Output:** $\{TP_{S_i}\}_{i=1,2}$;
Invariant: $S_k = S_{3-i}$ (* S_k is the other specification *); $TP = \mu_1 ... \mu_n$
Initialization: $TP_{S_i} = \epsilon \; \forall i \in \{1, 2\}$;
for ($j = 0$;$j \leq n$;j++) **do**
  if ($\mu_j \in \Sigma_L^{S_i}$) **then** $TP_{S_i} = TP_{S_i}.\mu_j$; $TP_{S_k} = TP_{S_k}.\bar{\mu_j}$
  if ($\mu_j \in \Sigma_L^{S_k}$) **then** $TP_{S_i} = TP_{S_i}.\bar{\mu_j}$; $TP_{S_k} = TP_{S_k}.\mu_j$
  if ($\mu_j \in \Sigma_U^{S_i}$) **then** $TP_{S_i} = TP_{S_i}.\mu_j$;
    TP_{S_k}=add_precursor(μ_j, S_i, TP_{S_k})
  if ($\mu_j \in \Sigma_U^{S_k}$) **then** $TP_{S_k} = TP_{S_k}.\mu_j$;
    TP_{S_i}=add_precursor(μ_j, S_k, TP_{S_i})
  if ($\mu_j \notin \Sigma^{S_k} \cup \Sigma^{S_i}$)**then** error(TP not valid : $\mu_j \notin \Sigma^{S_1} \cup \Sigma^{S_2}$)

function add_precursor(μ, S, TP): **return** TP
  $\sigma_1 := TP$; a_j =last_event(σ_1)
  while $a_j \in \Sigma_U^S$ **do** σ_1=remove_last(σ_1);
    a_j =last_event(σ_1) **end**
  $M = \{q \in Q^S$; $\exists$ q'|(q, $\bar{a}_j, q') \wedge \sigma = \bar{a}_j.\omega.\mu \in Traces(q)\}$
  if ($\forall q \in M$, $\sigma \notin Traces(q)$) **then** error(no path to μ)
  while (e=last_event(ω) $\notin \Sigma_L^S \cup \{\epsilon\}$) **do** ω=remove_last(ω)
  if (e $\in \Sigma_L^S$) **then** $TP_S = TP_{S_i}.\bar{e}$ **end**

Fig. 2. Algorithm to derive TP_{S_i} from TP

Some additional functions are used in the algorithm of figure 2. Let us consider a trace σ and an event a. The function *remove_last* is defined by : remove_last($\sigma.a$) = σ. The function *last_event* is defined by : last_event(σ)= ϵ if $\sigma = \epsilon$ and last_event(σ)= a if $\sigma = \sigma_1.a$. The *error* function returns the cause of the error and exits the algorithm.

Deriving unilateral interoperability test cases The second step of this method is the derivation of two unilateral interoperability test cases based on the obtained test purposes. For this, a conformance test generation tool is used. It takes as inputs a specification S_1 (resp. S_2) and the corresponding test purpose TP_{S_1} (resp. TP_{S_2}) and generates two conformance test cases TC'_1 and TC'_2 that are modified in order to obtain the unilateral iop test cases TC_1 and TC_2. These unilateral interoperability test cases will be executed unilaterally on the

corresponding IUT in the SUT.
The modifications on TC'_1 and TC'_2 to obtain TC_1 and TC_2 are realized to take into account the differences between upper and lower interfaces in interoperability testing. For example, an event $l!m$ (resp. $l?m$) in the obtained test case will be replaced by $?(l?m)$ (resp. $?(l!m)$) in the interoperability test case. This means that the unilateral interoperability tester observes that a message m is received from (resp. sent to) the other IUT on the lower interface l. No changes are made on the test cases for events on the upper interfaces as these interfaces are observable and controllable: a message can be sent (and received) by the tester to the IUT on these interfaces.

