Formalizing Interoperability for Test Case Generation Purpose

Alexandra Desmoulin and César Viho

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

Abstract. This study deals with interoperability formal definitions and test derivation avoiding the state-space explosion problem. First, the notion of *interoperability criteria* is introduced. An interoperability criterion formally describes the conditions that two implementations must verify in order to be considered interoperable. The second point studied in this paper is interoperability test derivation. Based on the equivalence of two interoperability criteria, we proposed a method to derive automatically interoperability test cases.

1 Introduction

Different types of tests exist to ensure that implementations will work correctly in a real operational environment. Among these tests, conformance testing is used to verify if an implementation behaves as described in its specification, generally a standard. Another type of test is the interoperability test. Goals of interoperability are multiple. First, one has to test if the considered implementations communicate correctly. Secondly, they must behave during their interaction as described in their respective specifications. Third, they must provide the expected services.

Conformance testing is precisely characterized. Testing architectures and conformance relations [5, 7] were defined leading to automatic test generation [4, 8] and execution. This is not the case for interoperability testing. However, some attempts to give definitions of interoperability or methods to derive interoperability tests exists in [1, 2, 3]. In this paper, we give formal definitions of interoperability with *interoperability criteria* (*iop criteria* for short in the following) that give conditions to be verified by implementations to be considered interoperable. These iop criteria manage quiescence. Indeed, implementations are allowed to be quiescent if it is foreseen in their specification. Based on these criteria, we describe a method to generate automatically interoperability test cases which avoids the well-known state-space explosion problem.

This paper is structured as follows. First, Section 2 describes possible interoperability testing architectures. Section 3 presents the formal definitions used

in this paper. The interoperability 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 an example in Section 6. Conclusion and future work are in Section 7.

2 Testing architecture

This study considers a one-to-one interoperability context. The interoperability system under test (SUT) is composed of two implementations (see figure 1). These two IUT (Implementation Under Test) are supposed to behave as described in their respective specification. They communicate with each other while providing the expected service.

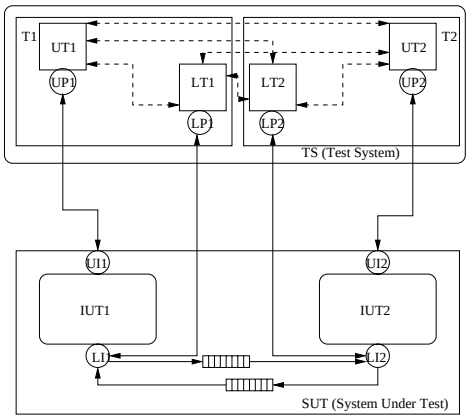

Fig. 1. Test architecture for an asynchronous interaction

In this context, two kind of interfaces can be differentiated. First, the interfaces LI_i (lower interfaces) are used for the interaction between the two IUT (see figure 1). These interfaces are only observable but not controllable. Indeed, a test system connected to such interfaces can only observe the events, but it cannot send a stimulus to these interfaces. The lower tester LT_i is in charge of the observation of LI_i via the *lower PO* (Point of Observation) LP_i.
The other interfaces are the interfaces UI_i (upper interfaces). These interfaces are not used for the interaction of the IUT but they are the interfaces through which the IUT communicate with its environment. These interfaces are observable and also controllable. The upper tester UT_i is in charge of the control and observation of UI_i via the *upper PCO* (Point of Control and Observation) UP_i. Thus, the tester T_i, composed by UT_i and LT_i, is the part of the Test System (TS) in charge of the control and observation of IUT_i.

Depending on the access of the different interfaces, different interoperability testing architectures can be distinguished as described in [2, 10]. The architecture is called **lower** (resp. **upper**) if only the lower interfaces (resp. the upper interfaces) are accessible, and **total** if both kind of interfaces are accessible. The interoperability testing architecture is called **unilateral** if only the interfaces of one of the two IUT are accessible. It is called **bilateral** if the interfaces of the two IUT are accessible but separately. The **global** architecture corresponds to the more usually considered case where all the interfaces of the two IUT are accessible with a global view.

Synchronous or asynchronous communication The interaction between the two IUT is asynchronous (cf. section 3.3). Notice also that the interaction between UP_i and the IUT can be either synchronous or asynchronous. It depends on the testing environment. We will consider that this latter is synchronous.

3 Formal background

In this study, we will use the well-known IOLTS (Input-Output Labeled Transition System) to model specifications. As usual in the black-box testing context, we also need to model implementations, even though their behaviours are normally unknown. They will also be represented by an IOLTS.

3.1 IOLTS model

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 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 interaction points (with the environment) of the system.* $\Sigma^M \subseteq P^M \times \{?, !\} \times A^M$ *where* P^M *is the finite set of interaction points (ports) through which the system communicates with lower or upper layer, or other systems, "?" and "!" respectively denote an input and an output of message,* A^M *is the alphabet of input-output messages exchanged by the system through its ports.*
- $\Delta^M \subseteq Q^M \times (\Sigma^M \cup \{\tau\}) \times Q^M$ *is the transition relation, where* $\tau \notin A^M$ *denotes an internal event. We note* $q \xrightarrow{\alpha}_M q'$ *for* $(q, \alpha, q') \in \Delta^M$ *and* $q \overset{\alpha}{\not\longrightarrow}$ *if there is no state* q' *such that* $(q, \alpha, q') \in \Delta^M$.

Σ^{M} can be decomposed as follow: $\Sigma^{\mathrm{M}} = \Sigma_U^{\mathrm{M}} \cup \Sigma_L^{\mathrm{M}}$ (with $\Sigma_U^{\mathrm{M}} \cap \Sigma_L^{\mathrm{M}} = \emptyset$), 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 in order to distinguish input from output messages. $\Sigma^{\mathrm{M}} = \Sigma_I^{\mathrm{M}} \cup \Sigma_O^{\mathrm{M}}$, where Σ_I^{M} (resp. Σ_O^{M}) is the finite set of input (resp. output) messages.

