Quiescence Management improves Interoperability Testing

Alexandra Desmoulin, César Viho

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Quiescence Management improves Interoperability Testing

Alexandra Desmoulin and César Viho

N°5530

Mars 2005

Systèmes communicants

ISSN 0249-6399 ISRN INRIA/RR--5530--FR+ENG

Quiescence Management improves Interoperability Testing

Alexandra Desmoulin * and César Viho †

Systèmes communicants
Projet Armor

Rapport de recherche n ° 5530 — Mars 2005 — 18 pages

Abstract: At any level of computer networks, interoperability testing generally deals with several components that communicate while trying to provide a designated service. When a component remains silent, the assigned testing verdict is generally Fail, assuming that its behavior is non-conformant. Sometimes, this silence may be anticipated given the component's specifications. In these cases, the fail verdict is unsatisfactory. In this paper, we show that "quiescence management" improves interoperability testing. Based on formal definitions of interoperability testing, we introduce new definitions that take into account the possible quiescence of components under test. Through several examples and scenarios, we show that these new definitions detect non-interoperability cases with higher precision. Moreover, these new definitions more clearly distinguish specification-driven quiescences from others, leading to unbiased interoperability tests with accurate verdicts.

Key-words: interoperability, test, quiescence, conformance

(Résumé : tsvp)

This paper is an extended version of our paper published in the Proceedings of the 17th IFIP International Conference on Testing of Communicating Systems (Testcom), Mai 28- June 2, 2005, Montreal, Canada.

* Alexandra.Desmoulin@irisa.fr
† viho@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

La gestion des blocages améliore le test d'interopérabilité

Résumé : Dans les réseaux informatiques, le test d'interopérabilité concerne différents composants qui communiquent les uns avec les autres tout en rendant un service défini. Quand un des composants reste bloqué, le verdict de test correspondant est en général Fail, considérant que le comportement du composant est non-conforme. Dans certains cas, ce blocage peut être autorisé dans la spécification du composant. Le verdict Fail n'est alors pas satisfaisant. Dans cet article, nous montrons que la gestion des blocages (quiescence management) améliore le test d'interopérabilité. A partir de définitions formelles du test d'interopérabilité, nous introduisons de nouvelles définitions qui prennent en compte les blocages possibles des composants testés. A travers différents exemples et scénarios, nous montrons que ces nouvelles définitions détectent avec plus de précision les cas de non-interopérabilité. De plus, ces nouvelles définitions différencient plus clairement les blocages prévus dans les spécifications des autres, permettant des tests d'interopérabilité non-biaisés.

Mots-clé : interopérabilité, test, blocages, conformité

1 Introduction

Different methods have been developed to test network components. Among these methods, we will focus on conformance and interoperability testing. Conformance testing evaluates the ability of a component to behave as described in its specification, generally a standard. Interoperability testing deals with the ability of two or more components to interact in an operational environment. This notion can be intuitively defined by the capacity of two or more components to behave as described in their specification during their interaction, to communicate correctly together, and to provide the foreseen service.

Conformance testing is precisely characterized : testing architectures and conformance relations [1, 2, 3, 4] were defined. This allows automatic test generation and execution. This is not the case for interoperability testing although some definitions exist in [5, 6, 7]. Two main reasons explain the current situation : interoperability is more often regarded as being a practical requirement that conformance is. Yet conformance testing is also considered as being prerequisite to the achievement of interoperability.

Conformance and interoperability concern the same objects (implementations, specifications, etc). For this reason, the different attempts to define the notion of interoperability use the concepts and theory defined for conformance testing. In [5], interoperability testing architectures and *interoperability relations* were defined. An interoperability relation defines the conditions that two implementations must satisfy to be considered interoperable. These definitions do not manage possible quiescence of implementations. This leads to incorrect verdicts during testing. Based on the interoperability relations defined in [5], new interoperability relations with quiescence management have been defined. We show that these new relations can help in solving this problem.

This paper is structured as follows. First the model and notations used for the interoperability definitions are presented in Section 2. In Section 3, we summarize the interoperability definitions of [5]. Some testing results obtained with these definitions are presented in Section 4. The new interoperability relations with quiescence management are defined in Section 5. Then, the new testing results with these relations are presented in Section 6 showing the contribution of quiescence management in interoperability testing. Finally, conclusion and future work are to be found in Section 7.

2 Model and notations

The model used to provide formal interoperability definitions, and which we consequently use, is the model of the IOLTS (Input-Output Labeled Transition System) [4]. We use it to model specifications. As usual in the black-box testing context, we also need to model imple-

mentations, even if their behaviors are supposedly unknown. They will also be represented by an IOLTS.

2.1 IOLTS model

Definition 2.1 *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. We note* p?a *for an input event and* p!a *for an output event with* p *as an interaction point on which the event is executed and* a *as the message.*
- $\Delta^M \subseteq Q^M \times (\Sigma^M \cup \tau) \times Q^M$ *is the transition relation, where* $\tau \notin A^M$ *denotes an internal event. We note* $q \xrightarrow{\alpha}_M q'$ *for* $(q, \alpha, q') \in \Delta^M$.

