TOWARDS A FORMAL FRAMEWORK FOR INTEROPERABILITY TESTING*

César Viho, Sébastien Barbin and Lénaïck Tanguy
IRISA/IFSIC - Université de Rennes I, Campus de Beaulieu, 35042 Rennes, France
{viho,ltanguy,sbarbin}@irisa.fr

Abstract This paper shows how the existing concepts of conformance testing can be used to define a formal framework for interoperability testing. First, the different possible interoperability testing architectures are discussed. Then, we define several *interoperability relations* based on the existing relations defined for conformance testing. A comparison of these relations is given, in terms of their power to detect non-interoperability. Some guidelines are given to help in generating interoperability tests.

Keywords: interoperability, test, architecture, relation, protocol, conformance

1. INTRODUCTION

In the context of distributed systems, there are basically two approaches to testing implementations to ensure that they will work effectively together. Conformance testing determines whether a single implementation under test (IUT) conforms to its specification (generally a standard) or not. In contrast, interoperability testing determines the ability of two or more implementations to work together in a real operational environment. As we can see, the purposes of these two kinds of tests are not really the same. While conformance testing evaluates an implementation in terms of its correspondence to a specific standard, interoperability testing compares an implementation with other products.

The words "interoperability", "interwork", "interoperate" are often used in the descriptions of computer systems. Different needs of interoperability testing can be identified. The most basic is to put together two implementations built by different vendors and verify that they "interwork" correctly. Another common situation is the so called "one against N" interoperability testing in which there already exist N (N≥1)

*This work has been supported by the CELAR/TCOM Contract 9842.561 MTI (Méhodologie de test d'interopéabilité), the European ITEA Project 99011 RTIPA, and the CNRS/Programme Télécommunications 99N36/0029

systems working together. One may wonder if a new system S will interoperate with the N existing systems. In other words, the question here is "will the N+1 systems still work together ?". There exist many other situations which require interoperability. Thus, it is difficult to give a unique definition of this notion. Indeed, several definitions of interoperability exist [1, 2, 3, 4]. However, the functional meaning is clear: the components of the system under test (SUT) have to communicate with each other correctly and provide the expected services.

A lot of work have been done in the area of conformance testing [5] leading to a precise definition of conformance [6]. The so called conformance/implementation relations in [7, 8] give formal characterizations of conditions under which an IUT can be considered as conformant to its specification. This allows the automatic generation of conformance tests [9]. Unfortunately, such work is lacking for interoperability testing [3]. There are several reasons for this situation. The first reason is because conformance of implementations has been considered a prerequisite for achieving interoperability. Thus, a very important effort has been concentrated on conformance testing. Another reason is that interoperability testing is considered a pragmatic and practical requirement, as it relates to implementations; contrary to conformance testing where the specification serves as reference.

The fact that both conformance and interoperability testing concerns the same objects (specification, IUT, etc) is evident. Thus, most of the concepts, methodologies and theory developed for conformance testing can be adjusted for interoperability testing. This simple observation suggests to adapt the existing formal conformance testing framework for interoperability testing. Following this idea, the work presented in this paper proposes a formal framework (architectures, interoperability relations, tests generation) for interoperability testing.

This paper is structured as follows. Section 2 presents the different classes of interoperability testing architectures. For each class, contexts in which an architecture can be used are explained. We give an indication of how the existing architectures can be positioned in regard to these classes. The model and notations used are described in Section 3. Section 4 gives formal definitions of interoperability relations, that are based on the existing relations defined for conformance testing. Each interoperability relation specifies formally the conditions to be satisfied by two implementations in order to be considered interoperable. These relations are compared in terms of their power to detect non interoperability. In Section 5, we give guidelines to generate interoperability tests for the proposed architectures and relations. Conclusion and future work are in Section 6.

2. INTEROPERABILITY ARCHITECTURES

Depending on how the implementations to be tested are interconnected, and depending on the degree to which we can observe their interaction, different interoperability testing architectures can be used. Indeed, a lot of architectures have been proposed [1, 2, 4, 10].

2.1. Definition of testing architectures

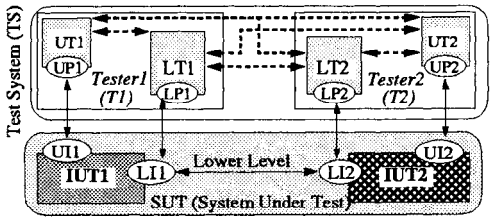

Figure 1. General architecture of interoperability testing

Let us consider the general architecture of figure 1 where the SUT is composed of two IUTs (IUT_1 and IUT_2). Each IUT is a black-box. The **Upper Interface** UI_i (resp. **Lower Interface** LI_i) is the interface of the IUT_i, through which it communicates with its upper (resp. lower) layer. The expected services are furnished through UI_i while LI_i is used by IUT_i to interact with the peer entities in the same layer.

Depending on the environment in which the interoperability testing will be done, different levels of control and observation of the SUT are possible. Thus, the **Test System** (TS) in charge of testing, via its PCOs (points of control and observation), the interoperability of this SUT can consist of some or all of these following elements. The **Upper Tester** i UT_i (resp. **Lower Tester** i resp.LT_i) is part of TS in charge of the control and/or the observation of UI_i (resp. LI_i), via the **Upper PCO** UP_i (resp. the **Lower PCO** LP_i). The **Tester** T_i (composed by UT_i and/or LT_i) is the part of TS in charge of the control and/or observation of IUT_i .

The different possible compositions of both TS and SUT induce different kinds of interoperability testing architectures as described below. The reader can see that they are easily extendible to the general case where TS and/or SUT are composed by more than two components.