In the following, $\mathcal{IOLTS}$ will denote the set of IOLTS. Let us consider $M \in \mathcal{IOLTS}$, and let $\alpha \in \Sigma^{\mathrm{M}}$ with $\alpha = p.\{?, !\}.m$, $\mu_i \in \Sigma^{\mathrm{M}} \cup \tau$, $\sigma \in (\Sigma^{\mathrm{M}})^*$, $q, q', q_i \in Q^{\mathrm{M}}$, we have[1]:

- $q \xrightarrow{\mu_1 \ldots \mu_n}_{\mathrm{M}} q' =_\Delta \exists\, q_0 = q, q_1 ..., q_n = q', \forall i \in [1, n], q_{i-1} \xrightarrow{\mu_i}_{\mathrm{M}} q_i$.

[1] $=_\Delta$ stands for "by definition"

- $q \overset{\epsilon}{\Rightarrow}_{\mathrm{M}} q' =_{\Delta} q = q'$ or $q \overset{\tau \ldots \tau}{\longrightarrow}_{\mathrm{M}} q'$.
- $q \overset{\alpha}{\Rightarrow}_{\mathrm{M}} q' =_{\Delta} \exists\, q_1, q_2, q \overset{\epsilon}{\Rightarrow}_{\mathrm{M}} q_1 \overset{\alpha}{\rightarrow}_{\mathrm{M}} q_2 \overset{\epsilon}{\Rightarrow}_{\mathrm{M}} q'$.
- $q \overset{\sigma}{\Rightarrow}_{\mathrm{M}} q' =_{\Delta} q \overset{\mu_1 \cdots \mu_n}{\Longrightarrow}_{\mathrm{M}} q' =_{\Delta} \exists\, q_0 = q, q_1 \ldots, q_n = q', \forall i \in [1, n], q_{i-1} \overset{\mu_i}{\Rightarrow}_{\mathrm{M}} q_i, \sigma = \mu_1 \cdots \mu_n$.
- $\Gamma(q) =_{\Delta} \{\alpha \in \Sigma^{\mathrm{M}} \mid \exists\, q' \text{ and } q \overset{\alpha}{\rightarrow}_{\mathrm{M}} q'\}$, and $out(q) =_{\Delta} \{\alpha \in \Sigma^{\mathrm{M}}_{\mathrm{O}} \mid \exists\, q' \text{ and } q \overset{\alpha}{\rightarrow}_{\mathrm{M}} q'\}$ is the set of outputs from q.
- q *after* $\sigma =_{\Delta} \{q' \in Q^{\mathrm{M}} \mid q \overset{\sigma}{\Rightarrow}_{\mathrm{M}} q'\}$ 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) and will be noted by (M *after* σ). In the same manner, $Out(M, \sigma) =_{\Delta} out(M \text{ after } \sigma)$.
- $Traces(q) =_{\Delta} \{\sigma \in (\Sigma^{\mathrm{M}})^* \mid q \text{ after } \sigma \neq \emptyset\}$ is the set of possible observable traces from q. And, $Traces(M) =_{\Delta} Traces(q_0^{\mathrm{M}})$.
- $\bar{\mu}$= p!a if μ = p?a and $\bar{\mu}$ = p?a if μ = p!a. For internal events, $\bar{\tau} = \tau$.

3.2 Quiescence, suspensive IOLTS and conformance relation ioco

Three main situations lead to quiescence of a system : *deadlocks*, *outputlocks* and *livelocks*. A *deadlock* corresponds to a state after which no event is possible : $q \in deadlock(M) =_{\Delta} \Gamma(q) = \emptyset$. An *outputlock* corresponds to a state after which only transitions labeled with input exist and none of these inputs is observed : $q \in outputlock(M) =_{\Delta} \Gamma(q) \subseteq \Sigma^{\mathrm{M}}_{I}$. A *livelock* corresponds to a loop of internal events : $q \in livelock(M) =_{\Delta} \exists \tau_1, \cdots, \tau_n, q \overset{\tau_1, \cdots, \tau_n}{\rightarrow} q$. Thus, $q \in quiescent(M) =_{\Delta} q \in deadlock(M) \vee q \in outputlock(M) \vee q \in livelock(M)$. A quiescence state $q \in quiescent(M)$ is modeled by $q \overset{\delta}{\rightarrow}_{\mathrm{M}} q$ where δ is treated as an observable output event. The obtained IOLTS is called suspensive IOLTS [7], is noted $\Delta(M)$, and we have $STraces(S) = Traces(\Delta(S))$. Figure 2 gives an example of two specifications using the IOLTS model. Quiescence is modeled in the states 0 and 2 of S_1, and in the state 0 of S_2.

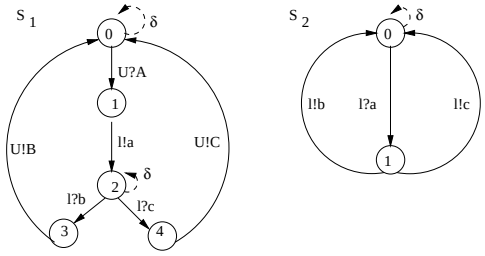

Fig. 2. Specifications S_1 and S_2

Interoperability criteria defined in Section 4.2 are based on the **ioco** conformance relation [7]. This relation says that an implementation I is **ioco**-conformant with respect to its specification S if I can never produce an output which could not be produced by S after the same suspension trace. Moreover, I

may be quiescent only if S can do so. Formally :
I **ioco** $S =_{\Delta} \forall \sigma \in STraces(S)$, $Out(\Delta(I), \sigma) \subseteq Out(\Delta(S), \sigma)$.