Let us consider an IOLTS M, and let $\alpha \in \Sigma^{\text{M}}$ with $\alpha = p.\{?,!\}.m$, $\mu_i \in \Sigma^{\text{M}} \cup \tau$, $\sigma \in (\Sigma^{\text{M}})^*$, $q, q', q_i \in Q^{\text{M}}$:

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

2.2 Some definitions

In interoperability testing, we usually need to observe some specific events among all possible traces of an IUT. These traces, reduced to the expected messages, can be obtained by a projection of those traces on a set. This latter being used to select the expected events.

Definition 2.2 *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 $\alpha \in X$.*

Definition 2.3 (Projection of an IOLTS on a set) *Let us consider an IOLTS $M = (Q, \Sigma, \Delta, q_0)$, a set X. The projection of M on the set of events X is noted by M/X and is defined by :*

- $M_X = (Q, \Sigma_X, \Delta(X), q_0)$
  $\forall (q_1, a, q_1') \in \Delta,\ a \in X,\ (q_1, a, q_1') \in \Delta(X),\ a \in \Sigma_X$
  $\forall (q_1, a, q_1') \in \Delta,\ a \notin X,\ (q_1, \tau, q_1') \in \Delta(X),\ a \notin \Sigma_X$
- $M/X = (M/X, \Sigma_{M/X}, \Delta_{M/X}, q_0^X)$ *is the IOLTS M_X obtained after determinization :*
  - $Q_{M/X} = 2^Q$
  - $\Sigma_{M/X} = \Sigma \setminus \{a \in \Sigma | a \notin \Sigma_X\}$.
  - $q_0^X = q_0$ *after* ϵ
  - $\Delta_{M/X}$ *is obtained as :* $(p, a, p') \in \Delta_{M/X}$ *if* $p = p'$ *after* a, *with* $p,\ p' \in 2^Q$ *and* $a \in \Sigma_{M/X}$.

Interoperability testing concerns the interaction of two or more implementations. In order to provide a formal definition of interoperability, we need to model interaction. This is done in the definition 2.4. In this definition, Σ_U and Σ_L are the set of events on the different interaction points as described in the testing architecture (figure 1 of section 3.1).

Definition 2.4 (Synchronous interaction $\|_S$) *The synchronous interaction of two IOLTS M_1 and M_2 is noted $M_1 \|_S M_2 = (Q^{M_1} \times Q^{M_2},\ \Sigma^{M_1\|_S M_2},\ \Delta^{M_1\|_S M_2},\ (q_0^{M_1}, q_0^{M_2}))$ with $\Sigma^{M_1\|_S M_2} \subseteq \Sigma^{M_1} \cup \Sigma^{M_2}$, and the transition relation $\Delta^{M_1\|_S M_2}$ is obtained as follows :* $\forall (q_1, q_2) \in Q^{M_1} \times Q^{M_2}$,

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

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

3 Summary of quiescence-less interoperability relations

Interoperability testing can be defined as a set of procedures used to verify if two or more implementations interact correctly. This test is not precisely characterized as conformance testing and is often considered as a pragmatic and a practical requirement. But different attempts to define interoperability exist [5, 8, 9, 7, 10, 6]. For the quiescence management, we used interoperability definitions of [5] called *interoperability relations*. These relations are based upon **ioconf** conformance relation and do not manage quiescence. These relations consider the testing architecture presented in section 3.1 and are presented in Section 3.2.

3.1 Test architectures

In order to provide a formal definition of interoperability testing, we have taken into consideration the general testing architecture of figure 1. Different architectures may be obtained from this architecture as described in [11, 8, 7, 12].

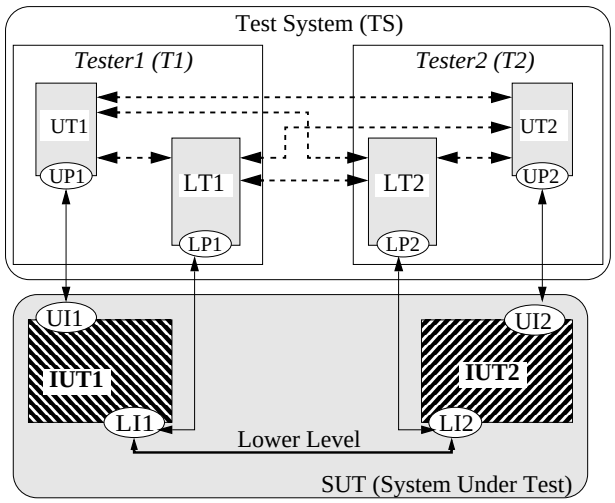

Figure 1: General architecture of interoperability testing

This testing architecture is composed of two interacting IUTs. Each of these two IUTs has two kind of interfaces : UI_i and LI_i which are the Upper Interfaces and the Lower Interfaces through which the implementation communicates with its upper and lower layers. Testers are linked to these interfaces : UT_i (Upper Tester) and LI_i (Lower Tester). Depending on the accessibility of the interfaces, these testers can or can not exist. Thus, we obtained different testing architectures. The *unilateral*, *bilateral* and *global* interoperability testing architectures respectively correspond to the architecture with testers which observe/control interfaces of a unique implementation, both implementations separately or