2.1.1 Unilateral interoperability testing architectures.

These architectures correspond to the situation in which the control and/or observation of only one component (IUT_1 or IUT_2) of the SUT

is possible. Thus, the decision of interoperability is based on these partial observations done by only one tester T_1 or T_2. The tester T_i may contain only LT_i (resp. UT_i). This can happen if the upper (resp. lower) interface is embedded under an upper (resp. lower) layer protocol. In this case, the architecture is called Unilateral **Lower** (resp. **Upper**) Interoperability Testing Architecture. The Unilateral **Total** Interoperability Testing Architecture corresponds to the case where the tester T_i contains both UT_i and LT_i. In this case, a test coordination procedure (TCP) may exist between UT_i and LT_i.

These architectures are used when the other IUTs which compose the SUT are embedded (i.e., all or part of their interfaces are not accessible) or in a "one against N" interoperability testing context (see section 1).

2.1.2 Bilateral interoperability testing architectures.

In these architectures, each tester T_i realizes separately the control and observation of the corresponding IUT_i, using a Unilateral Interoperability Testing Architecture. Depending on the composition of the testers, the terms **Upper, Lower** and **Total** apply also here.

This architecture is interesting in the unusual situations where the designers of the IUTs require their own control and opinion of how their respective IUT interact with the others. This happens for example when a certain level of confidentiality is still required by each of the vendors [4].

2.1.3 Global interoperability testing architectures.

In these architectures, the control and observation are done globally by the two testers T_1 and T_2 on both sides (IUT_1 and IUT_2) of the SUT. They are called **global** because the decision of interoperability is achieved according to a global view (simultaneously on both sides) of the SUT. This is the main difference with the previous architectures, as a test coordination procedure exists between the testers. In this sense, it is the more complex architecture. Global upper, lower, and total architectures are defined as for bilateral architectures.

2.1.4 Hybrid interoperability testing architectures.

Some situations of interoperability combine previously defined architectures. All the architectures which are not strictly unilateral, bilateral or global architectures are called **hybrid** architectures. As an example, the architecture in which we have only the upper tester in tester T_1 and upper and lower testers of the tester T_2, is typically a hybrid architecture. This kind of architecture can be encountered in the "testing against reference" context where one of the implementations serves as the reference, the upper tester over the tested IUT is used as a responder.

2.2 Positioning the existing architectures

Many interoperability architectures and associated terminologies can be found in the literature [1, 2, 4, 10]. In the following, we give elements which help in understanding why there is so much confusion in the state of the art of interoperability testing architectures. We show that each of the existing architectures can be associated with one of the canonical architectures described above.

Black-box versus grey-box testing
It is well-known that conformance testing is done in a black-box context. But there is still a debate about considering interoperability testing as black-box or grey-box testing [2]. The contested point comes from the possibility for the interoperability testing architecture to observe or not (and/or control) the interactions between the implementations which compose the SUT. Our purpose is not to close the debate, but one can see that "grey-box" architectures correspond to architectures which include lower testers. In contrast, the so called "black-box" architectures are those which do not contain any lower tester.

Protocol versus service interoperability testing
The definition of interoperability can be "layer-oriented" [4, 11]. For example in [4], they consider four levels of interoperability called respectively protocol, service, application and user-perceived interoperability architectures. It is easy to see that protocol interoperability architectures correspond to a lower interoperability testing architecture, while service (application or user perceived) interoperability architectures can be definitely classified as upper interoperability testing architectures.

Active versus passive testing architectures
If the test system has the possibility to give stimuli (control) and/or to corrupt events in the SUT, the corresponding testing architectures are called active [1]. If not, they are called passive architectures [10, 11]. Thus, any possible architecture (including those proposed in this paper) can be considered either as active or passive.

Monitor versus arbiter
In these architectures [10], a test component is placed between the implementations to control (monitor) or observe (arbiter) their communication. To be monitor or arbiter depends on the active/passive behavior of the tester. Thus, any architecture proposed in this paper that contains at least one lower tester can be used as monitor or arbiter. Arbiters are used to identify faulty implementations [12].

3. MODELS AND NOTATIONS

In order to give formal definitions of the notion of interoperability, we need a model which allows the formal description of all the elements involved in the interoperability testing activity. As said before, most of the terms and components in interoperability testing are similar to those used in conformance testing. We choose the commonly used model of the IOLTS (Input-Output Labeled Transition System) [13] to model specifications as well as implementations.

3.1. IOLTS, definitions and notations

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 interaction points (with the environment) of the system. Thus, we have :* $\Sigma^M \subseteq P^M \times \{?,!\} \times A^M$ *where* P^M *is the finite set of interaction points (ports) through which the system communicates with its environment (lower or upper layer, or other systems), "'?" and "!" respectively denote a 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* $(q, \alpha, q') \in \Delta^M$.

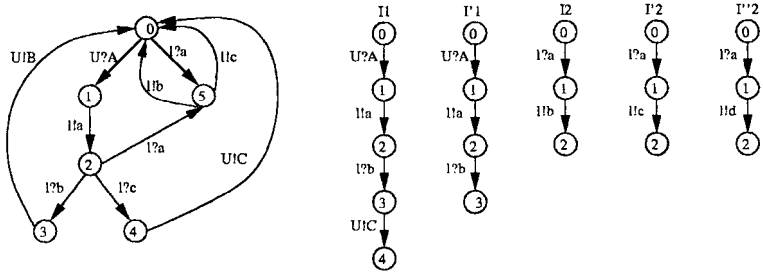

Figure 2. A specification S and possible implementations I_1, I'_1, I_2, I'_2, I''_2

Σ^M can be decomposed as follow: $\Sigma^M = \Sigma_U^M \cup \Sigma_L^M$, where Σ_U^M (resp. Σ_L^M is the set of messages exchanged on the upper (resp. lower) interface. Σ^M can be also decomposed in order to distinguish input messages from output messages. $\Sigma^M = \Sigma_I^M \cup \Sigma_O^M$ where Σ_I^M (resp. Σ_O^M) is the finite set of input (resp. output) messages.