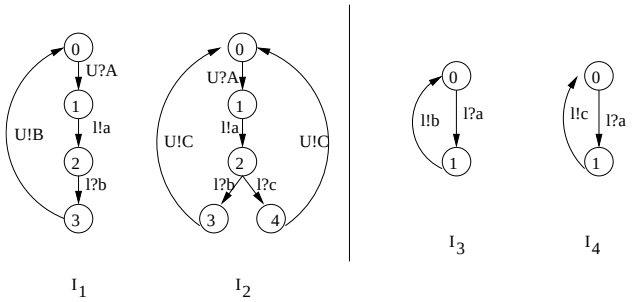

Fig. 3. Implementation I_1 and I_2 of S_1, and I_3 and I_4 of S_2

Let us consider the implementation I_1 and I_2 of S_1 of figure 3 : $\neg I_2$ **ioco** S_1 (because of the output $U!B$ after the reception of $l?b$) and $\neg I_1$ **ioco** S_1 (because if the tester sends the message c on the lower interface of I_1, the implementation remains quiet, but no quiescence is foreseen in the state 4 of S_1). For the implementations I_3 and I_4 of S_2 of figure 3, we have I_3 **ioco** S_2 and I_4 **ioco** S_2.

3.3 Interaction

Interoperability testing generally deals with interactions of two or more implementations. To provide a formal definition of interoperability in a one-to-one context, we need to model the interaction of two IOLTS.

Definition 2 (Synchronous interaction $\|$). *The synchronous interaction of two IOLTS M_1 and M_2 is noted $M_1 \| M_2 = (Q^{M_1\|M_2}, \Sigma^{M_1\|M_2}, \Delta^{M_1\|M_2}, (q_0^{M_1}, q_0^{M_2}))$ with $Q^{M_1\|M_2} \subseteq Q^{M_1} \times Q^{M_2}$, $\Sigma^{M_1\|M_2} \subseteq \Sigma^{M_1} \cup \Sigma^{M_2}$. The transition relation $\Delta^{M_1\|M_2}$ is obtained as follows. $\forall (q_1, q_2) \in Q^{M_1} \times Q^{M_2}$,*

$$\frac{(q_1, a, q_1') \in \Delta^{M_1}, a \in \Sigma_U^{M_1} \cup \{\tau\}}{((q_1, q_2), a, (q_1', q_2)) \in \Delta^{M_1\|M_2}}, \frac{(q_2, a, q_2') \in \Delta^{M_2}, a \in \Sigma_U^{M_2} \cup \{\tau\}}{((q_1, q_2), a, (q_1, q_2')) \in \Delta^{M_1\|M_2}} \quad (1)$$

$$\frac{(q_1, a, q_1') \in \Delta^{M_1}, (q_2, \bar{a}, q_2') \in \Delta^{M_2}, a \in \Sigma_L^{M_1}, \bar{a} \in \Sigma_L^{M_2}}{((q_1, q_2), a, (q_1', q_2')) \in \Delta^{M_1\|M_2}} \quad (2)$$

There are different ways to obtain the model of the interaction of two IOLTS with quiescence management. The method chosen here is calculating first the suspensive IOLTS $\Delta(M_1)$ and $\Delta(M_2)$, as explained in section 3.2. This step is then followed by constructing the interaction of $\Delta(M_1)$ and $\Delta(M_2)$, using rules (1) and (2) of the definition 2. The main difficulty here is to preserve information that indicates the IUT in which quiescence is observed and to make appearing new quiescence introduced by the interaction. A quiescent state is noted : $(q_1, q_2) \overset{\delta(1)}{\rightarrow}_{\mathrm{M}} (q_1', q_2')$ if $(q_1 \overset{\delta}{\rightarrow}_{\mathrm{M}} q_1') \in \Delta(M_1)$, $(q_1, q_2) \overset{\delta(2)}{\rightarrow}_{\mathrm{M}} (q_1', q_2')$ if $(q_2 \overset{\delta}{\rightarrow}_{\mathrm{M}} q_2') \in$

$\Delta(M_2)$, and $(q_1, q_2) \xrightarrow{\delta}_{\mathrm{M}} (q'_1, q'_2)$ if $((q_1, q_2) \xrightarrow{\delta(1)}_{\mathrm{M}} (q'_1, q'_2)) \wedge ((q_1, q_2) \xrightarrow{\delta(2)}_{\mathrm{M}} (q'_1, q'_2))$. It is obtained by propagating δ of $\Delta(M_1)$ and $\Delta(M_2)$.

As, in the considered interoperability testing architecture, the interaction between the two implementations is asynchronous, we also need to model this asynchronous interaction. As in [9], we can model the asynchronous environment with FIFO queues. In [6], the asynchronous transformation $\mathcal{A}$ is defined. This transformation applied to a specification S gives as result the IOLTS $\mathcal{A}(S)$ representing the behaviour of S in an asynchronous environment. As consequence, the asynchronous interaction of M_1 and M_2 corresponds to the synchronous interaction of $\mathcal{A}(M_1)$ and $\mathcal{A}(M_2)$, noted $M_1 \| _{\mathcal{A}} M_2$.

3.4 Projection

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 behaviour of the implementation on a set (called X in the following). This latter is used to select the expected events. Quiescence δ has to be seen in the projection as an observable event. For an IOLTS built from an interaction $M_1 \| _{\mathcal{A}} M_2$, quiescence $\delta(1)$ is an observable event of M_1 and $\delta(2)$ of M_2. The projection of an IOLTS M on the set of events X is noted by M/X and is obtained by hiding events (replacing by internal events) that do not belong to X, followed by determinization.

