Interoperability test generation : formal definitions and algorithm

Alexandra DESMOULIN — César VIHO

IRISA/Université de Rennes I
Campus de Beaulieu
35042 Rennes Cedex, FRANCE
{Alexandra.Desmoulin,Cesar.Viho}@irisa.fr

RÉSUMÉ. Le but de cet article est de fournir une méthode formelle pour la génération de tests d'interopérabilité. Contrairement aux travaux précédents, cette étude prend en compte les blocages des implémentations qui peuvent être observés durant un test d'interopérabilité. Ceci est réalisé via la notion de *critères d'interopérabilité*, qui donnent des définitions formelles des notions d'interopérabilité existantes. Il est tout d'abord prouvé que la gestion des blocages améliore la détection de la non-interopérabilité. L'équivalence de deux des critères est aussi prouvée permettant l'introduction d'une nouvelle méthode de génération de tests d'interopérabilité. Cette méthode permet d'éviter le problème d'explosion du nombre d'états existant lors de l'utilisation des approches classiques.

ABSTRACT. This study is aimed at providing a formal method for interoperability test generation. Contrary to previous work, this study takes into account quiescence of implementations that may occur during interoperability testing. This is done through the notion of *interoperability criteria* that give formal definitions of the different existing pragmatic interoperability notions. It is first proved that quiescence management improves non-interoperability detection. Two of these interoperability criteria are proved equivalent leading to a new method for interoperability test generation. This method avoids the well-known state explosion problem that may occur when using existing classical approaches.

MOTS-CLÉS : Interopérabilité, test, critère, génération de tests, blocage

KEYWORDS : Interoperability, test, criterion, test generation, quiescence

1. Introduction

Interoperability testing is still considered as a pragmatic issue. Indeed, it requires configurations of tested systems, specific parameterizations, etc. Results obtained may depend on these operations. Nevertheless, the same arguments applied for a long time to conformance testing. Yet, one can observe now that studies on formal approach for conformance testing has increased the knowledge in this area [1]. It allows automatic conformance test generation (TGV [2] for example) and improves quality of generated test cases. Despite a large literature on the interest of providing a formal approach for interoperability testing [3, 4], only few tentative have been proposed [5, 6].
The aims of the study presented here are double. First, we give formal definitions of interoperability testing. Contrary to a previous work [7], these definitions, called *interoperability criteria* (*iop criteria* in the following) manage quiescence. The second contribution of this work is a new method to generate automatically interoperability test cases. It uses a theorem proving the equivalence between two iop criteria and avoids the construction of the specification interaction that may lead to the well-known state-explosion problem [3].
The paper is structured as follows. Section 2 gives the considered interoperability testing architectures. Models and the formal background used in this paper are in Section 3. The iop criteria are defined in Section 4. In Section 5, the proposed method and associated algorithms for interoperability test case generation are described. Results obtained are illustrated with a simple example. Conclusion and future work are in section 6.

2. One-to-one interoperability testing architecture

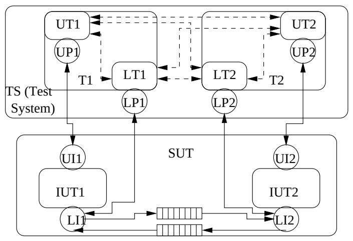

Figure 1. *Interoperability testing architecture*

In this study, we consider the one-to-one interoperability context (testing architecture in figure 1) : the System Under Test (SUT) is composed of two Implementation Under Test (IUT). The interaction between the two IUT is asynchronous (cf. Section 3.2).
There are two kind of interfaces : the Lower Interfaces LI_i and the Upper Interfaces UI_i. The interfaces LI_i, used for the interaction of the two IUT, are only observable but not controllable. A tester (LT_i) connected to such interfaces can only observe the events but not send a stimuli to the interfaces. The interfaces UI_i are the interfaces through which the IUT communicates with its environment. They are observable and also controllable by the tester UT_i. Depending on the access to the different interfaces, different architectures can be distinguished. For example, the interoperability testing architecture is called **unilateral** if only the interfaces of one IUT (of the two interacting IUT) are accessible, **bilateral** if the interfaces of both IUT are accessible but separately, or **global** if the interfaces of both IUT are accessible with a global view.