In the following we denote the set of all input/output labeled transition systems by *IOLTS*. Let us consider an IOLTS $M \in IOLTS$, and let $\alpha \in \Sigma^M$ with $\alpha = p.\{?,!\}.m$, $\mu_i \in \Sigma^M \cup \tau$, $\sigma \in (\Sigma^M)^*$, $q, q', q_i \in Q^M$:

- $q \xrightarrow{\mu_1 \cdots \mu_n}_M q' =_\Delta \exists\, q_0 = q, q_1 \ldots, q_n = q', \forall i \in [1,n], q_{i-1} \xrightarrow{\mu_i}_M q_i$.
- $q \overset{\epsilon}{\Rightarrow}_M q' =_\Delta q = q'$ or $q \xrightarrow{\tau \ldots \tau}_M q'$.
- $q \overset{\alpha}{\Rightarrow}_M q' =_\Delta \exists\, q_1, q_2, q \overset{\epsilon}{\Rightarrow}_M q_1 \xrightarrow{\alpha}_M q_2 \overset{\epsilon}{\Rightarrow}_M q'$.
- $q \overset{\sigma}{\Rightarrow}_M q' =_\Delta q \xrightarrow{\mu_1 \cdots \mu_n}_M q' =_\Delta \exists\, q_0 = q, q_1 \ldots, q_n = q', \forall i \in [1,n], q_{i-1} \overset{\mu_i}{\Rightarrow}_M q_i, \sigma = \mu_1 \cdots \mu_n$.
- $out(q) =_\Delta \{\alpha \in \Sigma_O^M \mid \exists\, q' \text{ and } q \xrightarrow{\alpha}_M q'\}$ is the set of outputs from q.
- $q\ after\ \sigma =_\Delta \{q' \in Q^M \mid q \overset{\sigma}{\Rightarrow}_M q'\}$ 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 $(q_0^M\ after\ \sigma)$ and will be *noted by* $(M\ after\ \sigma)$. In the same way, $Out(M, \sigma) =_\Delta out(M\ after\ \sigma)$.
- $Traces(q) =_\Delta \{\sigma \in (\Sigma^M)^* \mid q\ after\ \sigma \neq \emptyset\}$ is the set of possible observable traces from q. And, $Traces(M) =_\Delta Traces(q_0^M)$.

In interoperability testing, we usually need to observe some specific events among all the possible traces of a SUT. These traces, reduced to the expected messages, can be obtained by a **projection** of those traces on a set representing criteria used to select the expected events.

Definition 3.2 *Let us consider an IOLTS M, a trace* $\sigma \in (\Sigma^M)^*$, $\alpha \in \Sigma^M$, *and a set X. The projection of* σ *on X is noted by* σ/X *and is defined by:* $\epsilon/X = \epsilon, (\alpha.\sigma)/X = \sigma/X$ *if* $\alpha \notin X$, *and* $(\alpha.\sigma)/X = \alpha.(\sigma/X)$ *if* $\alpha \in X$.

For example, consider a trace $\sigma = U?A.l!a.l?c.U!B \in (\Sigma^M)^*$, then $\sigma/\{l!a, U?D\} = l!a$, $\sigma/\Sigma_{rr}^M = U?A.U!B$, $\sigma/\Sigma_r^M = U?A.l?c$. Notice that $\sigma/X \in (\Sigma^M)^*$ but we do not have necessarily $\sigma/X \in Traces(M)$.

Definition 3.3 *Let us consider an IOLTS M, a trace* $\sigma \in (\Sigma^M)^*$ *and a set X, then* $Out_X(M, \sigma) =_\Delta Out(M, \sigma) \cap X$ *is the set of outputs from M after* σ), *reduced to the outputs that belong to X.*

3.2. Synchronous and asynchronous interaction

Definition 3.4 (Synchronous composition $\|_S$) *The synchronous composition of two IOLTS* M_1 *and* M_2 *is noted* $M_1 \|_S M_2 = (Q^{M_1} \times Q^{M_2}, \Sigma^{M_1 \|_S M_2}, \Delta^{M_1 \|_S M_2}, (q_0^{M_1}, q_0^{M_2}))$ *where* $\Sigma^{M_1 \|_S M_2} \subseteq \Sigma^{M_1} \cup \Sigma^{M_2}$, *and the transition relation* $\Delta^{M_1 \|_S M_2}$ *is obtained as follow,* $\forall (q_1, q_2) \in Q^{M_1} \times Q^{M_2}$,

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

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

The interaction between components of a distributed system is done through an environment. This interaction can be either synchronous, or asynchronous. As in [13], we will suppose that the (synchronous or asynchronous) environment can be modeled by an IOLTS $\mathcal{E}$. Thus, the interaction of an IOLTS S_i with another IOLTS S_j through an environment $\mathcal{E}$ is obtained by the synchronous composition $S_i ||_s \mathcal{E} ||_s S_j$. This will be noted by $S_i || S_j$ and corresponds to the reachability graph in [l, 2].

4. INTEROPERABILITY RELATIONS

In this section, we give a formal definition of the notion of interoperability based on the model of IOLTS introduced in section 3.1. To a certain extent, testing the interoperability between two implementations consists of a kind of conformance testing of their interaction with the expected behavior or service. Thus, it becomes natural to study how these conformance relations [7, 8] can be adapted to obtain interoperability relations. We will consider the most commonly used **ioconf** conformance relation [9, 13]. It states that an implementation I is conformant to its specification S if after a trace of S, outputs of I are foreseen in S.

