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. Dans le contexte des protocoles réseaux, le test d'interopérabilité est utilisé pour vérifier si deux (ou plus) implémentations communiquent correctement tout en fournissant les services décrits dans les spécifications correspondantes. Le but de cet article est de fournir une méthode pour la génération de tests d'interopérabilité basée sur une définition formelle de la notion 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 combinatoire du nombre d'états que rencontrent les approches classiques.

ABSTRACT. In the context of network protocols, interoperability testing is used to verify that two (or more) implementations communicate correctly while providing the services described in their respective specifications. This study is aimed at providing a method for interoperability test generation based on formal definitions. Contrary to previous works, 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

In the network domain, protocol implementations must be tested to ensure that they will be working in an operational environment. They must be able both to communicate with other protocol implementations and to offer the service described in their specification. Differents kinds of tests are used to verify that implementations are able to work "correctly". Among these tests, functional tests are used to verify the external behavior of protocol implementations. These tests consider an implementation as a black-box. This means that testers only knows the implementation via events executed on its interfaces.

Among these functional tests, conformance testing is a kind of test that determines to what extent a single implementation of a standard conforms to its requirements. Conformance testing is precisely characterized with testing architectures, a standardized framework ISO/IEC IS9646 [1], formal definitions [2] and tools for generating automatically tests [3, 4].

In this paper, we consider another kind of functional testing called interoperability testing. Its objectives are to verify both that different implementations can communicate correctly and that they are provided the service described in their respective specification while interacting. This test 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 to conformance testing, but studies on formal approach for conformance testing has increased the knowledge in this area. Despite a large literature on the interest of providing a formal approach for interoperability testing (for example [5, 6]), only few tentatives of formal definitions or automatic method of interoperability test generation have been proposed [7, 8, 9].

In this study, we consider a context of interoperability called one-to-one context. This context is used to test the interoperability between two systems. It is the interoperability testing context the most used in practice to test, either the interoperability of two implementations or the interoperability of one implementation with another system (seen as the second implementation) composed of different entities.

The aims of the study presented here are double. First, we give formal definitions of the one-to-one interoperability notion. Contrary to a previous work [10], these definitions manage quiescence. They are called *interoperability criteria* (or *iop criteria* in the following). An interoperability criterion formally describes conditions under which different implementations (two in this study) can be considered interoperable. 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 a model of the specification interaction that may lead to the well-known state-explosion problem [11].

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 context

Interoperability testing is a kind of test used to verify that protocol implementations are able to communicate correctly while providing the services described in their respective specification. Two contexts can be differentiated for interoperability testing: one-to-one context (interaction of exactly two implementations) and a more general situation with N ($N > 2$) implementations. This study focus on the one-to-one context. Moreover, in interoperability testing, each implementation is seen as a black-box. This means that testers only know an implementation from its interaction with the environment or with other systems, thus from the events that are executed on the interfaces of the implementation.

In this study, we consider a System Under Test (SUT) composed of two implementations under test (IUT for short). The general testing architecture of this one-to-one interoperability context is represented in figure 1.

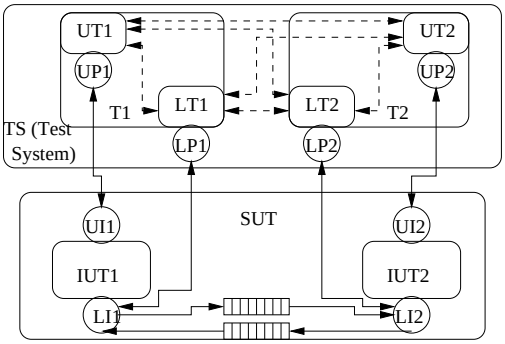

Figure 1. *Interoperability testing architecture*

In interoperability context, each of the IUTs has two kinds of interfaces. The *lower interfaces* LI_i are the interfaces used for the interaction of the two IUTs. These interfaces are only observable but not controllable.It means that a tester (LT_i) connected to such interfaces can only observe the events but not send stimuli to these interfaces. The *upper 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.