3. Formal background

3.1. IOLTS model

The well-known IOLTS (Input-Output Labeled Transition System) model [8] will be used to model specifications and to define interoperability criteria.

Definition 3.1 *An IOLTS is a tuple M = (Q^M,Σ^M,Δ^M, q_0^M) where*
• Q^M *is the set of states of the system and* $q_0^M \in Q^M$ *is the initial state.*
• Σ^M *denotes the set of observable (input and/or output) events on the system interfaces.* $p?m$ *stands for an input and* $p!m$ *for an output where* p *is the interface and* m *the message.*
• $\Delta^M \subseteq Q^M \times (\Sigma^M \cup \tau) \times Q^M$ *is the transition relation (*$\tau \notin \Sigma^M$ *is an internal event).*

Σ^{M} can be decomposed : $\Sigma^{\mathrm{M}} = \Sigma_U^{\mathrm{M}} \cup \Sigma_L^{\mathrm{M}}$, where Σ_U^{M} (resp. Σ_L^{M}) is the set of messages exchanged on the upper (resp. lower) interface. Σ^{M} can also be decomposed to distinguish input (Σ_I^{M}) and output (Σ_O^{M}) messages. Let us consider an IOLTS M, $\sigma \in (\Sigma^{\mathrm{M}})^*$, $q \in Q^{\mathrm{M}}$:
• $\Gamma(q)$ is the *set of observable events* from q, and $out(q)$ the *set of outputs* from q.
• q *after* σ is the set of states which can be reached from q by the sequence of actions σ. By extension, all the *states reached from the initial state* of the IOLTS M is (q_0^{M} *after* σ), noted by (M *after* σ). In the same manner, $Out(M, \sigma) = out(M$ *after* $\sigma)$.
• $Traces(q)$ is the *set of possible observable traces* from q. $Traces(M) = Traces(q_0^{\mathrm{M}})$.
• if we consider a link l between two IUT i and j, $\bar{\mu}$= $l_i!a$ if μ = $l_j?a$ and $\bar{\mu}$ = $l_i?a$ if μ = $l_j!a$. For internal events, $\bar{\tau}$ = τ.

Three situations lead to quiescence of a system : *deadlock* (a state after which no event is possible), *outputlock* (a state after which only transitions labeled with input exist) and *livelock* (a loop of internal events). Quiescence is modeled by δ and treated as an observable output event. The obtained IOLTS is called suspensive IOLTS [1] and noted $\Delta(M)$. Figure 2 gives an example of two specifications using the IOLTS model. $U?A$ corresponds to a request from the upper layer of S_2. Then, S_1 and S_2 may exchange some messages via their lower layer. The resulting response to the request can be either positive (output on the upper interface $U!R$) or negative (output $U!N$). Quiescence is possible and modeled on these specifications (see in states 0 and 3 of S_1, 0, 3 and 4 of S_2).

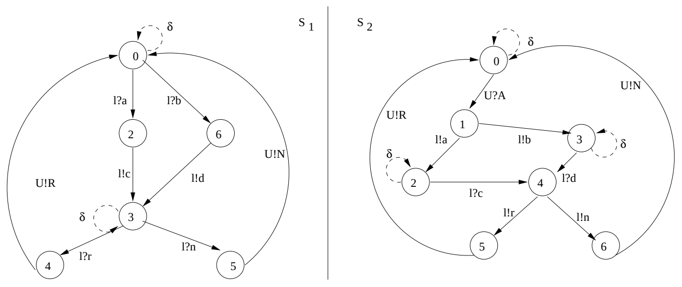

Figure 2. *Specifications* S_1 *and* S_2

3.2. Operations : interaction and projection

Interoperability testing concerns the interaction of two or more implementations. To provide a formal definition of interoperability, we need to model the different interactions.