Definition 4.1 I **ioconf** $S =_{\Delta} \forall \sigma \in Traces(S),\ Out(I, \sigma) \subseteq Out(S, \sigma)$

4.1. The notion of interoperability relation

An interoperability relation formally specifies conditions to be satisfied by two implementations in order to be considered interoperable.

Definition 4.2 *An interoperability relation is a relation R between two implementations* I_1 *and* I_2*:* $R(I_1, I_2)$ *means that* I_1 *interoperates with* I_2.

Interoperability relations are not transitive: the fact that an implementation I_1 interoperates with I_2 and that I_2 interoperates with I_3, does not allow one to draw a conclusion regarding interoperability between I_1 and I_3. The main difference between an interoperability relation and a conformance relation is that: a conformance relation connects an implementation with its specification while an interoperability relation concerns implementations. More precisely, two implementations can interoperate even though their interaction does not correspond to anything allowed in their respective specification. Thus, an interoperability relation can take into account or not the specifications on which the implementations are based. This leads to two main classes of interoperability relations called in the following **"Specification-less" interoperability relations** and **"Specification-based" interoperability relations.**

As the "Specification-less" interoperability relations do not use any specification, it is impossible to generate *a priori* tests based on these

relations. Verifying these relations can be done preliminary to other tests using specification-based relations. This can explain why it is called interconnectivity testing in [4]. As a consequence, we will no longer discuss these relations in the sequel.

4.2. Specification-based relations

Specification-based interoperability relations are relations which refers to the specification of at least one component of the SUT.

4.2.1 Specification-based lower interoperability relations.

In this section, we focus on the lower layer interfaces of the implementations. We consider the different possible situations and we propose the associated interoperability relations which can be used.

The relation R_1 considers the situation where we have the specification of only one of the two implementations. Let us consider that S_1 is the specification used to develop the implementation I_1. Now, let us consider that only the lower interface of I_1 is accessible. During the interaction between I_1 and I_2, the least we can expect from the implementation I_1 is to behave as foreseen in its specification S_1. This is described in the relation R_1. It considers the traces observed ($Traces(I_1||I_2)$) during the interaction between I_1 and I_2. It states that, after any corresponding trace in the specification S_1 of the unilaterally considered implementation (here I_1), all the outputs (sent to the other IUT on the lower layer) are allowed in S_1.

Definition 4.3 (Unilateral Lower Interoperability Relation R_1)
$\mathcal{R}_1(I_1, I_2) =_\Delta \forall \sigma_1 \in Traces(S_1), \forall \sigma \in Traces(I_1 \| I_2), \sigma/\Sigma^{I_1} = \sigma_1 \Rightarrow Out_{\Sigma_L^{I_1}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_1}}(S_1, \sigma_1)$

Remarks:

- In figure 2, $R_1(I_1, I_2'')$ but $\neg R_1(I_2'', I_1)$.
- Notice also that there is a subtle difference with the **ioconf** relation: $R_1(I_x, I_y)$ does not mean necessarily that I_x **ioconf** S_x.

The relation R_1 can be applied independently for I_2 (based on the specification S_2). When we have both $R_1(I_1, I_2)$ and $R_1(I_2, I_1)$, this corresponds to the interoperability relation R_2 defined below.

Definition 4.4 (Bilateral Lower Interoperability Relation R_2)

$$\mathcal{R}_2(I_1, I_2) =_\Delta \mathcal{R}_1(I_1, I_2) \wedge \mathcal{R}_1(I_2, I_1)$$

In figure 2, $R_2(I_1, I_2)$ but $\neg R_2(I_1, I_2'')$. In the two relations R_1 and R_2, only lower layer outputs of each IUT are considered. Typically, we can have $R_2(I_1, I_2)$, whatever happens in their interaction with upper layers.

The next relation R_3 is called **global** because it is based on the global behavior of the interactions between respectively the two specifications $(S_1||S_2)$ and the two implementations $(I_1||I_2)$. The relation R_3 states that: after a trace included in the composition of the specifications $S_1||S_2$ and observed in the interaction between the two implementations, all outputs observed on the lower layer are allowed in $S_1||S_2$.

Definition 4.5 (Global Lower Interoperability Relation R_3)
$R_3(I_1, I_2) =_\Delta$
$\forall \sigma \in Traces(S_1 \| S_2) \Rightarrow Out_{\Sigma_L^{I_1 \| I_2}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_1 \| S_2}}(S_1 \| S_2, \sigma)$.

In figure 2, $R_3(I_1, I_2)$ but $\neg R_3(I_1, I_2'')$. R_3 is a kind of reduced conformance relation in the sense that only a subset of outputs is considered.

4.2.2 Specification-based upper interoperability relations.

The following three relations are similar to the previous ones. The difference is that only observations on the upper (instead of lower) layer are used. Consequently, similar explanations and remarks apply for the three relations which are enumerated below.

◊ The **Unilateral Upper Interoperability relation** R_4 can be defined as follows: $\mathcal{R}_4(I_1, I_2) =_\Delta \forall \sigma_1 \in Traces(S_1), \forall \sigma \in Traces(I_1 \| I_2), \sigma/\Sigma^{I_1} = \sigma_1 \Rightarrow Out_{\Sigma_U^{I_1}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_U^{S_1}}(S_1, \sigma_1)$.

◊ For the **Bilateral Upper Interoperability relation** R_5, we have: $\mathcal{R}_5(I_1, I_2) =_\Delta \mathcal{R}_4(I_1, I_2) \wedge \mathcal{R}_4(I_2, I_1)$.