both implementations together. We can also distinguish architectures according to the accessibility of upper or lower interfaces. In this paper, we only consider the case of the accessibility of both interfaces : this architecture is called *total*.
With this architecture, the set Σ^M of observable events of the definition 2.1 can be decomposed as follows : $\Sigma^{\text{M}} = \Sigma^{\text{M}}_U \cup \Sigma^{\text{M}}_L$, where Σ^{M}_U (resp. Σ^{M}_L) is the set of messages exchanged on the upper (resp. lower) interface. Σ^{M} can be also decomposed in order to distinguish input messages from output messages. $\Sigma^{\text{M}} = \Sigma^{\text{M}}_I \cup \Sigma^{\text{M}}_O$, where Σ^{M}_I (resp. Σ^{M}_O) is the finite set of input (resp. output) messages.

3.2 Interoperability relations

In [5], different *interoperability relations* have been defined. These relations formally specify conditions to be satisfied by two implementations in order to be considered interoperable. These interoperability relations are based upon a conformance relation : the **ioconf** conformance relation defined in [4] as follows

$$I \textbf{ ioconf } S =_\Delta \forall \sigma \in Traces(S),\ Out(I,\sigma) \subseteq Out(S,\sigma) \quad .$$

The interaction considered is asynchronous : $M_i \| M_j = M_i \|_{\text{s}} \mathcal{E} \|_{\text{s}} M_j$ where $\mathcal{E}$ represents the asynchronous environment between the two IOLTS.

3.2.1 Definitions of the interoperability relations without quiescence management

Different interoperability relations were defined depending of the considered testing architecture and thus, of the access on the different interfaces. The *unilateral total interoperability relation* $\mathcal{R}_1$ consider the case where we have only access to one IUT. This relation is based on the fact that, during the interaction between I_1 and I_2, the least we can expect from the implementation I_1 is to behave as expected according to its specification S_1.

Definition 3.1 (Unilateral Total Interoperability Relation $\mathcal{R}_1$)

$$\mathcal{R}_1(I_1,I_2) =_\Delta \forall \sigma_1 \in Traces(S_1), \forall \sigma \in Traces(S_1 \| S_2),\ \sigma/\Sigma^{S_1} = \sigma_1 \Rightarrow Out((I_1 \| I_2)/\Sigma^{I_1}, \sigma) \subseteq Out(S_1, \sigma_1).$$

The relation $\mathcal{R}_1$ can be applied independently to I_2 (based on the specification S_2). The *bilateral lower interoperability relation* corresponds to the relation $\mathcal{R}_1$ applied for both I_1 and I_2.

Definition 3.2 (Bilateral Total Interoperability relation $\mathcal{R}_2$)

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

The *global total interoperability relation* $\mathcal{R}_3$ is based on the global behavior of the interactions between respectively : specifications $S_1 \| S_2$ and implementations $I_1 \| I_2$.

Definition 3.3 (Global Total Interoperability relation $\mathcal{R}_3$)
$\mathcal{R}_3(I_1, I_2) =_\Delta \forall \sigma \in Traces(S_1 \| S_2)$, $Out(I_1 \| I_2, \sigma) \subseteq Out(S_1 \| S_2, \sigma)$.

Remark : In [5], the formal interoperability relation definitions do not correspond to their literal definitions. Indeed, different relations have been defined corresponding to the different possible testing architectures. Thus, the interoperability relations must consider only events observable with the corresponding architecture during testing. But the interoperability relations were written in such a way that the traces also include non-observable events. For this reason, the formal definition of the interoperability relations were rewritten. The interoperability relations presented above are the corrected relations.

The properties of the interoperability relations proved in [5] are still true because the proofs were based on the literal definitions of the relations. Some of these properties are :

- $\mathcal{R}_3 \cong_{\mathcal{R}} \mathcal{R}_2$: this equivalence suggests that we may avoid the construction of the interaction of the specification.
- I_1 **ioconf** $S_1 \Rightarrow \mathcal{R}_1(I_1, I_2)$, and I_1 **ioconf** $S_1 \wedge I_2$ **ioconf** $S_2 \Rightarrow \mathcal{R}_2(I_1, I_2) = \mathcal{R}_3(I_1, I_2)$: two implementations conformant to their specification in the sense of **ioconf** are considered interoperable with these interoperability relations.