3.5 Modeling an implementation for interoperability testing : the iop-input completion

As described in figure 1, the two IUT interact asynchronously and testers are connected to their interfaces. When an IUT sends a message m that cannot be treated by the other IUT, the problem is how to consider this message in the point of view of the receiver. Indeed, this message m is put in the input FIFO queue of the receiver that cannot effectively treat it. Thus, this receiving implementation may be quiescent. It can neither treat the message m in its input FIFO queue ($l?m$), nor it can do any other action because its input FIFO queue is not empty and no output is possible. To model this behaviour, we choose to complete any implementation with inputs corresponding to the output alphabet of the other IUT specification. These new 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 (see [6]).

Definition 3 (The iop-input completion).
Let us consider an IUT $I_1 = (Q, \Sigma, \Delta, q_0)$ *based on the specification* $S_1 = (Q_{S_1}, \Sigma_{S_1}, \Delta_{S_1}, q_0^{S_1})$ *interacting with an IUT based on the specification* $S_2 = (Q_{S_2}, \Sigma_{S_2}, \Delta_{S_2}, q_0^{S_2})$. *The iop-input completion of* I_1 *is* $\mathcal{C}(I_1) = (Q_{\mathcal{C}}, \Sigma_{\mathcal{C}}, \Delta_{\mathcal{C}}, q_0)$.

$Q_{\mathcal{C}} = Q \cup \{q_E, q_{\mathcal{C}}\}$, *$q_E$ represents the error trap state and $q_{\mathcal{C}}$ is the other input-completion state.* $\Sigma_{\mathcal{C}} = \Sigma \cup \{\bar{a} | a \in \Sigma_O^{S_2} \cap \Sigma_L^{S_2}\}$. $\Delta_{\mathcal{C}} = \Delta \cup \{(q, a, q_E) | q \in Q, \bar{a} \in \Sigma_O^{S_2} \cap \Sigma_L^{S_2}, q \overset{a}{\not\longrightarrow}\} \cup \{(q, a, q_{\mathcal{C}}) | q \in Q, a \in \Sigma_I^{S_1} \cap \Sigma_U^{S_1}, q \overset{a}{\not\longrightarrow}\} \cup \{(q_{\mathcal{C}}, x, q_{\mathcal{C}}) | x \in \Sigma\}$.

Remark : The iop-input completion adds only transitions labeled with inputs to the original IOLTS representing the implementation. Thus, quiescence modeled in $\mathcal{C}(I_1)$ or in I_1 is the same. To model the deadlock in the error state q_E, quiescence must be modeled in the iop-input completed implementation $\mathcal{C}(I_1)$. Thus, $\Delta(\mathcal{C}(I_1))$ is the model of the behaviour of I_1 in an asynchronous environment. In the following, the implementations are considered iop-input completed. Quiescence is also modeled on the considered implementations.

4 Interoperability (iop) criteria

In this section, we define two iop criteria. These criteria formally describe conditions that have to be verified by two implementations to be considered interoperable. We prove their equivalence in terms of non-interoperability detection. These definitions only apply for compatible specifications. Indeed, two implementations cannot be interoperable if their specifications are not compatible.

4.1 Compatibility of the considered specifications

Two specifications are compatible if after any trace of the interaction, for each possible output on the interfaces used for the interaction, the corresponding input is foreseen in the other specification. In a formal way : $\forall \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $\sigma / \Sigma^{S_1} = \sigma_1$, $\sigma / \Sigma^{S_2} = \sigma_2 \Rightarrow \{Out_{\Sigma_L}(S_1, \sigma_1) \subseteq In_{\Sigma_L}(S_2, \sigma_2)$ and $Out_{\Sigma_L}(S_2, \sigma_2) \subseteq In_{\Sigma_L}(S_1, \sigma_1)\}$. In the following, specifications are supposed to be compatible.

4.2 Definition of the iop criteria

In this section, we consider the global interoperability testing architecture (see Section 2). It is the most commonly used in practice for one-to-one interoperability testing. We define two iop criteria considering the events executed on the different interfaces of the implementations in two different ways.
The first iop criterion is the global iop criterion iop_G. It 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 (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* $I_2 =_{\Delta} \forall \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $Out(I_1 \|_{\mathcal{A}} I_2, \sigma) \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$

The other iop criterion defined in this section is the bilateral iop criterion iop_B. It 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, and the same in the point of view of I_2 implementing the specification S_2.

Definition 5 (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 $\forall \sigma_2 \in Traces(\Delta(S_2))$, $\forall \sigma' \in Traces(S_2 \|_{\mathcal{A}} S_1)$, $\sigma / \Sigma^{S_2} = \sigma_2 \Rightarrow$
$Out((I_2 \|_{\mathcal{A}} I_1)/\Sigma^{S_2}, \sigma_2) \subseteq Out(\Delta(S_2), \sigma_2)$.

As an example, with the implementations I_1, I_2, I_3 and I_4 of figure 3 (implementing S_1 and S_2 of figure 2), we have the results :
• I_1 iop_B I_3 and I_1 iop_G I_3 although $\neg(I_1$ **ioco** $S_1)$.
• $\neg$ (I_2 iop_B I_3) and $\neg$ (I_2 iop_G I_3), but I_2 iop_B I_4 and I_2 iop_G I_4. Indeed, the output $U!C$ is not allowed in S_1 after $l?b$, but only I_3 can send b, not I_4.
• $\neg$ (I_1 iop_B I_4) and $\neg$ (I_1 iop_G I_4). *Such a non-interoperability case would not have been detected without quiescence management.* Indeed, the non-interoperability is due to the sending of message $l2!c$ by I_4 which is not expected by I_1. Thus, this message is put in the input queue of I_1 but not treated. The whole SUT is in a deadlock situation. This deadlock is not foreseen in the specification interaction. Thus the iop criteria are not verified due to non allowed quiescence.