To calculate the interaction of two IOLTS with quiescence management, we calculate first the suspensive IOLTS $\Delta(M_1)$ and $\Delta(M_2)$, and then we construct the interaction of $\Delta(M_1)$ and $\Delta(M_2)$, using rules of the synchronous interaction [9]. Quiescence is preserved and $\delta(i)$ (resp. δ) corresponds to quiescence of M_i (resp. of the two IOLTS).

As the interaction between the two IUT is asynchronous in the considered interoperability testing architecture, we also need to model asynchronous interaction. As in [8], we can model the asynchronous environment with FIFO queues. The asynchronous transformation applied to a specification S gives as result the IOLTS $\mathcal{A}(S)$ representing the behavior of S in an asynchronous environment. As consequence, the **asynchronous interaction** of M_1 and M_2 is the synchronous interaction of $\mathcal{A}(M_1)$ and $\mathcal{A}(M_2)$, noted $M_1\|_{\mathcal{A}}M_2$.

In interoperability testing, we usually need to observe some specific events of an IUT. The IUT, reduced to the expected messages, can be obtained by a **projection** of the IOLTS representing the whole behavior of the implementation on a set (called X in the following and used to select the expected events). This is noted by M/X and is obtained by hiding events (replacing by τ-transitions) that do not belong to X, followed by determinization.

3.3. Model of an implementation : iop-input completion

As usual in the black-box testing context, we need to model implementations, even if their behaviors are unknown. As described in figure 1, the two IUT interact asynchronously and testers are connected to their interfaces. But these testers can not differentiate events received by an IUT from events effectively treated. To model this behavior, we choose to complete any IUT with inputs corresponding to the output alphabet of the other IUT specification. These transitions lead the IOLTS into an error state. It is a deadlock state. On the upper interfaces, the IUT interacts directly with the tester (like in a conformance testing context). Thus, for events on the upper interfaces, the input-completion of the IUT corresponds to the input completion made for conformance testing. In the following, the IUT are considered iop-input completed with quiescence modeled.

4. Interoperability criteria

According to the different possible testing architecture (see Section 2), different notions of interoperability can be used [7]. In this section, we introduce interoperability notions called *interoperability (iop) criteria*. Thanks to quiescence management, these criteria detect more non-interoperability cases than the "interoperability relations" defined in [7]. Moreover, we prove the equivalence between the most commonly used in practice iop criterion iop_G and the so called bilateral iop criterion iop_B. We will describe here the criteria considering events on both kinds of interface.

The **unilateral total iop criterion** says that after a suspensive trace of S_1 observed during the (asynchronous) interaction of the implementations, all outputs and quiescence observed in I_1 are foreseen in S_1. Formally : $\forall\sigma_1 \in Traces(\Delta(S_1)), \forall\sigma \in Traces(S_1\|_{\mathcal{A}}S_2)$, $\sigma/\Sigma^{S_1} = \sigma_1 \Rightarrow Out((I_1\|_{\mathcal{A}}I_2)/\Sigma^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1), \sigma_1)$.

The **bilateral total iop criterion** is verified iff both unilateral total criteria are verified.

Definition 4.1 (The bilateral iop criterion iop_B) *Let I_1, $I_2 \in \mathcal{IOLTS}$ two IUT implementing respectively S_1, $S_2 \in \mathcal{IOLTS}$. I_1 iop_B $I_2 =_\Delta$*

$$\forall\sigma_1 \in Traces(\Delta(S_1)), \forall\sigma \in Traces(S_1\|_{\mathcal{A}}S_2), \sigma/\Sigma^{S_1} = \sigma_1 \Rightarrow$$
$$Out((I_1\|_{\mathcal{A}}I_2)/\Sigma^{S_1},\sigma_1) \subseteq Out(\Delta(S_1),\sigma_1)$$
and $$\forall\sigma_2 \in Traces(\Delta(S_2)), \forall\sigma' \in Traces(S_2\|_{\mathcal{A}}S_1), \sigma/\Sigma^{S_2} = \sigma_2 \Rightarrow$$
$$Out((I_2\|_{\mathcal{A}}I_1)/\Sigma^{S_2},\sigma_2) \subseteq Out(\Delta(S_2), \sigma_2).$$