Some words about complexity The first step of this method (algorithm of Figure 2) is linear in the maximum size of specifications. Indeed, the first part of this algorithm is linear as it is a simple traversal of the test purpose graph which is a small automaton compared to a specification graph. The other part of the algorithm (search of predecessor - only if the test purpose event is an event to be executed on a upper interface) is only linear as it is also a simple path search and is based on a stack structure.
The second step corresponds to the test generation. It uses a conformance test generation tool. In our case, we use TGV tool. As TGV [3] is linear in complexity, this step of the method is also linear in complexity.
Thus, the bilateral method costs less than the calculation of $S_1 \|_{\mathcal{A}} S_2$ needed for classical method. Moreover, if an iop test case can be obtained using classical approach, the bilateral method can generate an equivalent bilateral iop test case.

Causal dependency algorithm One objective of interoperability is to verify the correctness of the communication between the IUTs. Thus, iop test purposes may end with an input. This latter situation occurs in the unilateral test purposes derived by bilateral method. For example, if the iop test purpose ends with an output on lower interface, its mirror event (an input) is added -as last event- to one of the derived test purpose. The unilateral test case derivation generate a test case for which the PASS verdict is affected to an input (a non-observable event). An algorithm based on causal dependencies is used to complete bilateral method. The purpose of this completion is to produce outputs that help in verifying that the input is actually received by the corresponding IUT. The algorithm computes the set of causal dependency events (associated with the paths to these events), based on breadth-first search algorithms of the graph theory. It can also be used for refining interoperability test cases generated by classical methods based on test purpose ending with an input.

4 Applying the new method to a connection protocol

4.1 A simplified version of the ISDN connection protocol

ISDN (Integrated Services Digital Network) is a set of CCITT/ITU standards for digital transmission over ordinary telephone wire as well as over other media.

This protocol requires a connection. The IUT-T recommendation Q.920 [11] contains the description of the state diagram for sequences of primitives at a point-to-point data link connection endpoint. The specifications of Figure 3 consider a simplified version of this connection protocol. This version is simplified so that it could represent any other communication protocol with request-acknowledge connection negotiation. Two modes are possible: a client/server mode (S_1 and S_2 of Figure 3) or a complete client and server mode (specification S of Figure 3).

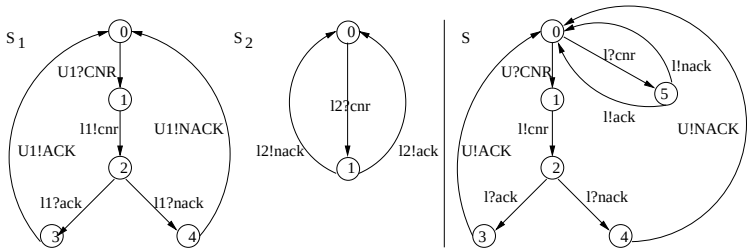

Fig. 3. Examples of specifications: S_1, S_2 and S

Specifications Let us describe S_1 and S_2 of Figure 3. $U1?CNR$ is a connection request from the upper layer, $l1!cnr$ (resp. $l2?cnr$) the request sent (resp. received) to the peer entity, $l2!ack/l2!nack$ the positive or negative response, and $U1!ACK/U1!NACK$ the response forwarded to the upper layer.

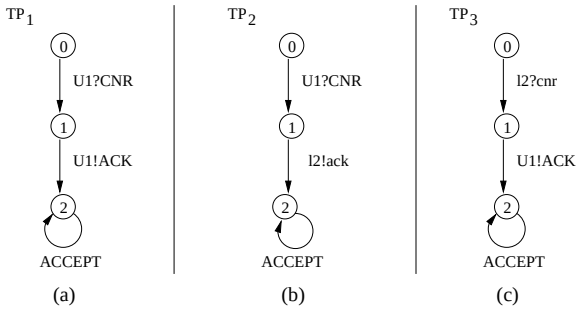

Fig. 4. Test Purpose examples: TP_1, TP_2 and TP_3

Test purposes A test purpose is an informal description of behaviors to be tested, in general an incomplete sequence of actions. Each state of the test purpose considers a particular event to be executed, but each state allows all actions.