4 Interoperability testing without quiescence management : some examples

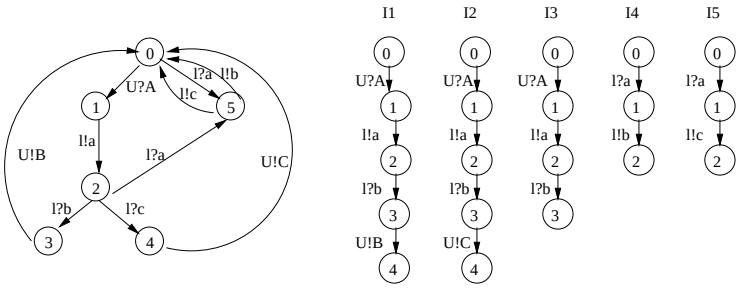

Figure 2: Specification S and implementations I_1, I_2, I_3, I_4 and I_5

On the example of the figure 2, let us consider these four interactions : I_1 with I_4, I_2 with I_4, I_3 with I_4, and I_1 with I_5. The results with the interoperability relations on these interactions are :

- For I_1 and I_4, we have : $\mathcal{R}_1(I_1, I_4)$, $\mathcal{R}_1(I_4, I_1)$, $\mathcal{R}_2(I_1, I_4)$ and $\mathcal{R}_3(I_1, I_4)$.
- For I_2 and I_4, we have : $\neg\mathcal{R}_1(I_2, I_4)$, $\mathcal{R}_1(I_4, I_2)$, $\neg\mathcal{R}_2(I_2, I_4)$ and $\neg\mathcal{R}_3(I_2, I_4)$.

- For I_3 and I_4, we have : $\mathcal{R}_1(I_3, I_4)$, $\mathcal{R}_1(I_4, I_3)$, $\mathcal{R}_2(I_3, I_4)$ and $\mathcal{R}_3(I_3, I_4)$.
- For I_1 and I_5, we have : $\mathcal{R}_1(I_1, I_5)$, $\mathcal{R}_1(I_5, I_1)$, $\mathcal{R}_2(I_1, I_5)$ and $\mathcal{R}_3(I_1, I_5)$.

This last result is unsatisfactory given I_5 sends a message that is unexpected in I_1. With an intuitive definition of interoperability, I_1 and I_5 should be considered non-interoperable.

Given the test architecture considered, the interoperability scenario (for each interaction) begins with the tester T_1 sending A to the upper interface of I_1 (or I_2). Then, the testers can not control the scenarios but only observe the message sent and received on the lower interfaces (communication between the two IUT). Testers can also receive messages sent by the IUT on its upper interface.

Notation : for the scenario description, the events in the traces are noted :

• For the exchange between a tester and an implementation $U_x\{!,?\}m$ where x is the number of the concerned IUT, $\{?,!\}$ the kind of the message from the point of view of the IUT, and m the message.

• For the exchange between the two implementations in interaction, the sending and the reception are modeled as explained in the definition 2.1 (cf. Section 2.1) with the number of the IUT concerned.

Thus the scenarios of interaction are :

1. For I_1 and I_4, we have : $U_1?A.l_1!a.l_4?a.l_4!b.l_1?b.U_1!B$.
2. For I_2 and I_4, we have : $U_2?A.l_2!a.l_4?a.l_4!b.l_2?b.U_2!C$.
3. For I_3 and I_4, we have : $U_3?A.l_3!a.l_4?a.l_4!b.l_3?b$.
4. For I_1 and I_5, we have : $U_1?A.l_1!a.l_5?a.l_5!c$ (with no reception of c by I_1).

For the third scenario (interaction of I_2 and I_4), the verdict of the test (when testing $\mathcal{R}_1(I_2, I_4)$ or $\mathcal{R}_3(I_2, I_4)$) is FAIL because of the output $U_2!C$ which is not allowed in the specification S_2 after the trace $U_2?A.l_2!a.l_4?a.l_4!b.l_2?b$ (only $U_2!B$ is allowed after this trace).

For all other scenarios, the verdicts are also FAIL whereas the corresponding interoperability relations are verified. The reason is the absence of quiescence management in the tests. Indeed, quiescence is observed with timers : after each event a timer is started and a situation of quiescence is observed if a timeout occurs (the timer is restarted after each other event). All the scenarios presented terminate : after the last event takes place, the implementation does not return to the initial state. Thus, after the last event of the scenario, a timer is started. As there is no other event that can occur, a timeout is observed : the verdict is FAIL because this timeout (and quiescence corresponding) is considered as an output of the implementations in interaction. But this quiescence can be foreseen in the specifications. In this case, the verdict must not be FAIL. For this reason, it is necessary to manage quiescence in interoperability relations.

5 Quiescence management

To manage quiescence, we need to model this kind of event. The definition 2.1 of the IOLTS does not model quiescence. This is done in Section 5.1. Then, the operations on the IOLTS used in the interoperability relations are rewritten with quiescence management in Sections 5.2 and 5.3. Finally, the interoperability relations with quiescence management are defined section 5.4.