In some situations as testing of embedded systems (or for confidentiality reasons), certain interfaces may not be accessible for the testers. Thus, different interoperability testing architectures can be distinguished depending on the access to the interfaces. For example, the interoperability testing architecture is called *lower* (resp. *upper*) if only the lower interfaces (resp. the upper interfaces) are accessible and *total* if both kinds of interfaces are accessible. It is called **unilateral** if only the interfaces of one IUT (on the two interacting IUTs) are accessible, **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.

The interaction between the two IUTs is asynchronous (cf. Section 3.2). This is generally modeled by an input FIFO queue for each lower interface.

3. Formal background

This section describes the different formal notations used in the following of the paper.

3.1. IOLTS model

We use IOLTS (Input-Output Labeled Transition System) [12] to model specifications. As usual in the black-box testing context (tests based on what is observed on the interfaces of the IUTs), implementations are also modeled, even though their behaviour are normally unknown. They are also represented by an IOLTS.

Definition 3.1 *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 system interfaces.* $p?m$ *stands for an input and* $p!m$ *for an output where* p *is the interface on which the event is executed and* m *the message.*
• $\Delta^M \subseteq Q^M \times (\Sigma^M \cup \tau) \times Q^M$ *is the transition relation, where* $\tau \notin \Sigma^M$ *denotes an internal event. A transition is noted* $q \xrightarrow{\alpha}_M q'$ *or* $(q, \alpha, q') \in \Delta^M$.

Σ^M can be decomposed as follow: $\Sigma^M = \Sigma^M_U \cup \Sigma^M_L$, where Σ^M_U (resp. Σ^M_L) is the set of messages exchanged on the upper (resp. lower) interface. Σ^M can also be decomposed to distinguish input (Σ^M_I) and output (Σ^M_O) messages.
Let us consider a state q, an IOLTS M and an event $\mu \in \Sigma^M$:
• $Traces(q)$ is the set of possible observable traces (successions of events) from q.
• q *after* σ is the set of states that can be reached from q by the trace σ. By extension, all the states reached from the initial state of the IOLTS M is (q_0^M *after* σ) and is noted by (M *after* σ).
• $\Gamma(q)$ 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 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.
• Considering a link l between the lower interfaces l_i of M_i and l_j of M_j, the mirror $\bar{\mu}$ of an event μ is defined by $\bar{\mu}$=$l_i!a$ if $\mu = l_j?a$ and $\bar{\mu} = l_i?a$ if $\mu = l_j!a$.