Let us consider the three iop test purposes of figure 4. These iop test purposes are applicable to the System Under Test (SUT) composed of two IUTs implementing respectively S_1 and S_2. For example, TP_1 of Figure 4(a) (resp. TP_2 of Figure 4(b)) means that, after the reception by I_1 (implementing S_1) of a connection request on its upper interface $U1$, this IUT I_1 (resp. I_2) must send a connection acknowledgment on its upper interface $U1$ (resp. a connection acknowledgment on its lower interface $l2$).
The three test purposes of figure 4 are also applicable for deriving interoperability test cases executable on a SUT composed of two IUTs implementing the specification S (the complete client and server mode).

4.2 CADP Toolbox used for implementing the method

Both classical and new (bilateral) methods were implemented into the CADP toolbox [10]. CADP is a toolbox for the design of communication protocols and distributed systems. It includes tools for explicit state space manipulation called BCG. BCG (Binary-Coded Graphs) is both a format for the representation of explicit LTSs and a collection of libraries and programs dealing with this format. The BCG format is used for representing specifications and test purposes and for manipulating these LTSs.
One step of the bilateral method is the generation of unilateral interoperability test cases using a conformance test generation tool. In order to automatize conformance test generation, different test tools were developed: TorX [4], TVeda [17], SAMSTAG [18], TGV [3], etc. The conformance tool used in this study is TGV (Test Generation using Verification techniques). TGV is integrated in the CADP toolbox and can take as entries a specification and a test purpose in the BCG format. This conformance test tool is used for the unilateral interoperability test case generation of the bilateral method, but also for the global test case generation of the classical approach with the specification interaction and a test purpose as entries.

4.3 Applying the classical approach on the client/server mode

Specification interaction The first step of classical methods is the calculation of a model of the system behavior, that is to say the calculation of the specification interaction (called also reachability graph). For the interaction of S_1 and S_2 of Figure 3, we obtain the IOLTS of Figure 5.

Global interoperability test case derivation Based on the model of the behavior of the system composed of the two specifications and on the test purposes of Figure 4, we can derive interoperability global test cases. The TGV tool was used, taking as entries the specification interaction (Figure 5) and a test purpose (Figure 4). Results of this test derivation are shown in Figure 6. These test cases are those obtained after modifications due to controllability differences between interoperability and conformance contexts. Interface $UT1$ is the interface of the

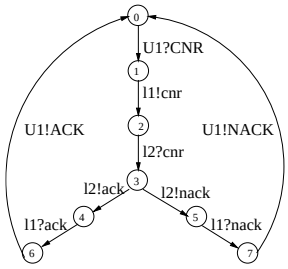

Fig. 5. Interaction of S_1 and S_2

tester connected to upper interface $U1$. Thus, $UT1!CNR$ means that a tester sends message CNR to the upper interface $U1$ of I_1. For events on lower interfaces, $?(\mu)$ corresponds to the observation of the event. However, inputs can only be deducted from the corresponding output (sent by the other IUT) and causal-dependency events.
These results are compared with test cases derived for the same test purposes by the bilateral method in next Section.