5.1 Quiescence and suspensive IOLTS

Three main situations lead to quiescence of a system :

- 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 are observed. This is noted : $q \in outputlock(M) =_\Delta \Gamma(q) \subseteq \Sigma_I^M$.
- A livelock corresponds to a loop of internal events : $q \in livelock(M) =_\Delta \exists \tau_1, \cdots, \tau_n, q \stackrel{\tau_1,\cdots,\tau_n}{\rightarrow} q$.

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 $q \stackrel{\delta}{\rightarrow}_M q$ where δ is treated as an observable output event. The obtained IOLTS is called suspensive IOLTS [13, 2] and is noted $\Delta(M)$.

To study quiescence management in the interoperability relations, we consider the conformance relation **ioco** [13].

Definition 5.1 I **ioco** $S =_\Delta \forall \sigma \in STraces(S)(= Traces(\Delta(S)))$, $Out(\Delta(I), \sigma) \subseteq Out(\Delta(S), \sigma)$

Quiescence management in some operations used in the interoperability relations of [5] needs to be studied. These operations are the projection of an IOLTS on a set and the interaction between implementations.

5.2 Projection with quiescence

To calculate the projection of an IOLTS M on a set X, the problem is to preserve information on all quiescent states. The steps to calculate this projection are :

1. Calculation of $\Delta(M) = (Q, \Sigma_\delta, \Delta_{\Delta(M)}, q_0)$ or of $M = M_1 \| M_2$ in the case of an interaction
2. Substitution of events of $\bar{X}$ by internal events. Two cases can be differenciated if we consider an IOLTS from an interaction or another type of IOLTS.

- for an IOLTS without $\delta(1)$ or $\delta(2)$ (that does not come from an interaction), t he result of this step is the IOLTS $X(\Delta(M))$=(Q, $\Sigma_{X(\Delta(M))}$, $\Delta_{X(\Delta(M))}$, q_0) with $\Sigma_{X(\Delta(M))} = \Sigma_\delta \setminus \Sigma_{\bar{X}}$ and $\Delta_{X(\Delta(M))}$ obtained with the rules :
  - $\forall(q, \delta, q) \in \Delta_{\Delta(M)}$, $(q, \delta, q) \in \Delta_{X(\Delta(M))}$
  - $\forall(q, a, q') \in \Delta_{\Delta(M)}$, $a \in X$, $(q, a, q') \in \Delta_{X(\Delta(M))}$
  - $\forall(q, a, q') \in \Delta_{\Delta(M)}$, $a \in \bar{X}$, $(q, \tau, q') \in \Delta_{X(\Delta(M))}$
- for an IOLTS from an interaction, the most difference is in the treatment of the events $\delta(1)$ and $\delta(2)$. If X is made of events of M_1, the events $\delta(1)$ have to be modeled in the result. The rule is $\forall(q, \delta(1), q) \in \Delta_{\Delta(M)}$, $(q, \delta, q) \in \Delta_{X(\Delta(M))}$. $\delta(2)$ is traited as an event of $\bar{X}$.

3. Calculation of livelocks : these livelocks can be due to the precedent step which had τ-events.

4. Determinization

The steps 2 and 4 are the two steps of the calculation of the definition 2.3. The steps 1 and 3 are necessary to preserve all information on quiescence.

5.3 Interaction with quiescence

The method chosen to calculate the interaction of two IOLTS with quiescence management is a method with calculation of the suspensive IOLTS followed by the calculation of the interaction. The steps to calculate the interaction with quiescence on M_1 and M_2 are :

1. Calculation of $\Delta(M_1)$ and $\Delta(M_2)$.

2. Then the following rules are applied :

   - Rules (1) and (2) of the definition 2.4 of the Section 2.2 i.e. propagation of events on the upper interface (rule (1)) and mapping of events on the lower interfaces (rule (2)).
   - propagation of quiescence modeled in the two IOLTS : a quiescent state is noted $(q_1, q_2) \overset{\delta(1)}{\rightarrow}_{\text{M}} (q_1', q_2')$ if $(q_1 \overset{\delta}{\rightarrow}_{\text{M}} q_1') \in \Delta(M_1)$, $(q_1, q_2) \overset{\delta(2)}{\rightarrow}_{\text{M}} (q_1', q_2')$ if $(q_2 \overset{\delta}{\rightarrow}_{\text{M}} q_2') \in \Delta(M_2)$, and we have $(q_1, q_2) \overset{\delta}{\rightarrow}_{\text{M}} (q_1', q_2')$ if $((q_1, q_2) \overset{\delta(1)}{\rightarrow}_{\text{M}} (q_1', q_2')) \wedge ((q_1, q_2) \overset{\delta(2)}{\rightarrow}_{\text{M}} (q_1', q_2'))$.
   - an other rule is necessary to model all quiescent states. This rule is applied on some particular states. The transitions starting from such states are labeled with output **and** input on the lower interface. Thus, no quiescence is modeled on the state. But if only the input events can be mapped with output events, quiescence must be modeled in the corresponding state of the interaction.