◊ Finally, the **Global Upper Interoperability relation** R_6 is defined by: $\mathcal{R}_6(I_1, I_2) =_\Delta \forall \sigma \in Traces(S_1 \| S_2), Out_{\Sigma_U^{I_1 \| I_2}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_U^{S_1 || S_2}}(S_1 || S_2, \sigma)$.

On the example of figure 2, we have:
- $\mathcal{R}_4(I_1, I_2'')$, $\mathcal{R}_4(I_2, I_1)$, $\mathcal{R}_4(I_2'', I_1)$ but $\neg \mathcal{R}_4(I_1, I_2)$.
- $\mathcal{R}_5(I_1, I_2'')$, $\mathcal{R}_5(I_1, I_2')$ but $\neg \mathcal{R}_5(I_1, I_2)$.
- $\mathcal{R}_6(I_1, I_2'')$, $\mathcal{R}_6(I_1', I_2)$, $\mathcal{R}_6(I_1, I_2')$ $\mathcal{R}_6(I_1', I_2')$, but $\neg \mathcal{R}_6(I_1, I_2)$.

4.2.3 Specification-based total interoperability relations.

The following relations are qualified **total** because the decision of interoperability uses observations on both upper and lower layers. These observations can be done locally/unilaterally, bilaterally or globally. As in the two sections above, we obtain the three following relations.

◊ The **Unilateral Total Interoperability relation** R_7 is defined as follows: $\mathcal{R}_7(I_1, I_2) =_\Delta \forall \sigma_1 \in Traces(S_1), \forall \sigma \in Traces(I_1 \| I_2), \sigma/\Sigma^{S_1} = \sigma_1 \Rightarrow Out_{\Sigma^{I_1}}(I_1 \| I_2, \sigma) \subseteq Out(S_1, \sigma_1)$.

◊ It is easy to obtain the **Bilateral Total Interoperability relation** R_8. It is defined by: $R_8(I_1, I_2) =_\Delta R_7(I_1, I_2) \wedge R_7(I_2, I_1)$.

◊ The definition of the **Global Total Interoperability relation** R_9 is

also straightforward. $\mathcal{R}_9(I_1, I_2) =_\Delta \forall \sigma \in Traces(S_1 \| S_2)$, $Out(I_1 \| I_2, \sigma) \subseteq Out(S_1 \| S_2, \sigma)$.

Remarks:

- In figure 2, we have $\mathcal{R}_7(I_1, I_2'')$ but $\neg\ \mathcal{R}_7(I_1, I_2)$, $\mathcal{R}_8(I_1, I_2')$ but $\neg\ \mathcal{R}_8(I_1, I_2'')$ and $\neg\ \mathcal{R}_8(I_1, I_2)$
- One can notice that $\mathcal{R}_7(I_1, I_2) = \mathcal{R}_1(I_1, I_2) \wedge \mathcal{R}_4(I_1, I_2)$, $\mathcal{R}_8(I_1, I_2) = \mathcal{R}_2(I_1, I_2) \wedge \mathcal{R}_5(I_1, I_2)$, and $\mathcal{R}_9(I_1, I_2) = \mathcal{R}_3(I_1, I_2) \wedge \mathcal{R}_6(I_1, I_2)$.
- $R_9(I_x, I_y)$ corresponds to $I_x \| I_y$ **ioconf** $S_x \| S_y$. We will discuss this point later in Section 5.

4.3. Comparison of interoperability relations

According to the rigor required for interoperability testing, we need to know which interoperability relation to use. In this section, we give a comparison between the specification-based interoperability relations defined in Sections 4.2.1, 4.2.2 and 4.2.3, in terms of their power of non-interoperability detection. Let us call the set of these relations *SBIR,* then the formalization of the comparison of the relations leads to the well-known theory of testing equivalence [6, 13] and the related preorder expressed here by a relation $\sqsubseteq_R \subseteq SBIR \times SBIR$.

Definition 4.6 $\forall\ R_x, R_y \in SBIR$,
$\mathcal{R}_x \sqsubseteq_\mathcal{R} \mathcal{R}_y =_{def} \forall I_1, I_2 \in \mathcal{IOLTS},\ \mathcal{R}_x(I_1, I_2) \Rightarrow \mathcal{R}_y(I_1, I_2)$

So, $R_x \sqsubseteq_R R_y$ means that interoperability tests based on R_x detect more non-interoperable implementations than R_y. By extension, the testing equivalence between two relations will be denoted by $\cong_R$. We note $R_x \not\sim_R R_y$ to say that R_x and R_y are not comparable. It is represented by "-" in the Figure 3 which synthesizes all the comparison of the specification-based interoperability relations. Proofs are given below.

	$\mathcal{R}_1$	$\mathcal{R}_2, \mathcal{R}_3$	$\mathcal{R}_4$	$\mathcal{R}_5, \mathcal{R}_6$	$\mathcal{R}_7$
$\mathcal{R}_8$ ($\cong_\mathcal{R} \mathcal{R}_9$)	$\sqsubseteq_\mathcal{R}$	$\sqsubseteq_\mathcal{R}$	$\sqsubseteq_\mathcal{R}$	$\sqsubseteq_\mathcal{R}$	$\sqsubseteq_\mathcal{R}$
$\mathcal{R}_7$	$\sqsubseteq_\mathcal{R}$	-	$\sqsubseteq_\mathcal{R}$	-	
$\mathcal{R}_5$ ($\cong_\mathcal{R} \mathcal{R}_6$)	-	-	$\sqsubseteq_\mathcal{R}$		
$\mathcal{R}_4$	-	-			
$\mathcal{R}_2$ ($\cong_\mathcal{R} \mathcal{R}_3$)	$\sqsubseteq_\mathcal{R}$				

Figure 3. Comparison of specification-based interoperability relations

4.3.1 Some proofs. The decision of interoperability relies on observations done on the available interfaces. So the set of interfaces can be used to compare interoperability relations. "Lower-oriented" and

"upper-oriented" relations are not comparable because their sets of interfaces are not comparable. Total interoperability relations are stronger than the corresponding upper or lower interoperability relations because their set of observations is greater. Bilateral and Global relations are stronger than their corresponding unilateral relations. Most of the proofs are based on these remarks. Nevertheless, we have observed that complete formal proofs of $R_2 \cong_R R_3$, $R_5 \cong_R R_6$, and $R_8 \cong_R R_9$ need some intermediate lemmas and propositions. We give them in the following.