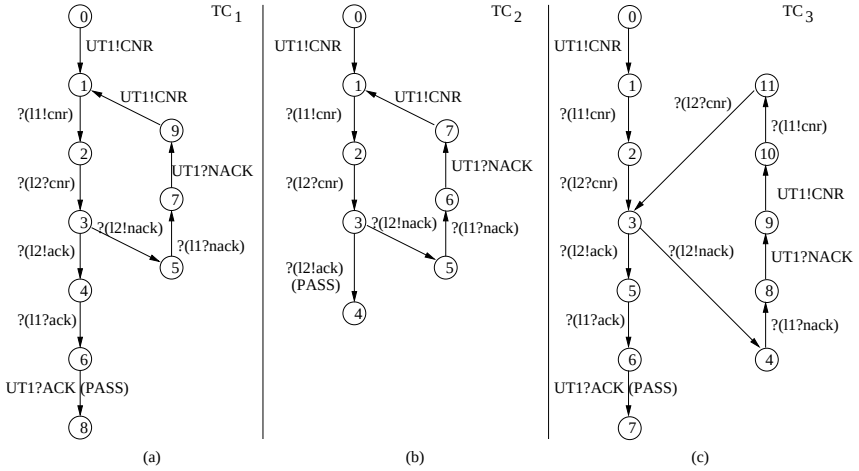

Fig. 6. Test Cases for the interaction of S_1 and S_2

4.4 Applying our new method on the client/server version

Unilateral test purpose derivation The first step of the bilateral interoperability test generation method is the derivation of the iop global test purposes into two unilateral iop test purposes. The implemented algorithm corresponds to the algorithm presented in Figure 2.

Applying this step to the iop test purposes of figure 4 and specifications S_1 and S_2 gives as result the unilateral iop test purposes of Figure 7. TP_1^1 and TP_1^2 of Figure 7(a) are the test purposes derived for TP_1 (Figure 4(a)) and respectively specifications S_1 and S_2. In the same way, TP_2^1 and TP_2^2 of Figure 7(b) (resp. TP_3^1 and TP_3^2 of Figure 7(c)) are derived from TP_2 of Figure 4(b) (resp. TP_3 of Figure 4(b)). The same notation will be used for test cases in the following.
When deriving the unilateral iop test purposes, for events on lower interfaces, the returned event is either the event itself, either its mirror. For event $U1!ACK$, as its predecessor is $\mu = l1?ack$, the returned event is $\bar{\mu} = l2!ack$ (TP_1^2 and TP_3^2) or $U1!ACK$ (TP_1^1 and TP_3^1). The difficulty is for deriving an event from $U1?CNR$ for TP_1^2 (Figure 7(a)) and TP_2^2 (Figure 7(b)). In S_1, this event is the first possible event after the initial state. Its predecessor must be found in the paths bringing back the entity in its initial state. The first predecessor found is $U1!NACK$. As this event is not an event of the interaction, the algorithm continues one more step to find $l1?nack$ as predecessor, and then returns $l2!nack$ (mirror of $l1?nack$).

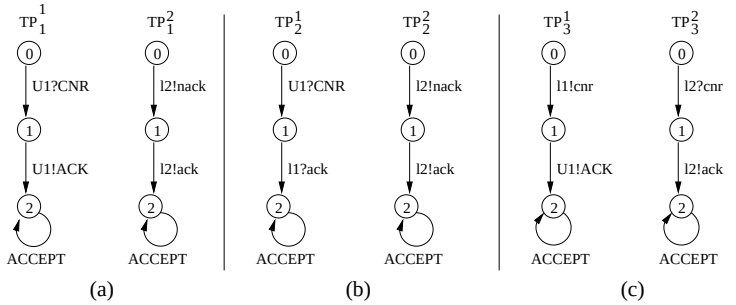

Fig. 7. Unilateral Test Purpose derived for specifications S_1 and S_2

Unilateral test case derivation The second step of the bilateral interoperability test generation method corresponds to the use of a conformance test tool (here TGV) on a unilateral test purpose and the corresponding specification. TGV will return conformance test cases that we want to reuse in interoperability context after some modifications.
A test case is controllable if the tester does not need to choose arbitrarily between different events. In conformance, inputs on lower interfaces correspond to outputs of the tester: a controllable conformance test case only considers one of the possible inputs on lower interfaces. In interoperability testing, inputs on lower interfaces are sent by the other implementation. An interoperability test case must take into account *all* possible inputs on lower interfaces. The complete test graph is an IOLTS which contains all sequences corresponding to a test purpose: all the inputs of the implementation that correspond to the test purposes

are considered in this IOLTS. Thus, to have test cases usable in interoperability context, the conformance tool used in this step (like TGV) for interoperability test generation must compute the complete test graph.