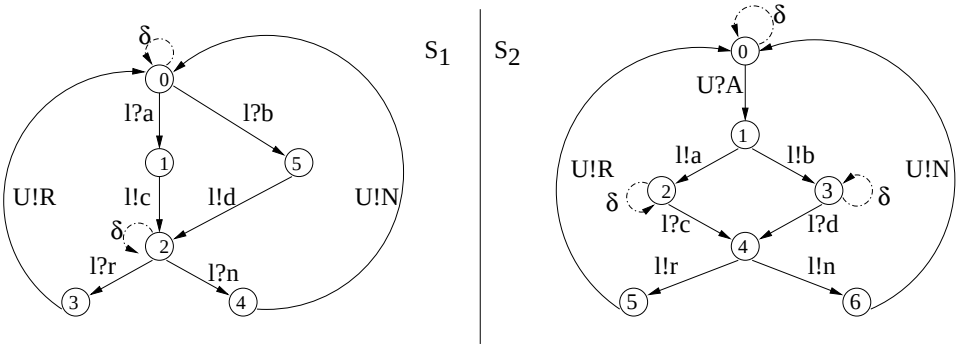

Figure 2. *Specifications* S_1 *and* S_2

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: Three situations can 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 for an IOLTS M or $\delta(i)$ for an IOLTS M_i. It is treated as an observable output event and observed practically using timers. The IOLTS obtained after calculating quiescence is called suspensive IOLTS [2] and noted $\Delta(M)$. On the examples of Figure 2, quiescence is possible and modeled on these specifications (see in states 0 and 2 of S_1, 0, 2 and 3 of 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 considering the interoperability testing architecture of Figure 1, we need to model the asynchronous interaction of two entities. For two IOLTS M_1 and M_2, this interaction is noted $M_1 \|_{\mathcal{A}} M_2$. This operation is calculated in two steps. First, $\Delta(M_1)$ and $\Delta(M_2)$ are transformed into IOLTS representing their behaviour in an asynchronous environment modeled with FIFO queues as in [12]. Then, these two IOLTS are composed to obtain $M_1 \|_{\mathcal{A}} M_2$. This operation preserve quiescence and $\delta(i)$ (resp. δ) corresponds to quiescence of M_i (resp. of the two IOLTS).

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. Implementation model

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 IUTs interact asynchronously and testers are connected to their interfaces. These testers can observe messages exchanged via lower interfaces. But they can not differentiate a message that was sent to the lower interfaces of one of the IUTs to a message actually received by this IUT. Thus, testers can only know which messages were sent by the interacting IUT. To model this situation, we choose to complete the model of an IUT with inputs corresponding to the output alphabet (lower interfaces) 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.

3.4. Specification model

As we are concerned by interoperability testing, the considered specifications must allow interaction. We call this property the interoperability specification compatibility

property (iop-compatibility property for short). Two specifications are iop-compatible iff, for each possible output on the interfaces used for the interaction after any trace of the interaction, the corresponding input is foreseen in the other specification. In a formal way : $\forall \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $\forall \sigma.a.\sigma' \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $a \in Out_{\Sigma_L}(S_1 \|_{\mathcal{A}} S_2, \sigma)$, $\sigma' = \beta_1 ... \beta_l$, $\Rightarrow \exists \beta_i$ such that $\beta_i = \bar{a}$.
In some situations (underspecification of input actions particularly), the two specifications need to be completed to verify this property. It is done by adding transitions leading to an error trap state and labeled with the inputs corresponding to messages that can be sent by the interacting entity (input m added in $In(S_j, \sigma/\Sigma_j)$ if $\bar{m} \in Out_L(S_i, \sigma/\Sigma_i)$). Indeed, this method considers the reception of an unspecified input as an error. This is the most common definition of unspecified inputs in network protocols. In the following, we will consider that specifications are iop-compatible.

4. Interoperability criteria

According to the different possible testing architecture (see Section 2), different notions of interoperability can be used [10]. In Section 4.1, we introduce interoperability formal definitions called *interoperability (iop) criteria*. An interoperability criterion formally describes the conditions that two IUTs (in the one-to-one context) must satisfy in order to be considered interoperable. Thanks to quiescence management [13], these iop criteria detect more non-interoperability situations than the "interoperability relations" defined in [10]. Moreover, in Section 4.2, we prove the equivalence -in terms of non-interoperability detection- between the most commonly used in practice iop criterion iop_G and the so called bilateral iop criterion iop_B.

4.1. Definitions of the interoperability criteria

We will describe here iop criteria considering events on both kinds of interfaces. In the same way, iop criteria considering only lower (or only upper) interfaces can be defined using projection on the considered interfaces.

The **unilateral iop criterion** focus on one of the IUTs during its interaction with the other implementation. This criterion is in the definition 4.1.

Definition 4.1 (Unilateral iop criterion iop_U)
Let I_1, I_2 two IUTs implementing respectively S_1, S_2. $I_1\ iop_U\ I_2 =_{\Delta}$
$\forall \sigma_1 \in Traces(\Delta(S_1)), \forall \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2), \sigma/\Sigma^{S_1} = \sigma_1 \Rightarrow Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}, \sigma_1)$
$\subseteq Out(\Delta(S_1), \sigma_1)$.

The unilateral iop criterion compares the events executed by one of the IUTs during its interaction with the second with the events described in the specification of this IUT. 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.