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

Proof: Let $(q_1, q_2) \in ((M_1 || M_2)$ after $\sigma)$ and $a \in Out(M_1 || M_2, \sigma)$. Rules of definition 3.4 apply for (q_1, q_2). Thus, we have either $a \in \Sigma^{M_1}$ or $a \in \Sigma^{M_2}$ which means that $a \in Out(M_1, \sigma/\Sigma^{M_1}) \cup Out(M_2, \sigma/\Sigma^{M_2})$. In the other sense, it is easy to see that $Out(M_1, \sigma/\Sigma^{M_1}) \cup Out(M_2, \sigma/\Sigma^{M_2}) \subseteq Out(M_1 \| M_2, \sigma)$. Thus, we have $Out(M_1 \| M_2, \sigma) = Out(M_1, \sigma/\Sigma^{M_1}) \cup Out(M_2, \sigma/\Sigma^{M_2})$. ◊

Lemma 4.2 *Let* $M_1, M_2 \in IOLTS$, *and* $\sigma \in Traces(M_1 \| M_2)$,
$$Out_{\Sigma_L^{M_1 \| M_2}}(M_1 \| M_2, \sigma) = Out_{\Sigma_L^{M_1} \cup \Sigma_L^{M_2}}(M_1 \| M_2, \sigma)$$

Proof: Using the definition 3.4 of the composition ||, we have $\Sigma_L^{M_1 \| M_2} \subseteq \Sigma_L^{M_1} \cup \Sigma_L^{M_2} \Rightarrow Out_{\Sigma_L^{M_1 \| M_2}}(M_1 \| M_2, \sigma) \subseteq Out_{\Sigma_L^{M_1} \cup \Sigma_L^{M_2}}(M_1 \| M_2, \sigma)$. Now, let $a \in Out_{\Sigma_L^{M_1} \cup \Sigma_L^{M_2}}(M_1 \| M_2, \sigma)$., then $\sigma\, a \in Traces(M_1 || M_2)$. Thus, $a \in \Sigma_L^{M_1 \| M_2}$ and $Out_{\Sigma_L^{M_1} \cup \Sigma_L^{M_2}}(M_1 \| M_2, \sigma) \subseteq Out_{\Sigma_L^{M_1 \| M_2}}(M_1 \| M_2, \sigma)$. ◊

Proposition 4.1 $R_2 \sqsubseteq_R R_3$.

Proof: Let $I_1, I_2, S_1, S_2 \in IOLTS$ such that $R_2(I_1, I_2)$. Let $\sigma \in Traces(S_1 || S_2)$ such that $\sigma \in Traces(I_1 || I_2)$. Let us consider $\sigma_1 = \sigma/\Sigma^{I_1}$ and $\sigma_2 = \sigma/\Sigma^{I_2}$, we can notice that $\sigma_1 \in Traces(S_1)$ and $\sigma_2 \in Traces(S_2)$. Using the definition of $R_2(I_1, I_2)$, we have $Out_{\Sigma_L^{I_1}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_1}}(S_1, \sigma_1)$ and $Out_{\Sigma_L^{I_2}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_2}}(S_2, \sigma_2)$. Thus, with the definition 3.3, and the classical properties of $\cup$ and $\cap$, we have :
$$Out_{\Sigma_L^{I_1} \cup \Sigma_L^{I_2}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_1} \cup \Sigma_L^{S_2}}(S_1, \sigma_1) \cup Out_{\Sigma_L^{S_1} \cup \Sigma_L^{S_2}}(S_2, \sigma_2).$$
Using the definition of σ_1 and σ_2, we have :
$Out_{\Sigma_L^{I_1} \cup \Sigma_L^{I_2}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_1} \cup \Sigma_L^{S_2}}(S_1, \sigma/\Sigma^{I_1}) \cup Out_{\Sigma_L^{S_1} \cup \Sigma_L^{S_2}}(S_2, \sigma/\Sigma^{I_2})$
$\Leftrightarrow Out_{\Sigma_L^{I_1} \cup \Sigma_L^{I_2}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_1} \cup \Sigma_L^{S_2}}(S_1 \| S_2, \sigma)$ (lemma 4.1)
$\Leftrightarrow Out_{\Sigma_L^{I_1 \| I_2}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_1 \| S_2}}(S_1 \| S_2, \sigma)$ (lemma 4.2)
Thus, $\forall \sigma \in Traces(S_1 || S_2)$, $Out_{\Sigma_L^{I_1 \| I_2}}(I_1 \| I_2, \sigma) \subseteq Out_{\Sigma_L^{S_1 \| S_2}}(S_1 \| S_2, \sigma)$ which proves: $R_2 \sqsubseteq_R R_3$. ◊

Proposition 4.2 $R_3 \sqsubseteq_R R_2$.