The **global total iop criterion** says that two implementations are considered interoperable if, after a suspensive trace of the asynchronous interaction of the specifications, all outputs and quiescence observed during the (asynchronous) interaction of the implementations are foreseen in the specifications.

Definition 4.2 (The global iop criterion iop_G) *Let I_1, $I_2 \in \mathcal{IOLTS}$ two IUT implementing respectively S_1, $S_2 \in \mathcal{IOLTS}$.*

$I_1\ iop_G\ I_2 =_\Delta \forall \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2),\ Out(I_1\ \|_{\mathcal{A}} I_2, \sigma) \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$

Comparisons between iop criteria are developed in [9]. The most important result is the following theorem 4.1 stating that iop_G and iop_B are equivalent. This theorem is proved with help of three intermediate lemmas (where M_1 and M_2 are two IOLTS).

Theorem 4.1 $I_1\ iop_G\ I_2 \Leftrightarrow I_1\ iop_B\ I_2$

Lemma 4.1 *Let $\sigma \in Traces(M_1 \|_{\mathcal{S}} M_2)$,*
$Out(M_1 \|_{\mathcal{A}} M_2, \sigma) = Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$.

Proof : **1)** Let $(q_1, q_2) \in [(M_1 \|_{\mathcal{A}} M_2) after \sigma]$ and $a \in Out(M_1 \|_{\mathcal{A}} M_2, \sigma)$. According to the interaction definition, either $a \in \Sigma^{M_1} \cup \{\delta, \delta(1)\}$ (i.e. $a \in Out(\Delta(M_1),\ \sigma/\Sigma^{M_1})$) or $a \in \Sigma^{M_2} \cup \{\delta, \delta(2)\}$ (i.e. $a \in Out(\Delta(M_2),\ \sigma/\Sigma^{M_2})$).
$\Rightarrow Out(M_1 \|_{\delta} M_2, \sigma) \subseteq Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$.
2) In the other sense, it is easy to see that : $Out(M_1 \|_{\mathcal{A}} M_2, \sigma) \subseteq Out(\Delta(M_1),\ \sigma/\Sigma^{M_1})$ $\cup\ Out(\Delta(M_2), \sigma/\Sigma^{M_2})$. $\diamondsuit$

Lemma 4.2 $((M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1}) \|_{\mathcal{A}} ((M_2 \|_{\mathcal{A}} M_1)/\Sigma^{M_2}) = M_1 \|_{\mathcal{A}} M_2$.

Proof : **1)** Let $\sigma_1 \in Traces((M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1})$, $\sigma_2 \in Traces((M_2 \|_{\mathcal{A}} M_1)/\Sigma^{M_2})$, and $\sigma = \sigma_1 \|_{\mathcal{A}} \sigma_2 \in Traces(((M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1}) \|_{\mathcal{A}} ((M_2 \|_{\mathcal{A}} M_1)/\Sigma^{M_2}))$. We have : $\sigma_1 \in Traces(\Delta(M_1))$ and $\sigma_2 \in Traces(\Delta(M_2))$. Thus, $\sigma = \sigma_1 \|_{\mathcal{A}} \sigma_2 \in Traces(M_1 \|_{\mathcal{A}} M_2)$.
2) Let $\sigma \in Traces(M_1 \|_{\mathcal{A}} M_2)$ such that $\sigma = \sigma_1 \|_{\mathcal{A}} \sigma_2$ with $\sigma_1 \in Traces(\Delta(M_1))$ and $\sigma_2 \in Traces(\Delta(M_2))$. We have $\sigma_1 = \sigma/\Sigma^{M_1}$ and $\sigma_2 = \sigma/\Sigma^{M_2}$.
Thus $\sigma_1 \in Traces((M_1 \|_{\mathcal{A}} M_2/\Sigma^{M_1}))$, $\sigma_2 \in Traces((M_2 \|_{\mathcal{A}} M_1/\Sigma^{M_2}))$
and $\sigma = \sigma_1 \|_{\mathcal{A}} \sigma_2 \in Traces(((M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1}) \|_{\mathcal{A}} ((M_2 \|_{\mathcal{A}} M_1)/\Sigma^{M_2}))$. $\diamondsuit$