Let us consider the implementations of Figures 3. I_1^1 implements the specification S_1 of figure 2, and I_1^2 and I_2^2 are implementation of S_2. We have the following results: $I_1^1 iop_U I_1^2$, $I_1^2 iop_U I_1^1$, $I_1^1 iop_U I_2^2$, but $\neg I_2^2 iop_U I_1^1$. Notice that this non-interoperability situation is detectable only with quiescence management. Indeed, no non-authorized output

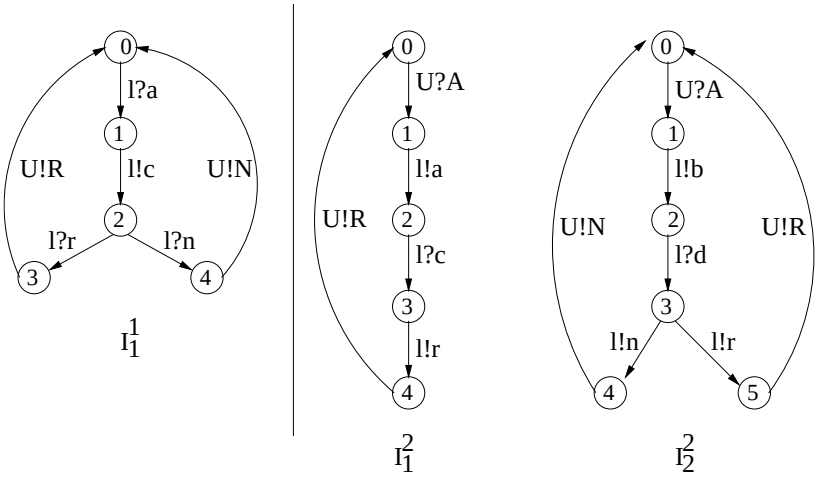

Figure 3. *Implementations I_1^1 of S_1 and I_1^2 and I_2^2 of S_2*

is executed by the IUT: the non-interoperability is due to a non-allowed quiescence after the reception of message $l?c$ by I_2^2.

The **bilateral iop criterion** considers both IUTs but separately. This criterion (definition 4.2) is verified iff both unilateral criteria, in I_1 and in I_2 point of view, are verified.

Definition 4.2 (Bilateral iop criterion iop_B)
Let I_1, I_2 two IUTs implementing respectively S_1, S_2. $I_1\ iop_B\ I_2 =_\Delta$
$\forall \sigma_1 \in Traces(\Delta(S_1)), \forall \sigma \in Traces(S_1 \|_\mathcal{A} S_2),\ \sigma/\Sigma^{S_1} = \sigma_1 \Rightarrow$
$Out((I_1 \|_\mathcal{A} I_2)/\Sigma^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1), \sigma_1)$
and $\forall \sigma_2 \in Traces(\Delta(S_2)), \forall \sigma' \in Traces(S_2 \|_\mathcal{A} S_1),\ \sigma/\Sigma^{S_2} = \sigma_2 \Rightarrow$
$Out((I_2 \|_\mathcal{A} I_1)/\Sigma^{S_2}, \sigma_2) \subseteq Out(\Delta(S_2), \sigma_2).$

On the example of Figure 3, we have $I_1^1 iop_B I_1^2$ and $\neg I_1^1 iop_B I_2^2$.

The **global iop criterion** (cf. definition 4.3) considers both kinds of interfaces and both IUTS globally.

Definition 4.3 (Global iop criterion iop_G)
Let I_1, I_2 two IUTs implementing respectively S_1, S_2.
$I_1\ iop_G\ I_2 =_\Delta \forall \sigma \in Traces(S_1 \|_\mathcal{A} S_2),\ Out(I_1\ \|_\mathcal{A} I_2, \sigma) \subseteq Out(S_1 \|_\mathcal{A} S_2, \sigma)$

The global iop criterion compares events executed by the IUTs during during their interaction with events described in the model of the interaction of the specifications. 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.