Proof: Let $I_1, I_2, S_1, S_2 \in IOLTS$ such that $R_3(I_1, I_2)$ and let us prove that it implies $R_1(I_1, I_2)$. As $R_1(I_1, I_2)$ does not refer to the specification of I_2, we can consider that $S_2 = I_2$. Let $\sigma_1 \in Traces(S_1)$, and $\sigma \in Traces(I_1 \| I_2), \sigma/\Sigma^{I_1} = \sigma_1$. As $\sigma_1 \in Traces(S_1)$ and $S_2 = I_2$, we have $\sigma \in Traces(S_1 \| S_2)$. Using the fact that $R_3(I_1, I_2)$ gives $Out_{\Sigma^{I_1 \| I_2}}(I_1 || I_2, \sigma) \subseteq Out_{\Sigma^{S_1 \| S_2}}(S_1 || S_2, \sigma)$. We want to prove that $Out_{\Sigma^{I_1}}(I_1 || I_2, \sigma) \subseteq Out_{\Sigma^{S_1}}(S_1, \sigma_1)$.

As $\Sigma^{I_1} \cap \Sigma^{I_2} = \emptyset$ and $\Sigma^{S_1} \cap \Sigma^{S_2} = \emptyset$ and considering only the outputs of I_1, we have: $Out_{\Sigma^{I_1}}(I_1 || I_2, \sigma) \subseteq Out_{\Sigma^{S_1}}(S_1 || S_2, \sigma)$. Using the definition 3.3, lemmas 4.1 and 4.2 gives the following equations $Out_{\Sigma^{S_1}}(S_1 || S_2, \sigma) = Out_{\Sigma^{S_1}}(S_1, \sigma/\Sigma^{S_1}) \cup Out_{\Sigma^{S_1}}(S_2, \sigma/\Sigma^{S_2}) = Out_{\Sigma^{S_1}}(S_1, \sigma/\Sigma^{S_1}) = Out_{\Sigma^{S_1}}(S_1, \sigma_1)$. Thus, $Out_{\Sigma^{I_1}}(I_1 || I_2, \sigma) \subseteq Out_{\Sigma^{S_1}}(S_1, \sigma_1)$. So, $\forall \sigma_1 \in Traces(S_1), \forall \sigma \in Traces(I_1 || I_2), \sigma/\Sigma^{I_1} = \sigma_1$, $Out_{\Sigma^{I_1}}(I_1 || I_2, \sigma) \subseteq Out_{\Sigma^{S_1}}(S_1, \sigma_1)$ and $\mathcal{R}_3(I_1, I_2) \Rightarrow \mathcal{R}_1(I_1, I_2)$.

Using the fact that R_3 is symmetrical, we have $R_3(I_1, I_2) \Rightarrow R_1(I_2, I_1)$. Thus, $\mathcal{R}_3(I_1, I_2) \Rightarrow (\mathcal{R}_1(I_1, I_2) \wedge \mathcal{R}_1(I_2, I_1)) = \mathcal{R}_2(I_1, I_2)$. ◊

Theorem 4.1 $R_3 \cong_R R_2$, $R_6 \cong_R R_5$, *and* $R_9 \cong_R R_8$

Proof: The first proof is achieved with the proposition 4.1 and the proposition 4.2. The two other proofs are achieved with similar lemmas adapted to upper and total contexts. This means that the global and bilateral interoperability relations are equivalent. ◊

5. INTEROPERABILITY TESTS GENERATION

Several methods have been proposed to generate interoperability tests [2, 14, 15, 16]. Now that we have identified the interoperability architectures and we have defined possible interoperability relations, we give in this section some guidelines to derive interoperability tests.

How to choose interoperability relations?
Depending on several parameters (accessibility of interfaces of the implementations, problems of confidentiality, etc), different architectures can be used for interoperability testing. In Section 2.1, we have identified the four possible classes of interoperability testing architectures. An architecture determines the possible interoperability relations which can be used. As proved in Section 4.3, the power of the generated tests depends on the chosen relation. In the following, we give elements which help in choosing appropriate interoperability relations.

▷ For unilateral testing architectures, the choice of interoperability relations depends on available interfaces (upper, lower or both).

▷ As bilateral and global interoperability relations are equivalent (see theorem 4.1), it can be more interesting to use a bilateral interoperability relation in a global testing architecture. Indeed, this choice avoids the often difficult and error-prone task of designing test coordination procedures which are required when using global interoperability relations.

▷ Hybrid architectures can be tested by generalizing interoperability relations to hybrid cases, or by using a combination of existing relations. For example, architectures where one lower interface is unavailable can be tested using either a combination of R_4 (unilateral upper) relation and R_7 (unilateral total) relation, or a combination of R_6 and R_1 (global upper and unilateral lower) relations.

Can conformance tests be used for interoperability?
There is still considerable debate about the value of conformance testing as a means of achieving interoperability [1, 16]. Let us consider the **ioconf** relation. Proposition 5.1 suggests that an implementation tested with conformance relation **ioconf** does not need to be tested again when the R_7 interoperability relation is used.

Proposition 5.1 *Let* $I_1, I_2 \in IOLTS$, I_1 **ioconf** $S_1 \Rightarrow R_7(I_1, I_2)$

Proof: Let us consider $I_1, I_2 \in IOLTS$ and $S_1 \in IOLTS$ the specification on which I_1 is based. I_1 **ioconf** S_1 implies $\forall \sigma_1 \in Traces(S_1)$, $Out(I_1, \sigma_1) \subseteq Out(S_1, \sigma_1)$. Let us consider a trace $\sigma \in Traces(I_1 \| I_2)$ such that $\sigma / \Sigma^{I_1} = \sigma_1$. Using lemma 4.1 and properties of the projection, we have $Out_{\Sigma^{I_1}}(I_1 \| I_2, \sigma) = Out(I_1, \sigma_1)$. Thus, as $Out(I_1, \sigma_1) \subseteq Out(S_1, \sigma_1)$, we have I_1 **ioconf** $S_1 \Rightarrow \mathcal{R}_7(I_1, I_2)$. ◊

The problem is that conformance testing is not exhaustive. Thus, the result of Proposition 5.1 is "theoretical". On the other hand, proposition 5.2 says that an implementation can be considered R_7-interoperable with another one without being conformant to its specification.