4.3 Equivalence of the two interoperability criteria

The most important result here is the following theorem 1. It says that the *global iop criterion* iop_G is equivalent to the the so-called *bilateral iop criterion* iop_B, in terms of non-interoperability detection. Its proof needs the lemmas defined in the following.

Theorem 1. $I_1\, iop_G\, I_2 \Leftrightarrow I_1\, iop_B\, I_2$

Lemma 1. *Let us consider $M_1, M_2 \in \mathcal{IOLTS}$, and let $\sigma \in Traces(M_1 \|_{\mathcal{A}} M_2)$,*
$Out(M_1 \|_{\mathcal{A}} M_2, \sigma) = Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$.

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 $a \in \Sigma^{M_1} \cup \{\delta, \delta(1)\}$, or $a \in \Sigma^{M_2} \cup \{\delta, \delta(2)\}$ ie. either $a \in Out(\Delta(M_1)$, $\sigma/\Sigma^{M_1})$, or $a \in Out(\Delta(M_2)$, $\sigma/\Sigma^{M_2})$.
$\Rightarrow Out(M_1 \|_{\mathcal{A}} M_2, \sigma) \subseteq Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$.
2. In the other sense, it is easy to see that : $Out(M_1 \|_{\mathcal{A}} M_2, \sigma) \subseteq Out(\Delta(M_1)$, $\sigma/\Sigma^{M_1})$ $\cup\, Out(\Delta(M_2), \sigma/\Sigma^{M_2})$.

Lemma 2. *Let $M_1, M_2 \in \mathcal{IOLTS}$.*
$((M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1}) \|_{\mathcal{A}} ((M_2 \|_{\mathcal{A}} M_1)/\Sigma^{M_2}) = M_1 \|_{\mathcal{A}} M_2$

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 $\sigma = \sigma_1 \|_{\mathcal{A}} \sigma_2 \in Traces(((M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1}) \|_{\mathcal{A}} ((M_2 \|_{\mathcal{A}} M_1)/\Sigma^{M_2}))$.

Lemma 3. *Let $M_1, M_2 \in \mathcal{IOLTS}$, $\sigma_1 \in Traces(\Delta(M_1))$, $\sigma \in Traces(M_1 \|_{\mathcal{A}} M_2)$ 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 $\Sigma^{(M_1 \|_{\mathcal{A}} M_2)/\Sigma^{M_1}} \cup \{\delta\} \subseteq \Sigma^{M_1} \cup \{\delta\}$

Proof. of theorem 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 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 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, $S_2 \in \mathcal{IOLTS}$ 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 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 $I_1 \, iop_G \, I_2 \Rightarrow I_1 \, iop_B \, I_2$.

Based on the theorem 1, one may wonder how it can help interoperability test generation. This is the purpose of the study developed in the next section.

5 Interoperability test generation

In this section, we investigate the way to generate interoperability test using the equivalence between the bilateral and global criteria (cf. theorem 1).

5.1 General principles for interoperability test generation

The goal is to generate interoperability test cases (TC) that can be executable on the SUT. We consider a System Under Test (SUT) composed of two IUT interacting asynchronously (cf. figure 1 in Section2). These IUT are represented by a suspensive iop-input completed IOLTS. In practice, the inputs of a general interoperability test generation algorithm are the two specifications on which the implementations are based, and a test purpose (TP). A TP is a particular property (or a behaviour in the interaction between the implementations) to be tested. In general, test purposes are incomplete sequences of actions. Let $\mathcal{S}$ be the set of specifications, $\mathcal{P}$ the set of test purposes and $\mathcal{TC}$ the set of interoperability test cases. The goal of a one-to-one interoperability test generation algorithm g is : $\mathcal{S} \times \mathcal{S} \times \mathcal{P} \rightarrow \mathcal{TC}$. Figure 4 (a) shows an example.

During conformance tests, a tester can send a stimulus to the implementation or receive an input. In the interoperability testing case, three kind of events are possible : sending of stimuli to the upper interfaces of the implementations, reception of inputs from these interfaces, but also observation of events (input and output) on the lower interfaces.

Interoperability test cases modeling A test case TC is represented by an extended version of IOLTS called T-IOLTS for Testing IOLTS. A T-IOLTS TC can be defined by $TC = (Q^{TC}, \Sigma^{TC}, \Delta^{TC}, q_0^{TC})$. $\{PASS, FAIL, INC\} \subseteq Q^{TC}$ are trap states representing interoperability verdicts. q_0^{TC} is the initial state. $\Sigma^{TC} \subseteq \{\mu|\bar{\mu} \in \Sigma_U^{S_1} \cup \Sigma_U^{S_2}\} \cup \{?(\mu)|\mu \in \Sigma_L^{S_1} \cup \Sigma_L^{S_2}\}$. $?(\mu)$ denotes the observation of the message μ on a lower interface. Δ^{TC} is the transition function.
In the following, any TC is supposed to be deterministic, and controllable (if a tester can do an output in a state, no other action is possible for the test case in this state). A TC must also be input and observation complete in the input and observation states : if an input or an observation is possible in a state, all other inputs and observations are possible in this state (generally denoted in test cases with ?*otherwise* label leading to $FAIL$). Moreover at least one of the verdict states ($PASS$, $FAIL$, or INC) is accessible from every state.

The execution of the test case TC of the SUT (composed of the two considered IUT) gives a verdict : $verdict(TC, SUT) \in \{PASS, FAIL, INC\}$. The meanings of the possible interoperability verdicts are PASS : no error was detected during the tests, FAIL : the interoperability criterion is not verified and INC (for Inconclusive) : the behaviour of the SUT seems valid but it is not the purpose of the test case.