3. Calculation of all the deadlocks not already modeled.

Remark : Another method to calculate this interaction is the calculation of the interaction with the rules of the definition 2.4 followed by the calculation of quiescence on the interaction. But we observe that some situations of quiescence modeled, which are necessary for quiescence management in interoperability testing, are not modeled with this method. These situations correspond to the case where two kinds of events are possible : inputs on the upper interface of one of the implementations (I_i) and outputs on the upper interface of the other implementation (I_j). In this case, quiescence of I_i can be allowed but not quiescence of I_j. The corresponding $\delta(i)$ is only modeled with the chosen method of interaction calculation.

Notation : In the traces of a scenario, the events of the lower interface were noted $l_a!m.l_b?m$ and the considered interaction was asynchronous. In the following study on interoperability testing with quiescence management of the Section 6, the considered interaction is synchronous. Thus, to model the mapping of the outputs and inputs on the lower interface, we note $l_a!m(l_b?m)$ or $l_a?m(l_b!m)$ for a point of view from I_a and $l_b!m(l_a?m)$ or $l_b?m(l_a!m)$ for a point of view from I_b.

5.4 Interoperability relations with quiescence management

With these operations (projection and interaction with quiescence), new interoperability relations can be defined. The different between these new relations noted $\mathcal{R}_x^\delta$ and the relations of section 3.2.1 is the quiescence management : for example, $\mathcal{R}_1^\delta$ can be deduced from $\mathcal{R}_1$ by using the projection and interaction of sections 5.2 and 5.3.

Definition 5.2 (Unilateral total interoperability relation)
$\mathcal{R}_1^\delta(I_1, I_2) =_\Delta \forall \sigma_1 \in Traces(\Delta(S_1)), \forall \sigma \in Traces(S_1 \|_\delta S_2),\ \sigma/\Sigma^{S_1} = \sigma_1 \Rightarrow$
$Out((I_1 \|_\delta I_2)/\Sigma^{S_1}, \sigma) \subseteq Out(\Delta(S_1), \sigma_1).$

The other interoperability relations with quiescence management can be written in the same way from the interoperability relations of section 3.2.1.

6 Interoperability testing with quiescence management

The different scenarios of interaction presented in Section 4 are studied with quiescence management in this section.

6.1 Interaction between I_1 and I_4

This example of interaction corresponds to the figure 3. Allowed quiescence is modeled on the specification : the concerned states are the states 0 and 2 with outputlocks. Quiescence is also modeled on the IUT and on the interaction of I_1 and I_4. We can notice that this

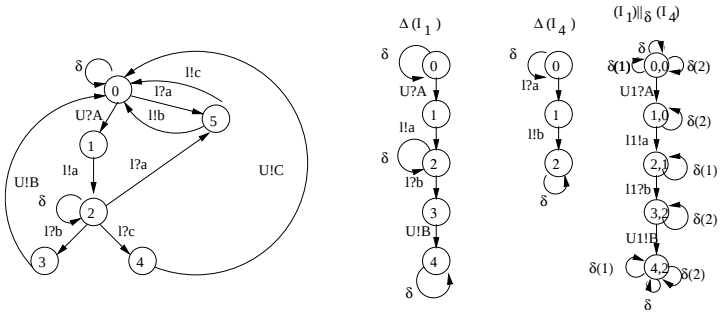

Figure 3: Interaction between I_1 and I_4

interaction ends with a deadlock. The results for the interoperability relations with quiescence management on the interaction of I_1 and I_4 are : $\mathcal{R}_1^\delta(I_1, I_4)$, $\mathcal{R}_1^\delta(I_4, I_1)$, $\mathcal{R}_2^\delta(I_1, I_4)$ and $\mathcal{R}_3^\delta(I_1, I_4)$. All outputs are allowed in the specification, but also all quiescent states. Thus, with the interoperability relations with quiescence management, this result of interoperability is preserved in this case.

The scenario of the interaction of I_1 and I_4 for a unilateral total interoperability relation is : $U1?A.l1!a.l1?b.U1!B$. Then this scenario terminates with a timeout (due to the deadlock at the end of the interaction). But this deadlock is allowed in the specification S_1 : the state 4 of I_1 corresponds to the state 0 of the specification where an outputlock is modeled. The scenario of the interaction of I_1 and I_4 for a global total interoperability relation is : $U1?A.l1!a(l4?b).l1?b(l4!b).U1!B$ followed by a timeout. As the quiescence of the state 0 of S_1 is propagated to the interaction of the two specifications, the deadlock at the end of the scenario is also allowed for this architecture and the scenario based on the corresponding interoperability relation.

Conclusion : As quiescence at the end of the scenario is allowed in the specifications, the verdict of the test is PASS. Thus with quiescence management, the verdict corresponds to the result of the interoperability relations : all the interoperability relations are verified for this interaction, and the verdicts of the test based on these relations are PASS.