On the example of Figure 3, we have $I_1^1 iop_G I_1^2$ and $\neg I_1^1 iop_G I_2^2$.

Contrary to iop_U and iop_B that are used in some specific contexts where some interfaces are not accessible, this iop criterion iop_G corresponds to the most used testing architecture. The next section focuses on the comparison between these iop criteria in terms of non-interoperability detection.

4.2. Comparison between iop criteria

Comparisons between iop criteria are developed in [13]. As it only considers events executed by one of the two IUTs, it is easy to see that the unilateral iop criterion has less non-interoperability detection power than the bilateral and global iop criteria.
The most important result in this comparison is the following theorem 4.1 stating that iop_G and iop_B are equivalent.

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

The three following lemmas are needed for proving this theorem. These lemons consider two IOLTS M_1 and $_2$.

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

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 $a \in \Sigma^{M_1}$ (i.e. $a \in Out(\Delta(M_1), \sigma/\Sigma^{M_1})$) or $a \in \Sigma^{M_2} \cup \{\delta, \delta(2)\}$ (i.e. $a \in Out(\Delta(M_2), \sigma/\Sigma^{M_2})$).
Thus, $Out(M_1 \|_{\delta} M_2, \sigma) \subseteq Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$.
2) In the other sense, the definition of the asynchronous interaction leads to : $Out(M_1 \|_{\mathcal{A}} M_2, \sigma)$ $\supseteq Out(\Delta(M_1),\ \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$. $\diamondsuit$

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 $\Sigma^{(M_1 \|_{\mathcal{A}} M_2)}/\Sigma^{M_1} \subseteq \Sigma^{M_1}$. $\diamondsuit$

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. Preliminary definitions

5.1.1. Interoperability test purpose

A interoperability (iop) 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 System Under Test (SUT). It supposes that any sequences of actions foreseen in the specification may occur between two consecutive actions of a test purpose.

5.1.2. Interoperability test case

In the tester point of view, two kinds of events are possible during conformance tests: sending a stimuli to the Implementation Under Test (IUT) or receiving an input from this IUT. 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 interoperability (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. Notice that in interoperability testing μ can be either an input or an output.

5.1.3. Interoperability verdicts

The execution of an iop test case TC on system composed of the two implementations ($SUT(I_1 \|_{\mathcal{A}} I_2)$) gives an interoperability (iop) verdict : $verdict(TC, SUT) \in \{PASS, FAIL, INC\}$. 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.

5.2. 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 IUTs. Data used to generate test cases are the specifications and a test purpose. In this Section, we present classical approaches. These approaches are based on a definition of interoperability that corresponds to the global iop criterion iop_G, that is to say on a comparison between events observed during the implementation interaction and events described in a model of the specification interaction.

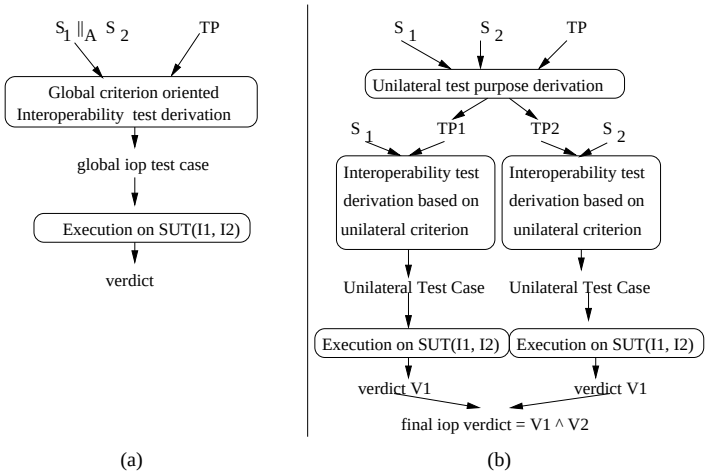

Figure 4. *Approaches for interoperability test generation*

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 [8, 14, 15, 16, 17]) 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 [7, 11] 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.