Test generation based on the global iop criteria The construction of Test Cases based on the Global iop Criteria iop_G begins with the construction of the asynchronous interaction $S_1 \parallel_{\mathcal{A}} S_2$. Then $S_1 \parallel_{\mathcal{A}} S_2$ is composed with the test purpose TP. During this operation, two main results are calculated. First TP is validated. If the events composing TP are not found in the specifications (or not in the order described in TP), TP is not a valid Test Purpose. The composition is also used to keep (in the interaction of the two specifications) only the events concerned by the Test Purpose. It calculates the different ways to observe/execute TP on the SUT.
Problem : the construction of $S_1 \| _{\mathcal{A}} S_2$ can cause state-space explosion. 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. Interoperability test derivation with this method may be impossible even for small specifications combined with "on-the-fly" techniques [4].

5.2 Using the equivalence between bilateral and global criteria

The theorem 1 of Section 4.3 proves that global and bilateral iop criteria are equivalent. We propose here a method to generate interoperability test cases that takes benefit from this result. This method uses conformance test tools.
Based on the bilateral iop criterion, the idea is to use a conformance test tool $\mathcal{F}$ such that $\mathcal{F} : (S_1, TP_{S_1}) \rightarrow TC_1$ and $\mathcal{F} : (S_2, TP_{S_2}) \rightarrow TC_2$. TP_{S_1} and TP_{S_2}

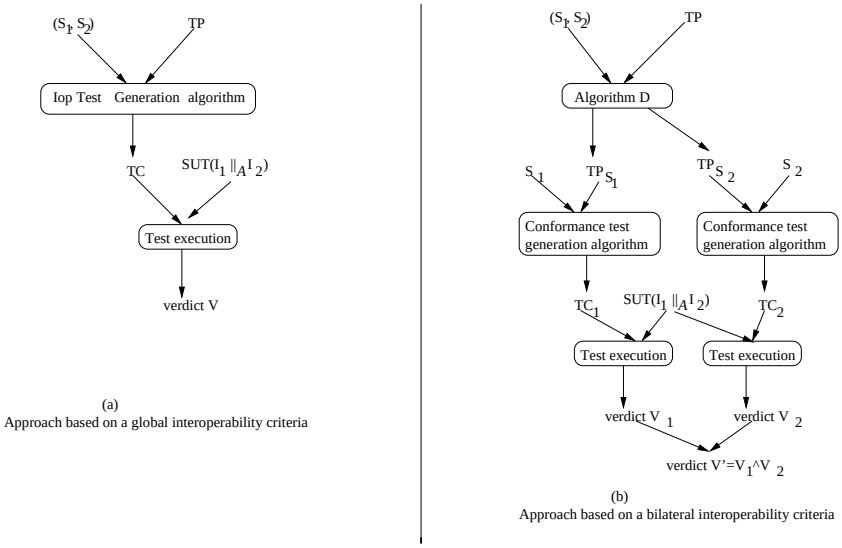

Fig. 4. Interoperability Test Cases Generation

are kind of "unilateral" test purposes derived from the test purpose TP. TP_{S_i} is obtained from TP and contains only events of S_i.
In this context, the meaning of the theorem 1 is : $verdict(TC,\ I_1\|_{\mathcal{A}}I_2) = verdict(TC_1,\ I_1\|_{\mathcal{A}}I_2) \wedge verdict(TC_2,\ I_1\|_{\mathcal{A}}I_2)$. The iop_G verdict $verdict(TC,\ I_1\|_{\mathcal{A}}I_2)$ is an interoperability verdict with a global architecture. The two other verdicts are kinds of conformance verdicts. $verdict(TC_1, I_1\|_{\mathcal{A}}I_2)$ (resp. $verdict(TC_2,\ I_1\|_{\mathcal{A}}I_2)$) is the verdict obtained by executing TC_1 (resp. TC_2) unilaterally on interfaces of I_1 (resp. I_2) during its interaction with I_2 (resp. I_1). The rules for the combination of these two verdicts to obtain the final iop_B verdict are given by : $PASS \wedge PASS = PASS$, $PASS \wedge INC = INC$, $PASS \wedge FAIL = FAIL$, $INC \wedge FAIL = FAIL$, $INC \wedge INC = INC$ and $FAIL \wedge FAIL = FAIL$.

Test generation based on the bilateral criterion iop_B As described in figure 4 (b), the generation of TC_1 and TC_2 based on the bilateral criterion can be decomposed in two principal steps. First, step 1 (algorithm $\mathcal{D}$) correspond to the derivation of TP_{S_1} and TP_{S_2} from TP. Then, step 2 is the calculation of TC_1 and TC_2. This step corresponds to the function $\mathcal{F}$ applied on (TP_{S_1}, S_1) and (TP_{S_2}, S_2) and uses a conformance test tool.
For the execution of the test cases, TC_1 is applied on I_1 (during its interaction with I_2), and TC_2 on I_2 (during its interaction with I_1) leading to two verdict $V1$ and $V2$. The final interoperability verdict $V' = V1 \wedge V2$ is obtained with the rules given above.

Step 1 : We will explain here how to obtain TP_{S_1} from TP.
TP says that after the execution of some events $\mu_1 ... \mu_{n-1}$, the tester must observe another event μ_n, but does not explicit necessarily what may happen be-

tween μ_i and μ_{i+1}. In a formal context, TP is represented by an extended IOLTS. The most difficult problem to obtain TP_{S_1} from TP is that $\mu_1...\mu_n$ may contain any events of both Σ^{S_1} and Σ^{S_2}. Thus, the algorithm to derive TP_{S_1} from TP, S_1 and S_2 consists in separating events from S_1 (in TP_{S_1}) and S_2 (in TP_{S_2}) while keeping all information needed for the test generation.