Lemma 4.3 *Let $\sigma_1 \in Traces(\Delta(M_1))$, $\sigma \in Traces(M_1 \|_{\mathcal{A}} M_2)$ and $\sigma_1 = \sigma/\Sigma^{M_1}$. $Out((M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1}, \sigma_1) \subseteq Out(\Delta(M_1), \sigma_1)$.*

Proof : $(M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1}$ is an IOLTS composed of events from $\Sigma^{(M_1 \|_{\mathcal{A}} M_2)}/Sigma^{M_1} \cup \{\delta\} \subseteq \Sigma^{M_1} \cup \{\delta\}$. $\diamondsuit$

Proof : (of the theorem 4.1) **1**) Let us prove first that $I_1\ iop_B\ I_2 \Rightarrow I_1\ iop_G\ I_2$.
Let $\sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $\sigma_1 \in Traces(\Delta(S_1))$ such that $\sigma_1 = \sigma/\Sigma^{S_1}$, $\sigma_2 \in Traces(\Delta(S_2))$ such that $\sigma_2 = \sigma/\Sigma^{S_2}$. Thus, $Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}, \sigma/\Sigma^{S_1}) \subseteq Out(\Delta(S_1), \sigma/\Sigma^{S_1})$ and $Out((I_2 \|_{\mathcal{A}} I_1)/\Sigma^{S_2}, \sigma/\Sigma^{S_2}) \subseteq Out(\Delta(S_2), \sigma/\Sigma^{S_2})$.
Using the lemma 4.1, $Out[((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1} \|_{\mathcal{A}} (I_2 \|_{\mathcal{A}} I_1)/\Sigma^{S_2}), \sigma] \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$.
With the lemma 4.2, $Out(I_1 \|_{\mathcal{A}} I_2, \sigma) \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$. That means $I_1\ iop_G\ I_2$.
2) Let us prove now that $I_1\ iop_G\ I_2 \Rightarrow I_1\ iop_B\ I_2$.
Let I_1, I_2, S_1, S_2 such that $I_1\ iop_G\ I_2$. Let $\sigma_1 \in Traces(\Delta(S_1))$ such that $\sigma_1 = \sigma/\Sigma^{S_1}$ with $\sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$. Using the definition of $I_1\ iop_G\ I_2$, we have : $Out(I_1 \|_{\mathcal{A}} I_2, \sigma) \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$. Projecting this inclusion on Σ^{S_1} gives
$Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}, \sigma_1) \subseteq Out(\ (S_1 \|_{\mathcal{A}} S_2)\ /\Sigma^{S_1}, \sigma_1)$
Using the lemma 4.3, $Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1), \sigma_1)$. And using the fact that iop_G is symmetrical, we have also $I_1\ iop_G\ I_2 \Rightarrow Out((I_2 \|_{\mathcal{A}} I_1)/\Sigma^{S_2}, \sigma_2) \subseteq Out(\Delta(S_2), \sigma_2)$. That means $I_1\ iop_G\ I_2 \Rightarrow I_1\ iop_B\ I_2$. $\diamondsuit$