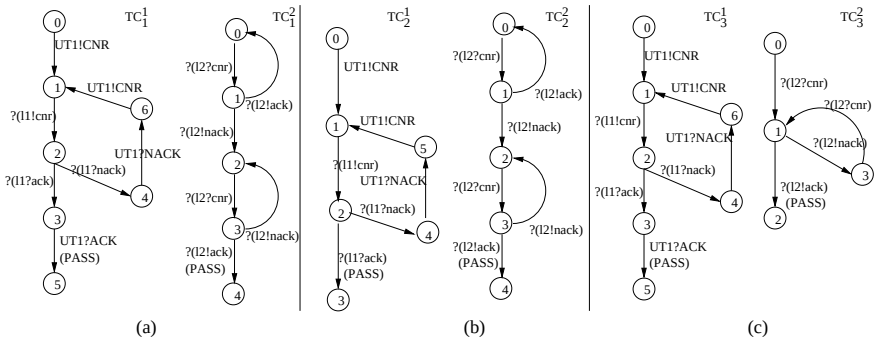

Fig. 8. Unilateral Iop Test Cases for specifications S_1 and S_2

The main modifications to be applied to the obtained conformance test cases concern the types of messages. The messages on lower interfaces are observations in interoperability testing whereas they corresponds to communication with the tester in conformance testing. The results on Figure 8 gives the test cases modified for interoperability.

Interoperability "scenario" and comparison with classical methods If we calculate the interaction of two unilateral test cases, we obtain the interoperability test case execution scenarios of Figure 9. The scenario obtained for TP_3 (Figure 9(c)) contains only one deadlock (state 7). This state corresponds to a $PASS$ verdict (PASS_1$\wedge$PASS_2=$PASS$).
The scenario obtained for TP_1 (Figure 9(a)) contains two deadlocks: states 15 and 7. State 15 corresponds to a PASS state. But state 7 is only to a verdict state (PASS_1) of TC_1^1, not to a verdict state of TC_1^2 (see test cases on Figure 8(a)). This means that, in this state, TC_1^1 is executed until a verdict state, but TC_1^2 has not reached a such state. The part of TC_1^2 not executed was generated because of the test purpose derivation (calculation of the predecessor of $U1?CNR$ when deriving TP_1^2). We can also remark that the trace executed until state 7 of $TC_1^1 \| _{\mathcal{A}} TC_1^2$ verifies the global test purpose TP_1. Thus, even though a deadlock state without both unilateral verdicts exists, the obtained scenario is complete regarding the iop test purpose TP_1.
For TP_2 (scenario on Figure 9(c)), the obtained interoperability test cases end with an input because the unilateral test purpose generated for S_1 ends with an input. To complete these iop test cases (TC_2^1 and $TC_2^1 \| _{\mathcal{A}} TC_2^2$), we can either add a postamble returning to the initial state, either use causal dependency algorithm (using breadth-first search algorithms). It will add paths until outputs