If all the events described in TP are events on the lower interfaces, the algorithm to obtain TP_{S_1} and TP_{S_2} represented figure 5 is very simple. But if TP contains events on the upper interfaces, this algorithm needs to go through the IOLTS representing the specification S_2. It finds a path between μ_{i-1} (or $\bar{\mu}_{i-1}$) and μ_i. This operation is however simple and it costs less than calculating $S_1 \|_{\mathcal{A}} S_2$ with the method based on a global iop criterion (cf. figure 4 (a)).
This algorithm only verify the existence of μ_i in the alphabet of the specifications. The verification of TP (thus the verification of TP_{S_1} and TP_{S_2} derived from TP) is done in step 2.
The algorithm of figure 5 uses some functions described hereafter. Let us consider a trace σ and an event a. The function *remove_last_event* is defined by : remove_last_event($\sigma.a$)=σ. And the function *last_event* 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.

Input: TP: test purpose; **Output:** $\{TP_{S_l}\}_{l=1,2}$;
Invariant: $S_k = S_{3-l}$ (* S_k is the other specification *); $TP = \mu_1...\mu_n$
Initialization: $\mu_0 = \epsilon$; $TP_{S_l} = \epsilon$;
for ($i = 0$; $i \leq n$; i++) **do**
  if ($\mu_i \in \Sigma^{S_l}$) **then** $TP_{S_l} = TP_{S_l}.\mu_i$ (* just add if it is an event of S_l *)
  if ($\mu_i \in \Sigma_L^{S_k}$) **then** $TP_{S_l} = TP_{S_l}.\bar{\mu}_i$ (* just add the mirror if μ_i is on the lower interface of S_k *)
  if ($\mu_i \in \Sigma_U^{S_k} \cup \{\tau\}$)
    $\sigma_1 := TP_{S_l}$; a_j =last_event(σ_1)
    while $a_j \in \Sigma_U^{S_k} \cup \{\tau\}$ **do**
      σ_1=remove_last_event(σ_1)
      a_{j-1} =last_event(σ_1) (* a_{j-1} is the last event added to TP_{S_l} and a mirror event $\bar{a}_{j-1}$ may exist in S_k *)
    end
    $M_{S_k} = \{q \in Q^{S_k}$ such that q $\stackrel{\bar{a}_{j-1}}{\rightarrow}$ and $\sigma = \bar{a}_{j-1}.\omega.\mu_i \in Traces(q)\}$
    if ($\forall q \in M_{S_k}$, $q \stackrel{\sigma}{\not\longrightarrow}$) **then** error(TP not valid : no path in S_k between the mirror event of last_event(TP_{S_l}) and μ_i)
    while (e=last_event(ω) $\notin \Sigma_L^{S_k} \cup \{\epsilon\}$) **do** ω=remove_last_event(ω) **end**
    if (e $\in \Sigma_L^{S_k}$) **then** $TP_{S_l} = TP_{S_l}.\bar{e}$
  else error(TP not valid : $\mu_i \notin \Sigma^{S_1} \cup \Sigma^{S_2}$)
end

Fig. 5. Algorithm to derive TP_{S_l} from TP

Step 2 : This step corresponds to the function $\mathcal{F}$ applied on (TP_{S_1}, S_1) and (TP_{S_2}, S_2) : (S_1, TP_{S_1}) $\rightarrow$ TC_1 and (S_2, TP_{S_2}) $\rightarrow$ TC_2. For the calculation of each test case (TC_1 and TC_2), the inputs are a specification (S_1) and a

test purpose (TP_{S_1}) based on this specification. We can use tools developed for conformance test generation like TGV [4] or TorX [8] for this step.
The most important difference consists in the access on the lower interfaces : these interfaces are observable and controllable in conformance testing but only observable in interoperability testing. Thus, the events on the lower interfaces described on the interoperability test cases obtained by $\mathcal{F}$ are only observed. The testers do not apply input on the lower interfaces. These inputs must come from the other implementation in interaction with the considered IUT. For example, if an event $l!m$ exists in the test case obtained from conformance test generation tools (which means that the tester must send the message m to the lower interface of the IUT), this will correspond to $?(l?m)$ in the interoperability test case. This means that the interoperability tester observes that a message m is received on the lower interface l. The events on the upper interfaces are controllable in both conformance and interoperability testing. Thus, no changes are made on the test cases for such events.

6 Applying the test generation algorithm to an example

Let us consider the two specifications S_1 and S_2 of figure 2 in Section 3.2. In the following, we show how the proposed algorithm can be used to derive interoperability tests. Two test purposes allow to consider two significant situations that one may deal with.
First example : let us choose a test purpose $TP = l1!a.l2!b$
This example corresponds to the most simple case where all the events described in TP are events executed on the lower interfaces. When deriving TP_{S_1} and TP_{S_2} from TP ($\mu_1.\mu_2 = l1!a.l2!b$ in the algorithm), we obtain $TP_{S_1} = \mu_1.\bar{\mu}_2 = l1!a.l1?b$ and $TP_{S_2} = \bar{\mu}_1.\mu_2 = l2?a.l2!b$. The obtained test cases TC_1 and TC_2 using TGV [4] are given in upper side of figure 6. ($PASS$) is a temporary verdict and $PASS$ is the definitive verdict obtained after a postamble which returns to initial state, and the transitions labeled with $?otherwise$ are not represented.

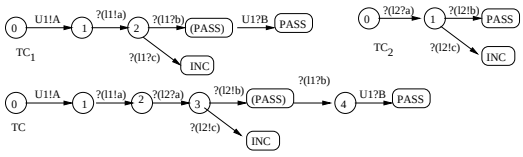

Fig. 6. The obtained test cases from $TP = l1!a.l2!b$

For interoperability test case generation based on the global relation, the obtained TC (cf. figure 6) comes from the composition of $S_1 \|_{\mathcal{A}} S_2$ with TP. Thus, final interoperability verdicts obtained with TC_1 and TC_2, executed simultaneously or not on the SUT, must be the same as the verdict obtained with TC. The proof is not given here but a look at TC_1 and TC_2 shows that there are the

same paths leading to the same verdicts as in TC.