6.2 Interaction between I_2 and I_4

The results with the interoperability relations with quiescence management on the interaction of I_2 and I_4 are : $\neg\mathcal{R}_1^\delta(I_2, I_4)$, $\mathcal{R}_1^\delta(I_4, I_2)$, $\neg\mathcal{R}_2^\delta(I_2, I_4)$ and $\neg\mathcal{R}_3^\delta(I_2, I_4)$. The result of non-interoperability is due to the output C on the upper interface of I_2 which is not allowed in S_2 after the executed trace.

The scenario of the interaction of I_2 and I_4 is : $U2?A.l2!a(l4?b).l2?b(l4!b).U2!C$. The verdict of this scenario is FAIL because of the output $U1!C$ which is not allowed in S_1. For the unilateral total architecture in the point of view of I_4, the timeout is allowed in the specification S_4 and the verdict is PASS : $\mathcal{R}_1^{\delta}(I_4, I_2)$.

Conclusion : Quiescence management does not change this verdict of non-interoperability due to a non-authorized output (for the unilateral total architecture in the point of view of I_2 and the global total architecture). In this scenario, the verdicts also correspond to the result of the corresponding interoperability relations.

6.3 Interaction between I_3 and I_4

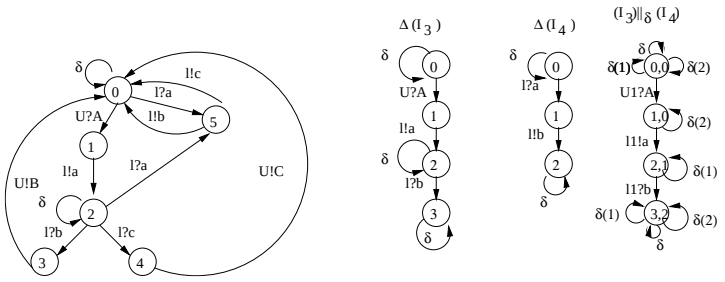

Figure 4: Interaction between I_3 and I_4

For this interaction (cf. figure 4), we have I_3 **ioconf** S but $\neg I_3$ **ioco** S : the deadlock at the end of I_3 is not allowed in the corresponding state (state 3) of S. The results with the interoperability relations with quiescence management on the interaction of I_3 and I_4 are : $\neg\mathcal{R}_1^{\delta}(I_3, I_4)$, $\mathcal{R}_1^{\delta}(I_4, I_3)$, $\neg\mathcal{R}_2^{\delta}(I_3, I_4)$ and $\neg\mathcal{R}_3^{\delta}(I_3, I_4)$.

The scenario of the interaction of I_3 and I_4 is : $U3?A.l3!a(l4?b).l3?b(l4!b)$. The timeout at the end of this scenario does not correspond to a quiescent state of the specification S_3 (but an outputlock exists in the specification of I_4 for the state corresponding to the state 4 of this implementation).

Conclusion : For this scenario, the verdict depends of the tested relation. For a global total interoperability relation or a unilateral total interoperability relation in the point of view of I_3, the verdict is FAIL. This verdict is due to the timeout at the end of the scenario. Indeed, no quiescence is foreseen in this state in the specification S_3 because in this state, I_3 must send the output B on its upper interface. For a unilateral total interoperability relation in the point of view of I_4, the verdict is PASS. Quiescence is allowed in S_4 after the trace $l_4?a.l_4!b$. All these verdicts correspond to the results of the considered interoperability relations for the tests.

6.4 Interaction between I_1 and I_5

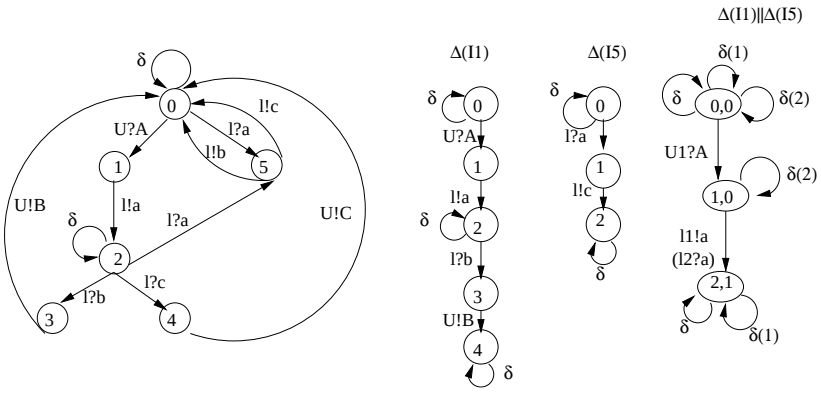

Figure 5: Interaction between I_1 and I_5

This interaction (cf. figure 5) corresponds to a case for which the results with the interoperability relations of [5] were not satisfying. All interoperability relations were verified but the message sent by I_5 does not correspond to the message expected by I_1. The results with the interoperability relations with quiescence management on the interaction of I_1 and I_5 are : $\mathcal{R}_1^\delta(I_1, I_5)$, $\neg\mathcal{R}_1^\delta(I_5, I_1)$, $\neg\mathcal{R}_2^\delta(I_1, I_5)$ and $\neg\mathcal{R}_3^\delta(I_1, I_5)$. These results correspond more to the practical definition and intuitive notion of interoperability.