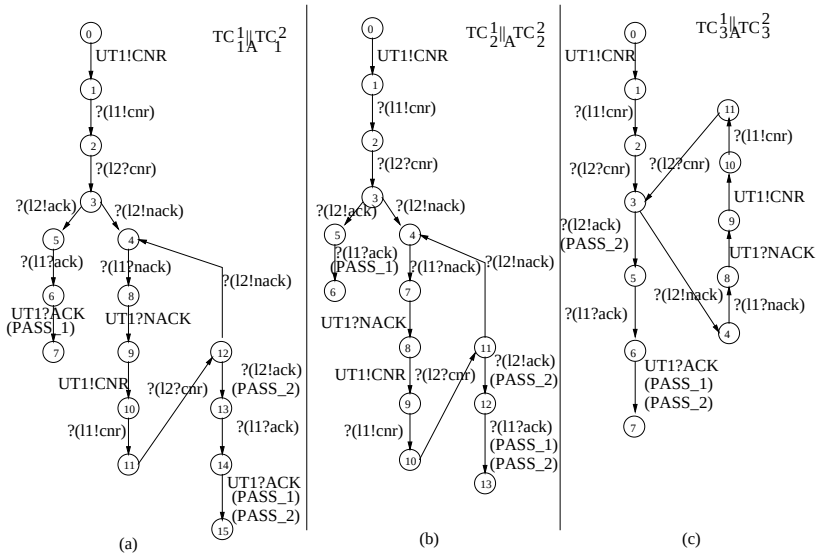

Fig. 9. Test Case interactions

that are executed only if the input $l1?ack$ is actually executed. In this simple example (specification S_1), only the event $U1!ACK$ (or $UT1?ACK$ for the tester) will be added with causal dependency event method.

We can observe that the global iop test case generated for TP_3 (Figure 6(c)) corresponds to the scenario obtained by the interaction of the unilateral test cases generated for this iop test purpose (Figure 9(c)). For TP_1 and TP_2 (global test cases on Figures 6 (a) and (b) and scenarios on Figure 9 (a) and (b)), there are more branches. But a look at glance on the traces contained in both the global test case and the scenario from bilateral method shows that they are equivalent in terms of verdicts. Indeed, the same execution paths lead to the same verdicts. These examples confirm the equivalence of both classical (global) and new (bilateral) methods in terms of non-interoperability detection.

4.5 Application to the complete client and server mode

Both methods were also applied on the specification S describing both client and server parts and using the same test purposes. The interaction $S\|_{\mathcal{A}}S$, calculated for classical approach, is composed of 454 states and 1026 transitions with input queues of each system bounded to one message. Results in the following table are given for a queue size of 3. The table gives the number of states s and transitions t (noted s/t in the table) for different test cases. The first two lines correspond to iop test cases derived with the bilateral method (S as specification 1 and 2) and the third line to the interaction of these test cases. The last line gives results for test cases derived with classical method. For this method, the

generated specification interaction has 47546 states and 114158 transitions.

	TP_1	TP_2	TP_3
S as spec. 1 (bilateral method)	9/17	8/16	9/17
S as spec. 2 (bilateral method)	13/24	13/24	12/22
$TC^1 \|\|_{\mathcal{A}} TC^2$	19546/57746	19468/57614	19405/57386
$S \|\|_{\mathcal{A}} S$ (global method)	54435/120400	18014/40793	54456/120443

We observe that we can derive unilateral test cases via the bilateral method. These test cases can be used for executing interoperability test cases. For classical (global) methods, we faced the state space explosion problem. Indeed, we were not able to compute $S\|_{\mathcal{A}}S$ for a queue size limited to 4 messages (on a system with a 2GHz processor and 1 Gb of memory). This shows that the bilateral method can be used to generate iop test cases even for specifications that produce state space explosion problem. Moreover, these test cases are not dependent of the queue size.

4.6 Summary of the experimentation results

The result on the examples can be summarized as follows.

1. In terms of non-interoperability detection, the obtained iop test cases confirm the equivalence of the bilateral and global iop criteria that was proved theoretically in [9].
2. The causal-dependency based algorithm can be used to complete iop test cases generated with both global and bilateral method, particularly when we have test purposes ending with inputs.
3. The bilateral method can be used to generate interoperability test cases even for specifications that produce state space explosion problem with classical methods.

5 Conclusion

In this paper, we present a new method for generating interoperability test cases. The interoperability criterion on which the presented method is based was proved equivalent in terms of non-interoperability detection to another interoperability criterion on which classical methods are generally based. This equivalence was confirmed by experimental results. Moreover, we show that the so-called bilateral interoperability test derivation method allows up to generate interoperability test cases in situations where it would have been impossible with the traditional methods because of state space explosion problem.
As future work, we will study the generalization of the formal interoperability definitions and test generation methods to a context with more than two implementations. We will also study how to apply the described method to a distributed testing architecture.