Second example : let us now consider $TP = U1?A.U1!B$
This example is more complex than the previous one because TP contains only events on the upper interfaces of S_1. TP_{S_1} is easy to derive from TP and $TP_{S_1} = TP$. Deriving TP_{S_2} from TP is more complex. Following the algorithm :
• $\mu_1 = U1?A$. Two possibilities, either $\omega = U1?A.l1!a.l1?b.U1!B$ or $\omega = U1?A.l1!a.l1?c.U1!C$. Let us choose $\omega = U1?A.l1!a.l1?b.U1!B$. So, last_event($\omega$)=$U1!B \notin \Sigma_L^{S_1}$ and ω = remove_last_event(ω)=$U1?A.l1!a.l1?b$. Next step, last_event(ω)=$l1?b \in \Sigma_L^{S_1}$, $TP_{S_2} = l2!b$ (if $\omega = U1?A.l1!a.l1?c.U1!C$, $TP_{S_2} = l2!c$).
• $\mu_2 = U1!B : \omega = l1!a.l1?b$. Thus, last_event($\omega$)=$l1?b \in \Sigma_L^{S_1} \Rightarrow TP_{S_2}$=$l2!b.l2!b$.
Thus, we obtain $TP_{S_1} = \mu_1.\mu_2 = U1?A.U1!B$ and $TP_{S_2} = l2!b.l2!b$.

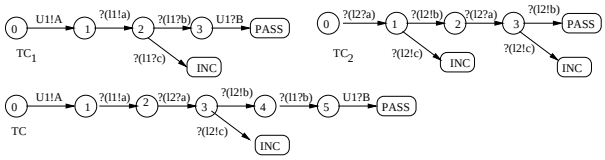

Fig. 7. Test cases obtained for $TP = U1?A.U1!B$

The obtained test cases TC_1 and TC_2 are given in upper side of figure 7. The execution of TC_1 with TC_2 until state 2 of TC_2 corresponds to the same events as the execution of TC. The most difference is that TC_2 contains supplementary events to be executed : there is a loop that returns to initial state that comes from the search of the previous event of $U1?A$ made to obtain TP_{S_2}. Thus, verdicts obtained with TC_1 and TC_2 will be the same as the verdict that would be obtained with TC. But the calculation of TC needs the interaction of S_1 and S_2 whereas TC_1 and TC_2 are obtained using existing conformance test generation tools.

Some words on parallel test case execution In the first example, TC_1 and TC_2 can be executed simultaneously because the derivation of TP_{S_1} and TP_{S_2} was simple. Indeed, the obtained test purposes contain only observations (no controllable events), TC_1 and TC_2 should be executed simultaneously with the tester T_1 observing and controlling IUT_1 and the lower tester LT_2 of T_2 observing IUT_2 (see figure 1).
In the second example, TC_1 and TC_2 can not be executed simultaneously. The most difference comes from the loop that returns to initial state in TC_2 (state 0 to state 2). There is no corresponding loop in TC_1. Thus, TC_2 is longer to execute than TC_1. TC_2 does not contain controllable events. Thus, the execution of this test case needs the application of a stimulus on I_1. I_1 can send a message on its lower interface to I_2. The observations are made on I_2 to verify TC_2.

7 Conclusion

In this paper, we propose formal interoperability definitions called iop criteria that give the conditions to be verified by two implementations in order to be considered interoperable. These two criteria (global iop criterion iop_G and bilateral iop criterion iop_B) are proved equivalent. This equivalence leads to method to generate interoperability test cases which avoids the calculation of the specification interaction, and thus the state-space explosion problem.
Future work will study the generalization of these iop criteria to a context with more than two IUT. As it is suggested by the obtained test cases, we will also consider how a distributed approach can be applied for interoperability testing.

References

[1] Sébastien Barbin, Lénaïck Tanguy, and César Viho. Towards a formal framework for interoperability testing. In M. Kim, B. Chin, S. Kang, and D. Lee, editors, *21st IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*, pages 53–68, Cheju Island, Korea, Août 2001.

[2] 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, 94. IFIP, Elsevier Science B.V.

[3] Khaled El-Fakih, Vadim Trenkaev, Natalia Spitsyna, and Nina Yevtushenko. Fsm based interoperability testing methods for multi stimuli model. In Roland Groz and Robert M. Hierons, editors, *TestCom*, volume 2978 of *Lecture Notes in Computer Science*, pages 60–75. Springer, 2004.

[4] J.C. Fernandez, C. Jard, T. Jéron, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming - Special Issue on Industrial Relevant Applications of Formal Analysis Techniques*, 29:123–146, 1997.

[5] ISO. Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework - Parts 1-7. *International Standard ISO/IEC 9646/1-7*, 92.

[6] C. Jard, T. Jéron, L. Tanguy, and C. Viho. Remote testing can be as powerful as local testing. In J. Wu, S. Chanson, and Q. Gao, editors, *Formal methods for protocol engineering and distributed systems, FORTE XII/ PSTV XIX' 99, Beijing, China*, pages 25–40. Kluwer Academic Publishers, October 1999.

[7] 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*, Lecture Notes in Computer Science, pages 46–65. Springer-Verlag, 99.

[8] J. Tretmans and E. Brinksma. Côte de Resyste–Automated Model Based Testing. In M. Schweizer, editor, *Progress 2002–3^{rd} Workshop on Embedded Systems*, pages 246–255, Utrecht, The Netherlands, Oct. 2002. STW Technology Foundation.

[9] 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, 93. IFIP Transactions.

[10] T. Walter, I. Schieferdecker, and J. Grabowski. Test architectures for distributed systems : state of the art and beyond. In Petrenko and Yevtushenko, editors, *Testing of Communicating Systems*, volume 11, pages 149–174. IFIP, Kap, Sep. 98.