The scenario of the interaction of I_1 and I_5 is : $U1?A.l1!a(l5?a)$. The message $l5!c$ is not sent by I_5 because in the synchronous context an implementation can not send a message if it is not waited by the implementation in interaction. Thus, the scenario ends after the exchange of the message a between I_1 and I_5 with a deadlock.

Conclusion : For this scenario, the verdict also depends of the tested relation. For a global total interoperability relation or a unilateral total interoperability relation in the point of view of I_5, the verdict is FAIL. This verdict is due to the timeout at the end of the scenario. No quiescence is allowed at the corresponding state of the specification S_5 after the input a : an output must occur. This verdict correspond to the results of the considered interoperability relations : these results are more satisfying because these two implementations are not considered interoperable. For a unilateral total interoperability relation in the point of view of I_1, the verdict is PASS. Quiescence is allowed in S_1 after the trace $U_1?A.l_1!a$. Thus, the non-interoperability is not detected in the point of view of I_1.

6.5 Synthesis and main results

After the study of these interactions, the following properties of interoperability relations with quiescence management can be highlighted :

- With quiescence management, the verdicts of testing scenarios correspond to the results of the considered interoperability relations. This was not the case without quiescence management. Indeed, all timeouts gave a FAIL verdict, but these timeouts can be allowed in the specification and do not correspond to an error in the implementations.

- With quiescence management, we can have two conformant implementations that are not considered interoperable. The interaction of I_1 and I_5 can be taken as example for this property.

- The results for the interoperability relations (and the verdicts of the tests) correspond more to the practical definition and intuitive notion of interoperability. Two implementations considered non-interoperable with the interoperability relations without quiescence management remain non-interoperable with the new interoperability relations. But two other cases of non-interoperable exist with the interoperability relations with quiescence management. The first case corresponds to the non-conformance of one of the implementations due to quiescence not allowed : an example is the interaction of I_3 and I_4 where $\neg I_3$ **ioco** I_4. The second case corresponds to the interaction of an implementation who wants to send a message which is not expected by the implementation in interaction : example of I_1 and I_5. These two cases are no longer considered interoperable with the new interoperability relations and the verdicts of the corresponding tests are FAIL.

This study considered a synchronous interaction between implementations. A point that remains to be studied is the difference between synchronous and asynchronous interaction. This study has already started but is not advanced enough to give formal results. Nevertheless, we give here some observations that seem interesting.

With an asynchronous interaction, the three first scenarios studied above (interaction of I_1 with I_4, I_2 with I_4 and I_3 with I_4) have the same results. But the last scenario (interaction of I_1 with I_5) is different if we consider an asynchronous interaction. Indeed, the message $l5!c$ can be sent by I_5 and is not received by I_1. But the timeout received after this event is foreseen in the specifications, the interoperability relations are verified and the verdict of the test is PASS even though the message c can not be received by I_1.

This latter situation proves that a more formal study is needed to examine the influence of an asynchronous environment on quiescence management in interoperability testing.

7 Conclusion

The goal of the study was to investigate the quiescence management in interoperability testing. Based on a previous work that gives formal definitions of interoperability, we provide new definitions that take into account predictable quiescences of components. Several examples and scenarios show that using these new definitions leads to more accurate verdicts in interoperability testing. The obtained results are more consistent with the intuitive notion of interoperability and practical usage. In light of this information, we can assume that quiescence management improves interoperability testing.

Our study considered a context of two implementations communicating via a synchronous environment. Future work will investigate interoperability criteria with quiescence management in an asynchronous context. We will also study the generalization of these interoperability criteria to a context with more than two implementations.

References

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

[2] Thierry Jéron. Le test de conformité : état de l'art. Rapport pour l'AEE (Architecture Electronique Embarquée), 2001.

[3] 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.

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

[5] 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.

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

[7] 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, September 1996.

[8] 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.

[9] 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.

[10] Machiel van der Bijl, Arend Rensink, and Jan Tretmans. Component based testing with **ioco**. In A. Petrenko and A. Ulrich, editors, *FATES 2003 — Formal Apporaches to Testing of Software*, volume 2931 of *Lecture Notes in Computer Science*, pages 86–100. Springer-Verlag, 2004.

[11] 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.

[12] 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, September 1998.

[13] J. Tretmans. Testing concurrent systems: A formal approach. In J.C.M Baeten and S. Mauw, editors, *CONCUR'99 – 10^{th} Int. Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 46–65. Springer-Verlag, 1999.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399