References

[1] ISO. Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework - Parts 1-7. *International Standard ISO/IEC 9646/1-7*, 1992.

[2] J. Tretmans. Testing concurrent systems: A formal approach. In J.C.M Baeten and S. Mauw, editors, *CONCUR'99 – 10^{th} Int. Conference on Concurrency Theory*, volume 1664 of *LNCS*. Springer-Verlag, 1999.

[3] C. Jard and T. Jéron. Tgv: theory, principles and algorithms, a tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Software Tools for Technology Transfer (STTT)*, October 2004.

[4] J. Tretmans and E. Brinksma. Torx: Automated model based testing. In A. Hartman and K. Dussa-Zieger, editors, *Proceedings of the First European Conference on Model-Driven Software Engineering*, Nurnberg, Germany, December 2003.

[5] A. Desmoulin and C. Viho. Quiescence Management Improves Interoperability Testing. In *17th IFIP International Conference on Testing of Communicating Systems (Testcom)*, Montreal, Canada, May-June 2005.

[6] R. Castanet and O. Koné. Deriving coordinated testers for interoperability. In O. Rafiq, editor, *Protocol Test Systems*, volume VI C-19, pages 331–345, Pau-France, 1994. IFIP, Elsevier Science B.V.

[7] S. Seol, M. Kim, S. Kang, and J. Ryu. Fully automated interoperability test suite derivation for communication protocols. *Comput. Networks*, 43(6):735–759, 2003.

[8] K. El-Fakih, V. Trenkaev, N. Spitsyna, and N. Yevtushenko. Fsm based interoperability testing methods for multi stimuli model. In R. Groz and R. Hierons, editors, *TestCom*, Lecture Notes in Computer Science. Springer, 2004.

[9] A. Desmoulin and C. Viho. Formalizing interoperability for test case generation purpose. In *IEEE ISoLA Workshop on Leveraging Applications of Formal Methods, Verification, and Validation*, Columbia, MD, USA, September 2005.

[10] H. Garavel, F. Lang, and R. Mateescu. An overview of cadp 2001. Technical Report 0254, INRIA, 2001.

[11] ITU-T. Digital Subscriber Signalling System No.1 - ISDN User-Network Interface Data Link Layer - General Aspects. *ITU-T Recommandation Q.920*, 1993.

[12] L. Verhaard, J. Tretmans, P. Kars, and E. Brinksma. On asynchronous testing. In G.V. Bochman, R. Dssouli, and A. Das, editors, *Fifth inteernational workshop on protocol test systems*, pages 55–66, North-Holland, 1993. IFIP Transactions.

[13] R. Castanet and O. Kone. Test generation for interworking systems. *Computer Communications*, 23:642–652, 2000.

[14] Nancy D. Griffeth, Ruibing Hao, David Lee, and Rakesh K. Sinha. Integrated system interoperability testing with applications to voip. In *FORTE/PSTV 2000*, pages 69–84. Kluwer, B.V., 2000.

[15] G. Bochmann, R. Dssouli, and J. Zhao. Trace analysis for conformance and arbitration testing. *IEEE transaction on software engeneering*, 15(11):1347–1356, November 1989.

[16] J. Gadre, C. Rohrer, C. Summers, and S. Symington. A COS study of OSI interoperability. *Computer standards and interfaces*, 9(3):217–237, 1990.

[17] R. Groz and N. Risser. Eight years of experience in test generation from fdts using tveda. In *FORTE*, Osaka, Japan, November 1997.

[18] J. Grabowski, D. Hogrefe, and R. Nahm. Test case generation with test purpose specification by mscs. In *SDL Forum*, Amsterdam, North-Holland, October 1993.