Proposition 5.2 *Let* $I_1, I_2 \in IOLTS$, $\mathcal{R}_7(I_1, I_2) \not\Rightarrow I_1$ **ioconf** S_1

Proof: It is based on the fact that : $Traces(I_1 \| I_2) / \Sigma^{I_1} \subseteq Traces(I_1)$. Thus, it can exist a trace $\sigma_1' \in Traces(I_1) \setminus Traces(I_1 \| I_2) / \Sigma^{I_1}$, such that $R_7(I_1, I_2)$ but not I_1 **ioconf** S_1. ◊

We have similar results for other interoperability relations. Given the complexity of standards and the limits on exhaustive conformance testing, interoperability testing is seen to be a practical requirement. In particular, it is used to uncover incompatibilities (such as incompatible options, coding, etc) even when both implementations have successfully undergone (whatever rigorous) conformance tests. It is a matter of increasing the confidence in the real interoperability between implementations.

How to generate interoperability tests

Interoperability relations defined in Section 4.2 can be written in the shape of "parameterized" conformance relations. Indeed, interoperability relations consider the interaction between implementations, rather than a separately considered implementation in conformance relations. Thus, parameters to be introduced in conformance relations (like **ioconf**) in order to obtain interoperability relations are implementations (resp. specifications) with which the considered implementation (resp. specification) has to interact and the available interfaces. Thus, one may wonder if existing automatic test generation tools like TGV [9] can be used and/or adapted for automatically generating interoperability tests. The general answer to this question needs some further

Theorems 4.1 which prove that bilateral and global interoperability relations are equivalent hints that we may avoid the construction of the composition of the specifications. It is well-known that this construction (even if it is done on-the-fly like in TGV) is a bottleneck for most of the existing tools. This is our current work and first results are promising.

6. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a formal framework (architectures, interoperability relations, test generation techniques) for interoperability testing. We show that the existing concepts of conformance testing can be used to give a formal definition of the notion of interoperability. Different interoperability testing architectures are proposed and discussed. We have defined several interoperability relations based on the existing implementation relations defined for conformance testing. A comparison of these relations is given, in terms of their power to detect non-interoperability. This comparison suggests that the generation of interoperability tests do not necessarily need to build the whole interaction between the specifications of the implementations. This paper ends with some guidelines to help in generating interoperability tests.

As future work, we will focus on quiescence management in the interoperability relations. Based on the defined interoperability relations and the results of their comparison presented in this paper, we will investigate more deeply the different methods and associated algorithms in order to automatically and efficiently generate interoperability tests.

REFERENCES

[1] O. Rafiq and R. Castanet. From conformance testing to interoperability testing. In *Protocol Test Systems,* volume III, pages 371–385, North-Holland, 1991. IFIP, Elsevier sciences publishers B. V.

[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, 1994. IFIP, Elsevier Science B.V.

[3] T. Walter and B. Plattner. Conformance and interoperability a critical assessment. Technical Report 9, Computer engineering and networks laboratory (TIK), Swiss federal institute of technology Zurich, 1994.

[4] J.P. Baconnet, C. Betteridge, G. Bonnes, F. Van den Berghe, and T. Hopkinson. Scoping further EWOS activity for interoperability testing. Technical Report EGCT/96/130 R1, EWOS, 1996.

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

[6] E. Brinksma, R. Alderden, J. Langerak, R. Van de Lagemaat, and J. Tretmans. A Formal Approach to Conformance Testing. In J. De Meer, L. Mackert, and W. Effelsberg, editors, *Second International Workshop on Protocol Test Systems,* pages 349–363, North Holland, 1990.

[7] M. Phalippou. *Relations d'implantations et Hypothèses de test sur les automates à entres et sorties.* PhD thesis, Université de Bordeaux, France, 1994.

[8] J. Tretmans. *A formal approach to conformance testing.* PhD thesis, University of Twente, Enschede, The Netherlands, 1992.

[9] 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,* 1997.

[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,* vol. 11, pages 149–174. IFIP, Kap, 1998.

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

[12] G. Bochmann, R. Dssouli, and J. Zhao. Trace analysis for conformance and arbitration testing. *IEEE Trans. on Software Eng.,* 15(11):1347–1356, 1989.

[13] L. Verhaard, J. Tretmans, P. Kars, and E. Brinksma. On Asychronous Testing. In G. Von Bochman, R. Dssouli, and A. Das, editors, *Fifth International Workshop on Protocol Test Systems,* pages 1–13, North Holland, 1993. IFIP Transactions.

[14] N. Arakawa, M. Phalippou, N. Risser, and T. Soneoka. Combination of conformance and interoperability testing. *FORTE'92,* V(C-10):397–412, 1992.

[15] S. Kang and M. Kim. Test sequence generation for adaptive interoperability testing. In A. Cavally and S. Budkowski, editors, 8th *International Workshop on Protocol Test Systems,* pages 193–206, Evry, France, 1995. IFIP.

[16] K. Myungchul, K. Gyuhyeong, and D.C. Yoon. Interoperability testing methodology and guidelines. In *Digital Audio- Visual Council, system integration TC,* volume DAVIC/TC/SYS/96/06/006, New York, 1996.

Acknowledgments

Authors wish to thank their colleagues D. Clarke for carefully reading this paper, and C. Jard & T. Jéron for their constructive remarks.