5. Interoperability test generation

5.1. Interoperability test generation : some generalities

The goal of an interoperability test generation algorithm is to generate interoperability Test Cases (TC) that can be executable on the SUT composed of the interacting IUT. Data used to generate test cases are the specifications and a test purpose. A test purpose TP is a particular property to be tested. It is defined with an incomplete sequence of actions that have to be observed or sent to the SUT. It supposes that any sequences of actions foreseen in the specification may occur between two consecutive actions of a test purpose.
In the tester point of view, two kinds of events are possible during conformance tests : sending a stimuli to the IUT or receiving an input. In interoperability testing, these events are possible only on the upper interfaces of the IUT. The main difference is that it is also possible to observe messages exchanged on the lower interfaces.
An iop test case TC will be represented by an extended IOLTS called T-IOLTS (Testing-IOLTS). A T-IOLTS TC is defined by $TC = (Q^{TC}, \Sigma^{TC}, \Delta^{TC}, q_0^{TC})$. $\{PASS, FAIL, INC\} \subseteq Q^{TC}$ 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 on a lower interface. Δ^{TC} is the transition function. The execution of an iop test case TC on $SUT(I_1\|_{\mathcal{A}}I_2)$ gives a verdict : $verdict(TC, SUT) \in \{PASS, FAIL, INC\}$. The iop verdict PASS means that no interoperability error was detected, FAIL means that the iop criterion is not verified, and INC (for Inconclusive) that the behavior of the SUT seems valid but it is not the purpose of the test case.
In the classical approach based on a criteria like iop_G (see figure 3(a)), the test generation algorithm begins with the construction of the asynchronous interaction $S_1 \|_{\mathcal{A}} S_2$. Then $S_1 \|_{\mathcal{A}} S_2$ is composed with the TP. The consistency of TP is checked in parallel and TC is generated. Yet, the construction of $S_1 \|_{\mathcal{A}} S_2$ can cause the well-known state-space explosion, as building $S_1 \|_{\mathcal{A}} S_2$ is exponential in the number of states of S_1 and S_2 and the FIFO queues size. Thus, interoperability test generation based on the global iop criterion may be impossible even for small specifications.

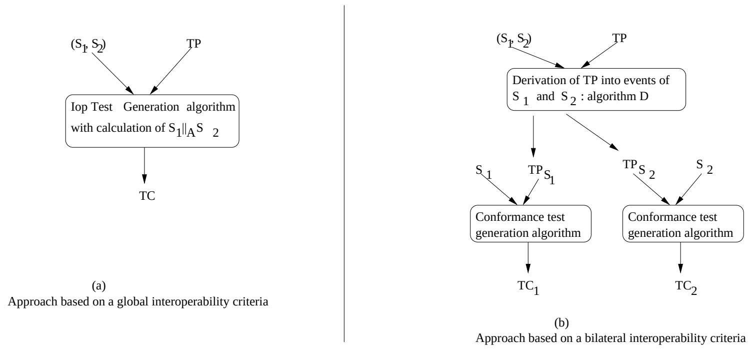
(S1, S2)
TP
Iop Test Generation algorithm with calculation of S1||A S2
TC
(a)
Approach based on a global interoperability criteria
(S1, S2)
TP
Derivation of TP into events of S1 and S2 : algorithm D
S1
TPS1
TPS2
S2
Conformance test generation algorithm
Conformance test generation algorithm
TC1
TC2
(b)
Approach based on a bilateral interoperability criteria