In classical approaches (see figure 4(a)), the test generation algorithm 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.

One problem with this method (classical method) is that we can have state space explosion when calculating the interaction of the specifications [11]. Indeed, the state number of the specification asynchronous interaction is in the order of $O((n.m^f)^2)$ where n is the state number of the specifications, f the size of the input FIFO queue on lower interfaces and m the number of messages in the alphabet of possible inputs on lower interfaces. Thus, interoperability test generation based on the global iop criterion may be impossible even for small specifications.

5.3. Method based on the bilateral iop criterion iop_B

The equivalence -in terms of non-interoperability detection- between the iop criteria iop_B and iop_G (cf. theorem 4.1) suggests to study a bilateral iop criterion based method.

5.3.1. Principles of the bilateral criterion based method

Let us consider an iop test purpose TP. The bilateral method can be decomposed in two main steps: cf. Figure 4(b). The first step is the derivation of two unilateral interoperability 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. According to the theorem 4.1: $verdict(TC, I_1 \|_{\mathcal{A}} I_2) = verdict(TC_1, I_1 \|_{\mathcal{A}} I_2) \wedge verdict(TC_2, I_1 \|_{\mathcal{A}} I_2)$. The verdicts obtained by the execution of the iop test cases on the SUT $verdict(TC, I_1 \|_{\mathcal{A}} I_2)$, $verdict(TC_1, I_1 \|_{\mathcal{A}} I_2)$ 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: $PASS \wedge PASS = PASS$, $PASS \wedge INC = INC$, $PASS \wedge FAIL = FAIL$, $INC \wedge FAIL = FAIL$, $INC \wedge INC = INC$ and $FAIL \wedge FAIL = FAIL$.

5.3.2. Unilateral test purpose generation

The first step of the algorithm consists in deriving TP_{S_1} and TP_{S_2} from TP. Recall that test purposes are generally abstract (see in Section 5.1). 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 obtained TP_{S_1} (resp. TP_{S_2}) contains only events of S_1 (resp. S_2).
The way to derive TP_{S_1} and TP_{S_2} from TP is described in the following algorithm (see figure 5). If all the events described in TP are to be executed on the lower interfaces, the algorithm is very simple. If TP contains an event μ_i on the upper interfaces, this algorithm needs to go through the IOLTS representing the other specification (S_k in the algorithm). It finds a path between μ_{i-1} (or $\bar{\mu}_{i-1}$) and μ_i in the interacting specification. This operation is however simple compared to the construction of the specification interaction for classical approach.

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

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, $\bar{a}_j, q') \wedge \sigma = \bar{a}_j.\omega.\mu_j \in Traces(q)\}$
  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**

Figure 5. *Algorithm to derive TP_{S_i} from TP*

5.3.3. Unilateral test case derivation

The second step of the bilateral criterion based method consists in using a conformance test generation tool (for example TGV [3]) to derive two unilateral iop test cases TC_1 and TC_2 (see figure 4(b)). The test cases derived by the conformance test generation tool are modified in order to take into account the differences between interoperability and conformance contexts: in conformance context, testers can control (send messages to the IUT) lower interfaces while in interoperability context, these interfaces are only observable. 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.

5.3.4. Few words about complexity

The first step of the method proposed here (cf. figure 4(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 [3]. Thus, it costs less than calculating $S_1 \|_{\mathcal{A}} S_2$ with the classical method based on a global iop criterion. Moreover, if an iop test case can be obtained using the classical approach, the proposed method based on iop_B can also generate an equivalent bilateral iop test case.

5.4. Applying the method to an example

Let us consider the two specifications S_1 and S_2 of figure 2 (Section 3.1). Let us consider the following interoperability testing purpose : " after the implementation I_1 sends the message a on the lower interface l_1, the other implementation I_2 will send the message N on its upper interface U_2. This test purpose is interesting because it contains events on

both interfaces and both IUTs. In an abstract way, TP is defined by $TP = l1?a.U2!N$.

First step : Deriving TP_{S_1} and TP_{S_2} from TP
$\mu_1 = l1?a$ and $\mu_2 = U2!N$ in the algorithm. The derivation of TP_{S_2} from TP is easy : $TP_{S_2} = \bar{\mu}_1.\mu_2 = l2!a.U2!N$.
Deriving TP_{S_1} from TP needs some few explanation. According to the algorithm, $\mu_1 = l1?a \in \Sigma^{S_1}$. Thus, $TP_{S_1} = l1?a$. As $\mu_2 = U2!N \in \Sigma_U^{S_2}$, we have $a_j = l1?a$, $\bar{a}_j = l2!a$ and $\omega = l2?c.l2!n$. Thus, last_event(ω)=$l2!n \in \Sigma_L^{S_2} \Rightarrow TP_{S_1}$=$l1!a.l1?n$.

Second step : Generating iop test cases
The obtained test cases TC_1 and TC_2 using TGV [3] are given in Figure 6. $UT1$ and $UT2$ and the testers connected respectively to $U1$ and $U2$. The notation $?(\mu)$ is used to represent the observation of event μ. $(PASS)$ is a temporary verdict and $PASS$ is the definitive verdict obtained after a postamble that makes the IUT return to the initial state.

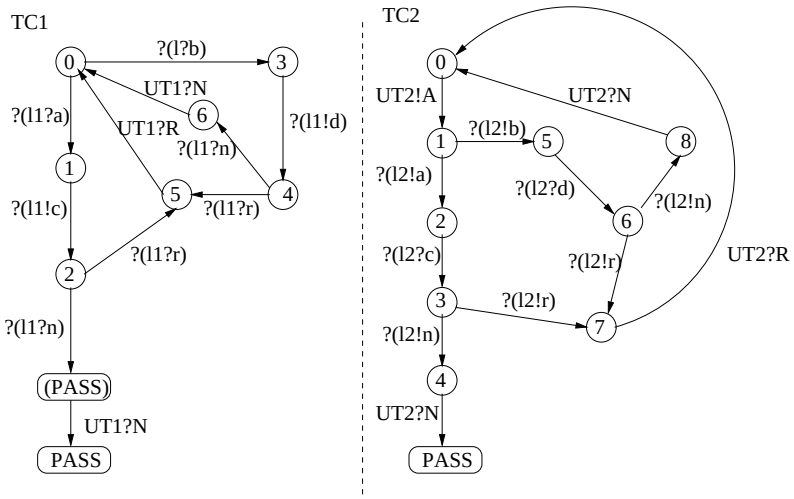

Figure 6. *Interoperability test cases obtained for $TP = l1?a.U2!N$: bilateral method*

For interoperability test case generation based on the global relation, the obtained TC are given in Figure 7: $UT1?N, UT2?N$ means that these two events can be executed in any order while $?(!1?n).UT1?N$ means that $?(!1?n)$ must be executed before $UT1?N$. These test cases comes from the composition of $S_1 \|_A S_2$ with TP.

According to the theorem 4.1, final interoperability verdicts obtained with TC_1 and TC_2, executed simultaneously or not on the SUT, should be the same as the verdict obtained with TC. Indeed, a look at glance to TC_1 and TC_2 shows the same paths and verdicts as in TC.