Figure 3. *Approaches for interoperability test generation*

5.2. Method based on the bilateral iop criterion iop_B

The equivalence of iop_B and iop_G (cf. theorem 4.1) suggests to study a method for iop test case generation based on iop_B. Let us consider an iop test purpose TP. The idea is to derive "unilateral test purposes" TP_{S_i} from TP. Each TP_{S_i} contains only events of S_i and represents TP in the point of view of S_i. The second step is to use a conformance

test generation tool $\mathcal{F}$ such that $\mathcal{F} : (S_1, TP_{S_1}) \rightarrow TC_1$ and $\mathcal{F} : (S_2, TP_{S_2}) \rightarrow TC_2$. According to the theorem 4.1, $verdict(TC,\ I_1 \parallel_{\mathcal{A}} I_2) = verdict(TC_1,\ I_1 \parallel_{\mathcal{A}} I_2) \wedge verdict(TC_2, I_1 \parallel_{\mathcal{A}} I_2)$. The rules for the combination of these verdicts to obtain the final iop_B verdict are : $PASS \wedge PASS = PASS$, $PASS \wedge INC = INC$, $INC \wedge INC = INC$, and $FAIL \wedge (FAIL \vee INC \vee PASS) = FAIL$.
1- The first step of the algorithm consists in deriving TP_{S_1} and TP_{S_2} from TP. Indeed, for $TP = \mu_1 ... \mu_{n-1}$, any traces foreseen in the specifications may occur to consecutive actions μ_i and μ_{i+1}. The way to derive TP_{S_1} and TP_{S_2} from TP is described in the following algorithm (see figure 4). If all the events described in TP are on the lower interfaces, the algorithm is very simple. If TP contains an event μ_i on the upper interfaces, this algorithm finds a path between μ_{i-1} (or $\bar{\mu}_{i-1}$) and μ_i in the interacting specification.

In figure 4, for a trace σ and an event a : $remove\_last\_event(\sigma.a)$=$\sigma$, $last\_event(\sigma)$= ϵ if σ= ϵ and $last\_event(\sigma)$= a if σ= $\sigma_1.a$. The *error* function returns the error and exits the algorithm.

```
Input : $TP$ : test purpose, $S_1, S_2$ ; 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 : $\mu_0 = \epsilon$ ; $TP_{S_i} = \epsilon$ ;
for (j=0 ;$j \leq n$ ;j++) do
   if ($\mu_j \in \Sigma^{S_i}$) then $TP_{S_i} = TP_{S_i}.\mu_j$ (* just add *)
   if ($\mu_j \in \Sigma_L^{S_k}$) then $TP_{S_i} = TP_{S_i}.\bar{\mu_j}$ (* just add the mirror *)
   if ($\mu_j \in \Sigma_U^{S_k} \cup \{\tau\}$)
      $\sigma_1 := TP_{S_i}$ ; $a_l$ =last_event($\sigma_1$)
      while $a_l \in \Sigma_U^{S_k} \cup \{\tau\}$ do $\sigma_1$=remove_last_event($\sigma_1$)
         $a_{l-1}$ =last_event($\sigma_1$) (* $a_{l-1}$ is the last event added to $TP_{S_i}$ and
      end                                  a mirror event $\bar{a}_{l-1}$ may exist in $S_k$ *)
      $M_{S_k}$ = $\{q \in Q^{S_k}$ such that q $\overset{\bar{a}_{l-1}}{\rightarrow}$ and $\sigma = \bar{a}_{l-1}.\omega.\mu_j \in Traces(q)\}$
      if ($\forall q \in M_{S_k}$, $q \overset{\sigma}{\not\longrightarrow}$) then error(TP not valid : no path to $\mu_i$)
      while (e=last_event($\omega$) $\notin \Sigma_L^{S_k} \cup \{\epsilon\}$) do $\omega$=remove_last_event($\omega$) end
      if (e $\in \Sigma_L^{S_k}$) then $TP_{S_i} = TP_{S_i}.\bar{e}$
   else error(TP not valid : $\mu_j \notin \Sigma^{S_1} \cup \Sigma^{S_2}$)
```

Figure 4. *Algorithm to derive TP_{S_i} from TP*

2- The second step of the proposed method consists in using conformance test generation tool to derive two unilateral iop test cases TC_1 and TC_2 (see figure 3(b)). The obtained test cases are modified in order to take into account the observation of both output and input events in interoperability testing. For example, $l!m$ (resp. $l?m$) will be replaced by $?(l?m)$ (resp. $?(l!m)$). 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.
Remark : the first step of the method proposed here (cf. figure 3(b)) is linear in the maximum size of specifications. Indeed, it is a simple path search algorithm. The second step is also linear in complexity, at least when using TGV [2]. Thus, it costs less than calculating $S_1 \|_{\mathcal{A}} S_2$ with the classical method based on a global iop criterion.