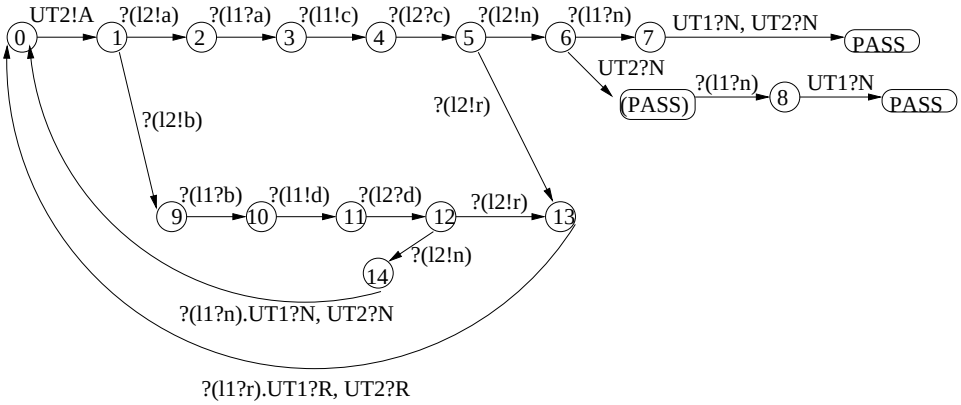

Figure 7. *Interoperability test case obtained for $TP = l1?a.U2!N$: global method*

6. Conclusion

In this paper, we focus on interoperability testing of protocol implementations. We consider a one-to-one interoperability testing context. We give formal definitions of the interoperability notion. These formal definitions take into account quiescence that can be observed when testing interoperability and are called *interoperability criteria* An interoperability criterion formally describes conditions under which two IUTs can be considered interoperable. Different iop criteria are defined in order to take into account the different configurations (i.e. the testing architectures) in which the interoperability of two implementations can be tested.

In this study, we also prove a theorem stating that two of the defined iop criteria are equivalent in terms of non-interoperability detection. This equivalence allows a new method for interoperability test generation that avoids the classical state-space explosion problem. Indeed, the so-called bitaleral approach developped based on the bilateral iop criteria is not based on a model of the specification interaction contrary to classical methods but considers each implementation separatly during their interaction. Thus, this bilateral interoperability test derivation method allows us to generate interoperability test cases in situations where it would have been impossible with the traditional method because of state space explosion problem.

The obtained test cases suggest a distributed approach for interoperability testing. Indeed, the so-called bilateral testing architecture corresponds to a context of distributing interoperability testing without synchronization between the testers connected to the two IUTs. Thus, we will study how to apply the defined interoperability test generation method to a distributed testing context.

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. Our first results on this context show that, after the determination of the testing configuration (particularly the topology connecting the implementations) that is much more complex than in one-to-one context, a similar approach can be applied for interoperability test derivation. This approach will also consider each of the N implementations separatly during their interaction. However, such an approach needs to consider complex dependences between events executed on different

implementations in order to be able to derive N unilateral test purposes based on which the N unilateral test cases are based.

7. Bibliographie

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

[2] 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

[3] JARD, C., JÉRON, T., « 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] TRETMANS J., BRINKSMA E., « TorX: Automated Model Based Testing », *Proceedings of the First European Conference on Model-Driven Software Engineering,Nurnberg, Germany*, December 2003

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

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

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

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

[9] VAN DER BIJL M., RENSINK A., TRETMANS J., « Component Based Testing with **ioco** », *FATES 2003- Formal Approaches to Testing of Software, volume 2931 of LNCS*, 2004

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

[11] R. CASTANET, O. KONÉ, « Deriving coordinated testers for interoperability », *In O. Rafiq, editor, Protocol Test Systems, volume VI C-19, Pau-France*, 1994

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

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

[14] EL-FAKIH K., TRENKAEV V., SPITSYNA N., YEVTUSHENKO N., « FSM Based Interoperability Testing Methods for Multi Stimuli Model », *TestCom 2004, Lecture Notes in Computer Science* 2004

[15] GRIFFETH N., HAO R., LEE D., SINHA R., « Integrated System Interoperability Testing with Applications to VOIP », *FORTE/PSTV 2000, Kluwer, B.V.* 2000

[16] BOCHMANN G., DSSOULI R., ZHAO J., « Trace analysis for conformance and arbitration testing », *IEEE transaction on software engineering, number 11, volume 15*, November 1989

[17] GADRE J., ROHRER C., SUMMERS C., SYMINGTON S., « A COS Study of OSI Interoperability », *Computer standards and interfaces, number 3, volume 9* 1990