5.3. Applying the method to an example

Let us consider the two specifications S_1 and S_2 of figure 2 and the interoperability test purpose $TP = l1?a.U2!N$. This test purpose is interesting because it contains events on both interfaces and both IUT. Applying the algorithm of figure 4, we obtain : TP_{S_1}=$l1!a.l1?n$ and $TP_{S_2} = \bar{\mu}_1.\mu_2 = l2!a.U2!N$. The obtained test cases TC_1 and TC_2 are given in upper side of figure 5. For interoperability test case generation based on the global relation, the obtained TC (cf. third test case in figure 5) comes from the composition of $S_1\|_{\mathcal{A}}S_2$ with TP. According to the theorem 4.1, final interoperability verdicts obtained with TC_1 and TC_2 should be the same as the verdict obtained with TC. Due to page limitation, the proof cannot be given here but a look at glance to TC_1 and TC_2 shows the same paths and verdicts in TC.

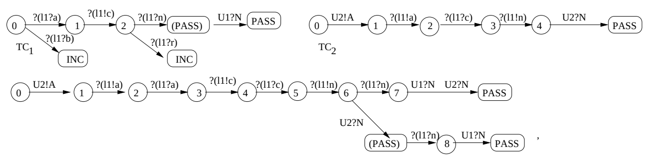

Figure 5. *Interoperability test cases obtained for $TP = l1?a.U2!N$*

6. Conclusion

In this paper, *interoperability criteria* taking quiescence into account are defined, describing the conditions under which two IUT can be considered interoperable. A theorem proving that two of them are equivalent allows a new method for interoperability test generation that avoids the classical state-explosion problem.
The obtained test cases suggest a distributed approach for interoperability testing. Our study restricted the proposed framework to the one-to-one interoperability testing context. Further studies will consider distributed interoperability testing for architecture composed of more than two implementations.

7. Bibliographie

[1] TRETMANS J., « Testing Concurrent Systems : A Formal Approach », *CONCUR'99 – 10^{th} Int. Conference on Concurrency Theory, volume 1664 of LNCS, pages 46-65*, 1999

[2] FERNANDEZ J.-C., JARD C., JÉRON T., VIHO C., « An experiment in automatic generation of test suites for protocols with verifi cation technology », *Science of Computer Programming - Special Issue on Industrial Relevant applications of Formal Analysis Techniques*, 1997

[3] RAFIQ O., CASTANET R., « From conformance testing to interoperability testing », *Protocol Test Systems, volume III, pages 371-385, North-Holland*, 1991

[4] ARAKAWA N., PHALIPPOU., RISSER N., SONEOKA T., « Combination of conformance and interoperability testing », *Formal Description Techniques FORTE'92*, september 1992

[5] CASTANET R., KONE O., « Test generation for interworking systems », *Computer Communications, volume 23, pages 642-652,Elsevier Science*, 2000

[6] SEOL S., KIM M., KANG S., RYU J., « Fully automated interoperability test suite derivation for communication protocols », *Comput. Networks, 43(6) :735-759*, 2003

[7] BARBIN S., TANGUY L., VIHO C., « Towards a formal framework for interoperability testing », *FORTE'01*, august 2001

[8] VERHAARD L., TRETMANS J., KARS P., BRINKSMA E., « On asynchronous testing », *Fifth international workshop on protocol test systems, North-Holland* 1993

[9] DESMOULIN A., VIHO C., « Quiescence management improves interoperability testing », *TestCom 2005, Lecture Notes in Computer Science volume 3502, pages 364-378*, mai 2005